МАТЕМАТИКА И ПРИКЛАДНАЯ МАТЕМАТИКА

Б.Н. ИВАНОВ

# ДИСКРЕТНАЯ МАТЕМАТИКА

## АЛГОРИТМЫ И ПРОГРАММЫ

## ПОЛНЫЙ КУРС

МОСКВА
ФИЗМАТЛИТ®
2007

УДК 519(075.8)+681.142.2
ББК 32.973.3
И 20

Иванов Б. Н. **Дискретная математика. Алгоритмы и программы. Полный курс.** — М.: ФИЗМАТЛИТ, 2007. — 408 с. — ISBN 978-5-9221-0787-7.

Учебное пособие по курсу дискретной математики. Изложение носит достаточно полный и строгий характер. Теоретические основы курса сопровождаются практически значимыми алгоритмами, реализованными в конкретных компьютерных программах. Книгу можно рассматривать в качестве хорошего справочника методов и алгоритмов дискретной математики, широко применяемых в практическом программировании.

Пособие рассчитано на студентов специальностей, учебные планы которых предполагают изучение каких-либо разделов курса дискретной математики, в первую очередь на математиков-прикладников, а также программистов, занятых разработкой прикладного программного обеспечения.

Рекомендовано Дальневосточным региональным учебно-методическим центром (УМО) в качестве учебного пособия для студентов технических специальностей вузов региона.

Рецензенты:
кафедра математического моделирования и информатики ДВГТУ, зав. кафедрой доктор физико–математических наук, профессор А.А.Буренин;
доктор физико–математических наук, профессор В.В. Катрахов.

ISBN 978-5-9221-0787-7

# ОГЛАВЛЕНИЕ

## Предисловие

Традиционно к дискретной математике относят такие области математического знания, как комбинаторику, теорию чисел, математическую логику, теорию алгебраических систем, теорию графов и сетей и т. д. Дискретная математика всегда оставалась наиболее динамичной областью знаний. Сегодня наиболее значимой областью применения методов дискретной математики является область компьютерных технологий. Это объясняется необходимостью создания и эксплуатации электронных вычислительных машин, средств передачи и обработки информации, автоматизированных систем управления и проектирования. На грани дискретной математики и программирования появляются новые дисциплины, такие как разработка и анализ вычислительных алгоритмов, нечисленное программирование, комбинаторные алгоритмы, алгоритмизация процессов. Курсы по дискретной математике и примыкающие к ним дисциплины изучаются во всех институтах и университетах на отделениях программирования, математики, технических направлениях, экономических и гуманитарных специальностях.

Цель написания учебника — научить студентов не только основам дискретной математики, но и показать ее роль в современных компьютерных технологиях, вооружить читателя методами, применяемыми для решения широкого круга задач. Пособие написано в доступной форме, достаточно полно и строго. Особое внимание в книге уделяется вопросу практической компьютерной реализации. В каждом конкретном случае содержится достаточно информации для применения рассматриваемых методов на практике. В этом отношении много полезного найдет читатель, более склонный к программистской работе и интересующийся предлагаемыми алгоритмами в силу их практического использования. Рассматриваемый в книге набор приемов и правил алгоритмического характера может составить хорошую основу формирования культуры разработки, анализа и программной реализации алгоритмов любого заинтересованного читателя.

Практическая компьютерная реализация большого количества рассматриваемых задач потребовала включения в книгу специальных разделов дискретной математики: представление абстрактных объектов, сортировка и поиск, порождение комби-

наторных объектов. Однако увеличение объема материала книги только лишь усугубило многие вопросы, позволило уточнить их суть, а так же придать общее звучание разделам курса, которые на первый взгляд никоим образом не связаны. Книгу можно рассматривать в качестве хорошего справочника методов и алгоритмов дискретной математики, широко используемых в разработках прикладного программного обеспечения.

Уровень математической подготовки, требующийся для понимания материала книги, может меняться от главы к главе. Минимум необходимых познаний в программировании соответствует уровню первокурсника, уже научившегося писать довольно пространные программы. Необходимый уровень математического образования соответствует типичной подготовке студента, прослушавшего ряд основных математических курсов, таких как математический анализ, аналитическая геометрия, линейная алгебра.

Учебник ориентирован на двухсеместровый лекционный курс, читаемый автором на механико-математическом факультете Дальневосточного государственного технического университета. Без сомнения, книга может составить хорошую основу курсов, примыкающих к дискретной математике, таких как информатика, алгоритмизация процессов, анализ вычислительных алгоритмов.

Учебник предназначен прежде всего для студентов специальности «Прикладная математика», а также других специальностей, изучающих дискретную математику и программирование. Книга будет полезна преподавателям, аспирантам и научным работникам, применяющим методы дискретной математики в прикладных задачах.

Глава 1

# ВВЕДЕНИЕ В МАТЕМАТИЧЕСКУЮ ЛОГИКУ

Математическая логика (называемая также *символической логикой*, *формальной логикой*) — это логика, развиваемая с помощью математических методов. Изучать математическую логику — значит, изучать логику, используемую в математике. Математическая логика как наука насыщена весьма богатым и разнообразным материалом. Мы начнем с некоторых ее вспомогательных важных разделов, чтобы затем иметь возможность продвигаться вширь и вглубь.

## § 1.1. Системы счисления

В данном разделе рассматриваются системы счисления как средства представления целых чисел. Изучение свойств систем счисления имеет важное и практическое значение. Так, двоичная система счисления лежит в основе всех современных вычислительных устройств.

### 1.1.1. Системы счисления

Определение. *Системой счисления* называется совокупность приемов и правил, используемых для представления чисел с помощью цифровых знаков: $\pm, 0, 1, 2, \ldots$ .

Для представления чисел используют позиционные системы счисления и непозиционные. Например, к непозиционным относится «римская система счисления» и другие подобного рода. Каждая непозиционная система счисления предлагает свои уникальные правила для записи чисел. Далее будем обращаться лишь к позиционным системам счисления.

Определение. *Позиционная система счисления* — это такая система счисления, в которой числовое значение каждой цифры зависит от ее места (позиции или разряда) в записи числа. При переходе же в следующий соседний разряд числовое содержание цифры меняется в одно и то же число раз.

Определение. *Основанием* позиционной системы счисления называется количество цифр, используемых в данной системе для записи чисел.

В следующей таблице рассматриваются примеры систем счисления и принятые (используемые) ими для записи чисел цифровые знаки.

| Основание системы счисления | Используемые цифровые знаки |
|---|---|
| Десятичная | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 |
| Восьмеричная | 0, 1, 2, 3, 4, 5, 6, 7 |
| Шестнадцатеричная | 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F |
| Двоичная | 0, 1 |
| $q$-я система счисления | $0, 1, \ldots, q-1$ |

В системе счисления с основанием $q$ каждое положительное целое число $N$ имеет единственное представление в виде конечной последовательности

$$N = (a_{k-1},\ a_{k-2},\ \ldots,\ a_2,\ a_1,\ a_0)_q, \tag{1.1}$$

где каждое $a_i$ — целое, удовлетворяющее условию $0 \leqslant a_i < q$, и $a_{k-1} \neq 0$. Число, в записи которого все $a_i = 0$, $i = 0, 1, \ldots, k-1$, является нулем. Нуль представляется последовательностью $(0)_q$. Целое, соответствующее последовательности (1.1), имеет вид

$$N = a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_2q^2 + a_1q^1 + a_0q^0. \tag{1.2}$$

Форма представления целых чисел (1.2) называется *показательной* или *экспоненциальной*.

Утверждение 1.1.1. *Представление* (1.1) *целых положительных чисел $N$ является единственным.*

Доказательство — индукция по $N$. Проверим единственность такого представления для $N = 0$. Одно представление нуля — это $0 = (0)_q$. Допустим существование другого представления $0 = (a_{k-1}, a_{k-2}, \ldots, a_0)_q = a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_0q^0$, где $0 \leqslant a_i < q$. Условие $a_iq^i \geqslant 0$ представления (1.2), где $a_iq^i = 0$ лишь при $a_i = 0$, необходимо влечет $a_i = 0$ для всех $i = 0, 1, \ldots, k-1$ в записи числа $0 = a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_0q^0$. Единственность представления $N = 0$ доказана.

Пусть единственность представления выполняется для всех целых $0, 1, \ldots, N-1$. Покажем, что это верно и для числа $N$. Допустим, что это не так и существуют два различных представления:

$$\begin{aligned} N &= (a_{k-1},\, a_{k-2},\, \ldots,\, a_2,\, a_1,\, a_0)_q = \\ &= (b_{m-1},\, b_{m-2},\, \ldots,\, b_2,\, b_1,\, b_0)_q. \end{aligned} \tag{1.3}$$

Пусть $m > k$, где $b_{m-1} > 0$, тогда

$$\begin{aligned} N &= b_{m-1}q^{m-1} + b_{m-2}q^{m-2} + \ldots + b_0q^0 > \\ &> (q-1)q^{m-2} + \ldots + (q-1)q^0 \geqslant \\ &\geqslant a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_0q^0 = N. \end{aligned}$$

Получили противоречие $N > N$. Следовательно, возможно лишь $m = k$, $b_{k-1} > 0$ и $a_{k-1} > 0$. Рассмотрим для этого случая разность двух представлений (1.3):

$$0 = N - N = (b_{k-1} - a_{k-1})q^{k-1} + \sum_{i=0}^{k-2}(b_i - a_i)q^i.$$

Если $b_{k-1} - a_{k-1} > 0$, то

$$\begin{aligned} 0 &= (b_{k-1} - a_{k-1})q^{k-1} + \sum_{i=0}^{k-2}(b_i - a_i)q^i \geqslant \\ &\geqslant 1q^{k-1} + \sum_{i=0}^{k-2}(b_i - a_i)q^i \geqslant 1q^{k-1} - \sum_{i=0}^{k-2}(q-1)q^i = \\ &= 1q^{k-1} - (1q^{k-1} - 1) = 1. \end{aligned}$$

Таким образом, предположение $b_{k-1} - a_{k-1} > 0$ приводит к противоречию: $0 \geqslant 1$. Значит, $b_{k-1} = a_{k-1}$, где $b_{k-1} > 0$. Отсюда следует, что число $N - b_{k-1}q^{k-1}$ имеет два представления: $N - b_{k-1}q^{k-1} = (b_{k-2}, b_{k-3}, \ldots, b_0) = (a_{k-2}, a_{k-3}, \ldots, a_0)$. С другой стороны, $N - b_{k-1}q^{k-1} < N$ и по предположению индукции оно имеет единственное представление. Следовательно, $b_i = a_i$, $i = 0, 1, \ldots, k-2$, а так как $b_{k-1} = a_{k-1}$, то и $N$ имеет единственное представление. □

Доказательством существования представления (1.1) для целого $N \geqslant 0$ является алгоритм 1.1 перевода этого числа в $q$-ю

систему счисления $(a_{k-1}, a_{k-2}, \ldots, a_0)_q$. Алгоритм формирует последовательность $a_i$, $i = 0, 1, \ldots, k-1$, способом повторного деления $N$ на основание $q$ и записи остатков от деления в качестве $a_i$, $i = 0, 1, \ldots, k-1$. На первом шаге при делении числа $N$ на $q$ остаток будет $a_0$. На каждом следующем шаге в качестве делимого выступает целое частное предыдущего шага от деления на $q$. Повторное деление заканчивается, как только очередной остаток становится равным нулю. Значение $k$ показывает количество разрядов в записи числа $N$.

Алгоритм 1.1. $N$ $q$
$(a_{k-1}, a_{k-2}, \quad, a_0)_q$

$a_0 = 0$;
$s = N$;
$k = 0$;
**while** $s \neq 0$ **do begin**
  $a_k = s \ mod \ q$; { $q$}
  $s = \left\lfloor \frac{s}{q} \right\rfloor$; { $q$}
  $k = k + 1$;
**end;**
**if** $k = 0$ **then** $k = k + 1$.

Замечание. Очевидно, что наиболее широко используется десятичная система счисления. Двоичная же система лежит в основе всех современных вычислительных устройств. Сложность визуального восприятия содержания чисел в двоичной системе счисления с необходимостью требует средств записи их эквивалента в более компактной и содержательной форме. Именно шестнадцатеричная (в большей степени) и восьмиричная системы счисления используются для представления чисел в двоичной системе. Например, сегодня сложно найти язык программирования высокого уровня, где бы это было не реализовано. Более того, на этом основании в языках программирования двоичная система счисления даже может и не вводиться.

### 1.1.2. Смешанные системы счисления

Смешанные системы счисления являются обобщением рассмотренных в п. 1.1.1 систем счисления с основанием $q$. В смешанных системах определяется не единственное основание $q$, а целая последовательность таких оснований $q_0, q_1, q_2, \ldots$ . Теперь

представление числа $N$ в виде конечной последовательности $N = (a_{k-1}, a_{k-2}, \ldots, a_0)$ соответствует целому

$$N = a_{k-1} \prod_{i=0}^{k-2} q_i + \ldots + a_2 q_1 q_0 + a_1 q_0 + a_0, \qquad (1.4)$$

где $0 \leqslant a_i < q_i$, $i = 0, 1, \ldots, k-1$, и $a_{k-1} > 0$, если $N > 0$.

Обычная система счисления с основанием $q$ — это частный случай смешанной системы, где все $q_i = q$, $i \geqslant 0$. Единственность представления (1.4) для смешанных систем доказывается подобно тому, как это рассмотрено для обычных систем счисления. Смешанные системы счисления в практической деятельности встречаются так же часто, как и десятичная система.

Пример. Рассмотрим систему счисления измерения времени — секунды, минуты, часы, дни и недели. Это смешанная система с основаниями $q_0 = 60$, $q_1 = 60$, $q_2 = 24$ и $q_3 = 7$. В данной системе число $N = (5, 4, 3, 2, 1)$ интерпретируется как величина, равная 5 неделям, 4 дням, 3 часам, 2 минутам и 1 секунде. Значение целого $N = (5, 4, 3, 2, 1)$, соответствующего представлению (1.4), равно

$$\begin{aligned} N &= 5q_3q_2q_1q_0 + 4q_2q_1q_0 + 3q_1q_0 + 2q_0 + 1 = \\ &= 5 \cdot 7 \cdot 24 \cdot 60 \cdot 60 + 4 \cdot 24 \cdot 60 \cdot 60 + 3 \cdot 60 \cdot 60 + 2 \cdot 60 + 1 = \\ &= 3380521. \end{aligned}$$

В принятой интерпретации величина $N$ равна интервалу времени, выраженному в секундах.

Перевод целого $N \geqslant 0$ в смешанную систему счисления $N = (a_{k-1}, a_k, \ldots, a_0)$ с основаниями $q_0, q_1, q_2, \ldots$ можно выполнить посредством алгоритма 1.2. Нетрудно заметить, что положив в этом алгоритме все основания равными $q_i = q$, мы повторим алгоритм 1.1 перевода чисел для обычной системы счисления. Значение $k$ устанавливается равным количеству разрядов в записи числа $N$.

Алгоритм 1.2. $N$
$(a_{k-1}, a_{k-2}, \quad, a_0)$ $(q_{k-1}, q_{k-2}, \quad, q_0)$

$a_0 = 0$;
$s = N$;
$k = 0$;
**while** $s \neq 0$ **do begin**
$a_k = s \ mod \ q_k$; { $q_k$}

$s = \left\lfloor \dfrac{s}{q_k} \right\rfloor$; { $q_k$}
$k = k + 1$;
**end;**
**if** $k = 0$ **then** $k = k + 1$.

### 1.1.3. Задача оптимального кодирования

З а д а ч а оптимального кодирования формулируется следующим образом: *найти систему счисления, в которой наиболее выгодно* (*оптимально*) *хранить информацию.*

Р е ш е н и е. Найти оптимальную систему счисления — значит, определить ее основание $q_{\text{opt}}$. Уточним понятие «оптимальная» или «наиболее выгодная» система счисления. Без ограничения общности можно считать, что информация представляется последовательностью цифровых знаков $a_i$ в какой-либо системе с основанием $q$, где $0 \leqslant a_i < q$. Тогда информацию (данные) можно рассматривать как некоторое целое число

$$N = (a_{k-1},\, a_{k-2},\, \ldots,\, a_0)_q. \tag{1.5}$$

Так как значения $0 \leqslant a_i < q$, то для хранения цифровых знаков $a_i$, $i = 0, 1, \ldots, k-1$, необходимо иметь $q$ состояний в каждом числовом разряде устройства хранения данных. Оценим общее число состояний в запоминающем устройстве, необходимых для хранения числа $N$. Значение числа (1.5) запишем в показательной форме:

$$N = a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_0q^0. \tag{1.6}$$

Последнее выражение (1.6) позволяет определить количество разрядов $k$, необходимых для хранения числа $N$. Действительно,

$$\begin{aligned} N &= a_{k-1}q^{k-1} + a_{k-2}q^{k-2} + \ldots + a_0q^0 \leqslant \\ &\leqslant (q-1)q^{k-1} + (q-1)q^{k-2} + \ldots + (q-1)q^0 = \\ &= (q-1)(q^{k-1} + q^{k-2} + \ldots + q^0) = q^k - 1. \end{aligned}$$

Итак, $N \leqslant q^k - 1$. Отсюда $k = \lceil \log_q(N+1) \rceil \geqslant \log_q(N+1)$. Следовательно, для хранения $N$ потребуется число состояний $S = q \cdot k = q \cdot \lceil \log_q(N+1) \rceil$.

Именно суммарное число состояний

$$S(q) = q \cdot \lceil \log_q(N+1) \rceil \tag{1.7}$$

и является показателем оптимальности системы счисления. Основание оптимальной системы $q_{opt}$ должно доставлять минимум значению выражения (1.7).

Можно возразить, что элементная база технических средств определяет и систему счисления хранения данных. В целом так и есть, и все-таки, в какой системе оптимум лучше и насколько?

На рис. 1.1 представлен график функции числа состояний $S(x) = x \times \times \log_x(N+1)$ от переменного основания $x$ системы счисления. Минимальное значение достигается в точке $x = e \approx 2.72$, для которой $S'(x) = 0$. Отсюда заключаем, что наиболее предпочтительными (оптимальными) являются две системы счисления — двоичная и троичная.

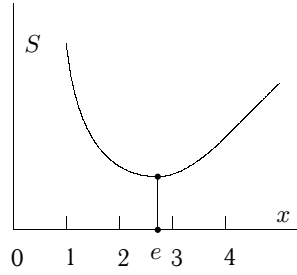


Рис. 1.1

### 1.1.4. Системы счисления с основаниями 2, 8 и 16

Системы счисления с основаниями 2, 8 и 16 занимают особое место в различного рода компьютерных технологиях как аппаратных, так и программных. В качестве пояснения приведем следующий пример. Память вычислительных устройств подразделяется на байты. Байт — это набор из 8 бит. Каждый бит имеет 2 состояния — 0 и 1. Таким образом, информация в памяти представляется как последовательность 8-разрядных чисел (байт) в 2-ичной системе счисления. Сложность восприятия двоичных наборов сводит на нет их широкое использование в практических приложениях. Однако 8-битовый размер байта позволяет очень удобно представить содержимое его в 16-ричной системе счисления. Действительно, разделив байт пополам по 4 бита, получим представление байта в виде двузначного числа в 16-ричной системе счисления. Информация в таком виде легко читается и не носит объемного характера. Именно это обстоятельство делает незаменимой 16-ричную систему счисления, практически, во всех вычислительных устройствах.

#### Связь систем счисления с основаниями 2 и 8

Рассмотрим представление целого $N \geqslant 0$ в 8-ричной и 2-ичной системах:

$$\begin{aligned} N_8 &= (b_{m-1}, b_{m-2}, \ldots, b_0)_8 = \\ &= b_{m-1}8^{m-1} + b_{m-2}8^{m-2} + \ldots + b_0 8^0, \end{aligned} \tag{1.8}$$

$$N_2 = (a_{k-1}, a_{k-2}, \ldots, a_0)_2 = \\ = a_{k-1}2^{k-1} + a_{k-2}2^{k-2} + \ldots + a_0 2^0, \tag{1.9}$$

где $0 \leqslant a_i < 2$, $0 \leqslant b_j < 8$ и $a_{k-1} > 0$, $b_{m-1} > 0$, если $N \neq 0$. Значения $k, m \geqslant 1$ определяют количество разрядов в записи соответствующего представления числа $N$.

Запишем представление (1.9) для $N_2$, объединяя справа налево (от младших разрядов к старшим) каждые последующие три слагаемых в одно слагаемое. Недостающие старшие разряды дополняем нулевыми значениями. Получим

$$N_2 = \left(a_{3(m-1)+2}2^2 + a_{3(m-1)+1}2^1 + a_{3(m-1)}2^0\right) 2^{3(m-1)} + \ldots + \\ + \left(a_5 2^2 + a_4 2^1 + a_3 2^0\right) 2^3 + \left(a_2 2^2 + a_1 2^1 + a_0 2^0\right) 2^0 = \\ = \left(a_{3(m-1)+2}, a_{3(m-1)+1}, a_{3(m-1)}\right)_2 8^{m-1} + \ldots + \\ + (a_5, a_4, a_3)_2 8^1 + (a_2, a_1, a_0)_2 8^0. \tag{1.10}$$

Единственность представления числа $N$ в какой-либо системе счисления (утверждение 1.1.1) дает основание заключить, что разложения (1.8) и (1.10) совпадают. Следовательно, выполняются равенства разрядов указанных разложений:

$$\begin{aligned} b_0 &= (a_2, a_1, a_0)_2 = a_2 2^2 + a_1 2^1 + a_0 2^0, \\ b_1 &= (a_5, a_4, a_3)_2 = a_5 2^2 + a_4 2^1 + a_3 2^0, \\ b_2 &= (a_8, a_7, a_6)_2 = a_8 2^2 + a_7 2^1 + a_6 2^0, \\ &\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ b_{m-1} &= (a_{3m-1}, a_{3m-2}, a_{3m-3})_2 = \\ &= a_{3m-1}2^2 + a_{3m-2}2^1 + a_{3m-3}2^0. \end{aligned} \tag{1.11}$$

Отсюда следует и правило перевода целых $N \geqslant 0$.

Правило. Сравнивая теперь представления $N$ в 8-ричной (1.8) и в 2-ичной (1.9) системах счисления, заключаем, что каждая цифра разряда $(b_i)_8$, $i \geqslant 0$, 8-ричной системы счисления определяет соответствующие цифры трех разрядов $(a_{3i+2}, a_{3i+1}, a_{3i})_2$ 2-ичной системы счисления; верно и обратно.

Для заинтересованных читателей приведем таблицу (табл. 1.1) соответствия значений $(b_i)_8$ и $(a_{3i+2}, a_{3i+1}, a_{3i})_2$.

Таблица 1.1

| $(b_i)_8$ | $(a_{3i+2}, a_{3i+1}, a_{3i})_2$ |
|---|---|
| 0 | $000 = 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| 1 | $001 = 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| 2 | $010 = 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| 3 | $011 = 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |
| 4 | $100 = 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| 5 | $101 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| 6 | $110 = 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| 7 | $111 = 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |

### Связь систем счисления с основаниями 2 и 16

Нетрудно заметить, что столь удобный переход от 2-ичной системы к 8-ричной и обратно, в обход общей процедуре перевода чисел, стал возможен из-за соотношения между основаниями указанных систем: $8 = 2^3$. Основания же 2-ичной и 16-ричной систем связаны соотношением $16 = 2^4$, что позволяет сформулировать следующее правило перевода чисел для последних систем счисления.

Правило. Пусть целое $N \geqslant 0$ представлено в 16-ричной и 2-ичной системах счисления:

$$N_{16} = (b_{m-1}, b_{m-2}, \ldots, b_0)_{16}, \tag{1.12}$$

$$N_2 = (a_{k-1}, a_{k-2}, \ldots, a_0)_2, \tag{1.13}$$

где $0 \leqslant a_i < 2$, $0 \leqslant b_j < 16$ и $a_{k-1} > 0$, $b_{m-1} > 0$, если $N \neq \neq 0$. Значения $k, m \geqslant 1$ определяют количество разрядов в записи соответствующего представления числа $N$.

В (1.12) каждая цифра разряда $(b_i)_{16}$, $i \geqslant 0$, 16-ричной системы счисления определяет в (1.13) соответствующие цифры четырех разрядов $(a_{4i+3}, a_{4i+2}, a_{4i+1}, a_{4i})_2$ 2-ичной системы счисления; верно и обратно.

Приведем таблицу (табл. 1.2) соответствия значений $(b_i)_{16}$ и $(a_{4i+3}, a_{4i+2}, a_{4i+1}, a_{4i})_2$.

Пример. Пусть $N = 3272_{10}$. Найти $x_2 = y_8 = z_{16} = N$.

Решение. Удобнее выполнить перевод по следующей схеме: $N_{10} \to y_8 \to x_2 \to z_{16}$. Так и поступим.

Таблица 1.2

| $(b_i)_{16}$ | $(a_{4i+3}, a_{4i+2}, a_{4i+1}, a_{4i})_2$ |
|---|---|
| 0 | $0000 = 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| 1 | $0001 = 0\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| 2 | $0010 = 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| 3 | $0011 = 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |
| 4 | $0100 = 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| 5 | $0101 = 0\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| 6 | $0110 = 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| 7 | $0111 = 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |
| 8 | $1000 = 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| 9 | $1001 = 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| $A$ | $1010 = 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| $B$ | $1011 = 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |
| $C$ | $1100 = 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 0\cdot 2^0$ |
| $D$ | $1101 = 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0$ |
| $E$ | $1110 = 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0$ |
| $F$ | $1111 = 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0$ |

Согласно алгоритму 1.1 перевода из 10-ичной системы в любую другую получим $3272_{10} = 6310_8 = y_8$.

Далее воспользуемся правилом перевода из 8-ричной в 2-ичную. Имеем $6_8 = 110_2$, $3_8 = 011_2$, $1_8 = 001_2$, $0_8 = 000_2$. Отсюда $x_2 = \underbrace{110}\,\underbrace{011}\,\underbrace{001}\,\underbrace{000}_2$.

По правилу перевода из 2-ичной системы в 16-ричную запишем $y_{16} = \underbrace{1100}\,\underbrace{1100}\,\underbrace{1000}_2 = CC8_{16}$.

## § 1.2. Введение в булеву алгебру

Под алгеброй в современной математике понимают *множество объектов* произвольной природы с определенными на них *операциями* и свойствами этих операций, даваемых в форме *аксиом*. Запишем основные требования, предъявляемые к алгебре. Операции алгебры должны быть применимы ко всем объектам множества. В результате выполнения операций должны получаться объекты той же природы, что и исходные. В этом случае говорят, что множество объектов замкнуто относительно опера-

ций. Операций в алгебре должно быть конечное число и каждая операция должна быть конечноместной и т. д.

Прежде формального введения в булеву алгебру удобно познакомиться с неформальным изложением данного вопроса. Это означает, что необходимо рассмотреть различного рода понятия, определения и, конечно же, объекты алгебры, возможные операций и их свойства.

### 1.2.1. Булевы функции

Определение. Переменная $x$ называется *булевой*, если она способна принимать только два значения 0 и 1. В качестве примера интерпретации такого рода переменных может выступать обычный настенный выключатель света на два положения. Здесь 1 соответствует положению переключателя вверх и 0 — положению вниз.

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется *булевой* (или логической, или функцией алгебры логики, или переключательной), если все ее аргументы $x_i$ являются булевыми, а сама функция также может принимать только два значения 0 и 1. Множество всех булевых функций от переменных $x_1, x_2, \ldots, x_n$ обозначают через $P_2$.

#### Способы задания булевых функций

Способы задания булевых функций не отличаются от способов задания обычных функций анализа. К таковым способам задания стандартно относятся:

1) табличный;
2) графический;
3) аналитический.

#### (1) Табличный способ задания

Пусть $w = f(x_1, x_2, \ldots, x_n)$ — булева функция $n$ аргументов. Область определения данной функции можно рассматривать и как множество упорядоченных наборов (или векторов, или двоичных наборов) $D = \{(x_1, x_2, \ldots, x_n) \mid x_i \in \{0, 1\},\ i = 1, 2, \ldots, n\}$, на каждом из которых функция принимает одно из двух значений: $w \in \{0, 1\}$. Количество таких наборов $(x_1, x_2, \ldots, x_n)$, согласно правилу прямого произведения (см. п. 2.2), равно

$$|D| = |\underbrace{\{0,1\} \times \{0,1\} \times \ldots \times \{0,1\}}_{n}| = 2 \cdot 2 \cdot \ldots \cdot 2 = 2^n.$$

Нетрудно определить и количество всех функций $w = f(x_1, x_2, \ldots, x_n)$. Отдельная функция $w = f(x_1, x_2, \ldots, x_n)$ задана, если определены ее значения $(w_1, w_2, \ldots, w_{2^n})$ на всех наборах $(x_1, x_2, \ldots, x_n) \in D$, где $w_j \in \{0, 1\}$ — значение функции $w = f(x_1, x_2, \ldots, x_n)$ на $j$-м наборе $(x_1, x_2, \ldots, x_n) = (01 \ldots 1) \in D$, $j = 1, 2, \ldots, 2^n$. Итак, количество булевых функций $w = f(x_1, x_2, \ldots, x_n)$ совпадает с числом двоичных наборов $(w_1, w_2, \ldots, w_{2^n})$, где $w_j \in \{0, 1\}$. Согласно правилу прямого произведения (см. п. 2.2) число последних равно

$$|\underbrace{\{0,1\} \times \{0,1\} \times \ldots \times \{0,1\}}_{2^n}| = 2 \cdot 2 \cdot \ldots \cdot 2 = 2^{2^n}.$$

Таблица 1.3

| Число | Двоичная форма | | | Функция |
|---|---|---|---|---|
| № | $x$ | $y$ | $z$ | $w = f(x, y, z)$ |
| 0 | 0 | 0 | 0 | $f_0 = 1 = f(0, 0, 0)$ |
| 1 | 0 | 0 | 1 | $f_1 = 1 = f(0, 0, 1)$ |
| 2 | 0 | 1 | 0 | $f_2 = 0 = f(0, 1, 0)$ |
| 3 | 0 | 1 | 1 | $f_3 = 1 = f(0, 1, 1)$ |
| 4 | 1 | 0 | 0 | $f_4 = 0 = f(1, 0, 0)$ |
| 5 | 1 | 0 | 1 | $f_5 = 0 = f(1, 0, 1)$ |
| 6 | 1 | 1 | 0 | $f_6 = 0 = f(1, 1, 0)$ |
| 7 | 1 | 1 | 1 | $f_7 = 1 = f(1, 1, 1)$ |

В качестве примера рассмотрим табличное представление булевой функции трех аргументов $w = f(x, y, z)$, где $w, x, y, z \in \{0, 1\}$. Область определения функции — это множество двоичных наборов $D = \{(x, y, z) \mid x, y, z \in \{0, 1\}\}$. Число наборов равно $|D| = 2^3 = 8$, а количество таких функций равно $2^{|D|} = 2^{2^3} = 256$. Значения функции $f(x, y, z)$ удобно представить в виде табл. 1.3, где перечислены всевозможные наборы из нулей и единиц длины 3 и для каждого набора указано значение функции $f_i \in \{0, 1\}$ на этом наборе.

В таблицах, аналогичных табл. 1.3, обычно употребляется расположение наборов, соответствующих порядку естественного роста двоичных чисел: $0, 1, \ldots, 2^n - 1$, в примере положено $n = 3$.

Определение. Таблицы значений булевых функций, подобные табл. 1.3, называются *таблицами истинности* булевых функций. Название таблиц происходит от интерпретации значений 1 — истина (TRUE), 0 — ложь (FALSE).

### (2) Графический способ задания

Рассмотрим графическое представление булевой функции трех аргументов $w = f(x, y, z)$, заданной таблично (табл. 1.3). Заметим, что множество наборов области определения функции $D = \{(x, y, z) \mid x, y, z \in \{0, 1\}\}$ является множеством координат точек вершин единичного трехмерного куба (рис. 1.2). Очевидный способ графического представления булевой фунции — это отметить каким-то образом вершины куба, в которых функция принимает значение 1. Именно так на рис. 1.2 и сделано. В соответствии с таблицей значений (табл. 1.3) отмечены вершины, в которых булева функция равна 1.

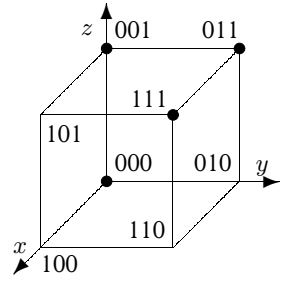



Рис. 1.2

Замечание. Очевидно, что область определения булевой функции $n$ аргументов $w = f(x_1, x_2, \ldots, x_n)$ составляется из наборов координат точек вершин единичного $n$-мерного куба.

### (3) Аналитический способ задания

Приведем таблицы истинности, обозначения и названия булевых функций одного и двух аргументов. В табл. 1.4 представлены все (их $2^{2^1} = 4$) функции одного аргумента, в табл. 1.5 — все функции двух аргументов (их $2^{2^2} = 16$).

Таблица 1.4

| $x$ | 0 1 | Обозначение | Название |
|---|---|---|---|
| 0 | 0 0 | 0 | Нуль, const 0 |
| 1 | 0 1 | $x$ | Повторение $x$ |
| 2 | 1 0 | $\neg x$, $\overline{x}$ | Отрицание $x$, не $x$ |
| 3 | 1 1 | 1 | Единица, const 1 |

Функции 0 и 1 называются соответственно *тождественным нулем* и *тождественной единицей*. Иногда эти функции 0 и 1 рассматривают как функции, зависящие от пустого множества переменных.

Таблица 1.5

| $x$ $y$ | 0011 0101 | Обозначение | Название |
|---|---|---|---|
| 0 | 0000 | $0$ | Нуль, const 0 |
| 1 | 0001 | $x \cdot y$, $x \wedge y$, $x\&y$ | Конъюнкция |
| 2 | 0010 | $y \nrightarrow x$, $x \cdot \overline{y}$ | Запрет по $y$ |
| 3 | 0011 | $x$ | Повторение $x$ |
| 4 | 0100 | $x \nrightarrow y$, $\overline{x} \cdot y$ | Запрет по $x$ |
| 5 | 0101 | $y$ | Повторение $y$ |
| 6 | 0110 | $x \oplus y$ | Сумма по модулю 2 |
| 7 | 0111 | $x \vee y$ | Дизъюнкция |
| 8 | 1000 | $x \downarrow y$ | Стрелка Пирса |
| 9 | 1001 | $x \sim y$ | Эквивалентность |
| 10 | 1010 | $\neg y$, $\overline{y}$ | Отрицание $y$ |
| 11 | 1011 | $y \to x$, $y \Rightarrow x$, $y \supset x$ | Импликация от $y$ к $x$ |
| 12 | 1100 | $\neg x$, $\overline{x}$ | Отрицание $x$ |
| 13 | 1101 | $x \to y$, $x \Rightarrow y$, $x \supset y$ | Импликация от $x$ к $y$ |
| 14 | 1110 | $x \mid y$ | Штрих Шеффера |
| 15 | 1111 | $1$ | Единица, const 1 |

Функции одного и двух аргументов, представленные в таблицах 1.4 и 1.5, называются *элементарными*.

Символы $\neg$, $|$, $\downarrow$, $\wedge$, $\nrightarrow$, $\vee$, $\to$, $\oplus$, $\sim$, участвующие в обозначениях элементарных функций, называются *логическими связками* (*операциями*) или *функциональными символами*.

### 1.2.2. Формулы, реализация функций формулами

Построение *формул* выполняется из *элементарных булевых функций* и носит рекурсивный характер. Пусть $X$ — некоторый фиксированный *алфавит переменных*, $\sigma = \{\neg, |, \downarrow, \wedge, \nrightarrow, \vee, \to, \oplus, \sim\}$ — *множество функциональных символов* (*базис*) и $F$ — множество булевых функций, соответствующих функциональным символам $\sigma$.

Определение. *Формулой над* $\sigma$ называется всякое (и только такое) выражение вида:

1) $x$ — любая переменная из множества $X$;
2) $(\neg A)$, $(A|B)$, $(A \downarrow B)$, $(A \wedge B)$, $(A \nrightarrow B)$, $(A \vee B)$, $(A \to B)$, $(A \oplus B)$, $(A \sim B)$, где $A, B$ — это формулы над $\sigma$.

Примеры формул. $G_1 = ((y \to x) \oplus ((\neg y) \cdot x))$, $G_2 = (\neg(x \nrightarrow (((\neg x)|((\neg x)|(\neg y))) \downarrow y) \vee y))$, $G_3 = ((x \vee y) \sim z)$.

Реализация функций формулами. Всякой формуле $G$ однозначно соответствует некоторая функция $f_G$. Понятие булевой функции $f_G$, реализуемой формулой $G$, вводится рекурсивно следующим образом:

1) формуле $G = x$, где $x \in X$, сопоставляется функция $f_G(x) = x$;

2) если $G$ равна одной из формул $(\neg A)$, $(A|B)$, $(A \downarrow B)$, $(A \wedge B)$, $(A \nrightarrow B)$, $(A \vee B)$, $(A \to B)$, $(A \oplus B)$, $(A \sim B)$, где $A, B$ — это формулы над $\sigma$, то $f_G$ равно соответствующей элементарной булевой функции $\neg f_A$, $f_A|f_B$, $f_A \downarrow f_B$, $f_A \wedge f_B$, $f_A \nrightarrow f_B$, $f_A \vee f_B$, $f_A \to f_B$, $f_A \oplus f_B$, $f_A \sim f_B$. Если $f_G(x_1, x_2, \ldots, x_n)$, $x_i \in X$, то значение ее на произвольном наборе $(\alpha_1, \alpha_2, \ldots, \alpha_n)$, $\alpha_i \in \{0, 1\}$, совпадает со значением на этом наборе для соответствующей ей элементарной булевой функции.

Таким образом, зная таблицы истинности элементарных функций (*функций базиса*) (табл. 1.5), можно вычислить и таблицу истинности функции $f_G$, которую реализует формула $G$.

Эквивалентные формулы. Формулы $G_1$ и $G_2$ над $\sigma$ называются *эквивалентными*, если они реализуют равные булевы функции $f_{G_1}$ и $f_{G_2}$.

Приоритет операций. Для булевых формул допустима и бесскобочная их запись. Например, $G = x \oplus y \sim z \to z \to y \vee z$. Для составления таблицы истинности таких формул необходимо установить последовательность выполнения операций — расставить скобки. С этой целью каждой логической операции $\neg, |, \downarrow, \wedge, \nrightarrow, \vee, \to, \oplus, \sim$ назначается свой отличный от других приоритет операций. Приоритет операций устанавливается следующим их списком:

$$\neg,\ |,\ \downarrow,\ \wedge,\ \nrightarrow,\ \vee,\ \to,\ \oplus,\ \sim, \tag{1.14}$$

где чем левее расположение операции по списку, тем сильнее ее приоритет.

**Порядок выполнения операций в формулах**

1. Если формула написана со скобками, то порядок определяется скобками.
2. Если формула написана без скобок, то одинаковые операции выполняются в порядке их появления; порядок же различных операций, в этом случае, определяется в соответствии с их приоритетом (1.14): $\neg, |, \downarrow, \wedge, \nrightarrow, \vee, \to, \oplus, \sim$.

Определение. Формула называется *тождественно-истинной*, если ее значение на всех наборах переменных равно 1.

Формула называется *тождественно-ложной*, если ее значение на всех наборах переменных равно 0. В противном случае формула называется *условно-истинной*.

Пример. Составить таблицу истинности формулы

$$G = x \oplus y \sim z \to z \to y \vee z.$$

Решение. Расстановка скобок в формуле $G$ дает следующий порядок выполнения операций

$$G = \underbrace{\Big(\underbrace{(x \oplus y)}_{4} \sim \underbrace{\big(\underbrace{(z \to z)}_{2} \to \underbrace{(y \vee z)}_{1}\big)}_{3}\Big)}_{5}.$$

Значения истинности для формулы $G$ приведены в табл. 1.6.

Таблица 1.6

| | | (1) | (2) | (3) | (4) | (5) |
|---|---|---|---|---|---|---|
| № | $x\ y\ z$ | $y \vee z$ | $z \to z$ | (2) $\to$ (1) | $x \oplus y$ | (4) $\sim$ (3) |
| 0 | 0 0 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 0 1 | 1 | 1 | 1 | 0 | 0 |
| 2 | 0 1 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 1 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 1 0 0 | 0 | 1 | 0 | 0 | 1 |
| 5 | 1 0 1 | 1 | 1 | 1 | 0 | 0 |
| 6 | 1 1 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 1 1 | 1 | 1 | 1 | 1 | 1 |

### 1.2.3. Замена переменных и суперпозиция

В основу рекурсивного правила получения новых формул (п. 1.2.2) из элементарных булевых функций положены *операции замены переменных и суперпозиции*. Рассмотрим их определение и примеры использования.

Определение. *Операцией замены переменных* булевой функции $f(x_1, x_2, \ldots, x_n)$ называется переименование ее пере-

менных или замена их порядка. Пусть $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — перестановка (п. 2.5), тогда операция замены исходных переменных $(x_1, x_2, \ldots, x_n)$ на новые $(y_{\pi_1}, y_{\pi_2}, \ldots, y_{\pi_n})$ записывается как

$$g(y_1, y_2, \ldots, y_n) = f(x_1, x_2, \ldots, x_n) = f(y_{\pi_1}, y_{\pi_2}, \ldots, y_{\pi_n}),$$
$$\left| \begin{array}{l} y_{\pi_1} \to x_1 \\ y_{\pi_2} \to x_2 \\ \ldots \\ y_{\pi_n} \to x_n \end{array} \right.$$

где $g(y_1, y_2, \ldots, y_n)$ — новая булева функция — результат операции замены переменных. Ниже даны примеры таких операций:

$$(x \oplus y) \vee z = (z \oplus z) \vee x, \quad \left| \begin{array}{l} z \to x \\ z \to y \\ x \to z \end{array} \right. \qquad (x \sim y) \nrightarrow z = (y \sim z) \nrightarrow w, \quad \left| \begin{array}{l} y \to x \\ z \to y \\ w \to z \end{array} \right.$$

Определение. Операция *суперпозиции* (или *наложения*) булевых функций состоит в том, что вместо аргументов данной булевой функции $f(x_1, x_2, \ldots, x_n)$ подставляют некоторые другие булевы функции $h_1(y_{k_{11}}, y_{k_{12}}, \ldots, y_{k_{1m_1}})$, $h_2(y_{k_{21}}, y_{k_{22}}, \ldots, y_{k_{2m_2}})$, ... ..., $h_r(y_{k_{r1}}, y_{k_{r2}}, \ldots, y_{k_{rm_r}})$ и записывают

$$f(x_1, x_2, \ldots, x_n) = g(x_{j_1}, x_{j_2}, \ldots, x_{j_s}, y_{k_{11}}, y_{k_{12}}, \ldots, y_{k_{rm_r}}),$$
$$\left| \begin{array}{l} h_1(y_{k_{11}}, y_{k_{12}}, \ldots, y_{k_{1m_1}}) \to x_{i_1} \\ h_2(y_{k_{21}}, y_{k_{22}}, \ldots, y_{k_{2m_2}}) \to x_{i_1} \\ \ldots \\ h_r(y_{k_{r1}}, y_{k_{r2}}, \ldots, y_{k_{rm_r}}) \to x_{i_r} \end{array} \right.$$

В качестве примера приведем суперпозицию следующих функций:

$$x \oplus y = ((x \sim z) \oplus (z \to y)) \oplus (x | (y \nrightarrow x)).$$
$$\left| \begin{array}{l} (x \sim z) \oplus (z \to y) \to x \\ x \,|\, (y \nrightarrow x) \to y \end{array} \right.$$

Замечание. Нетрудно заметить, что операция замены переменных является частным случаем операции суперпозиции, если в последней в качестве функций $h_1, h_2, \ldots, h_r$ положить обычные булевы переменные.

### 1.2.4. Существенные и фиктивные переменные

Скажем, что булева функция $f(x_1, x_2, \ldots, x_n)$ *существенно* зависит от переменной $x_i$, если существует такой набор значений $(\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n)$ переменных $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$, что

$$f(\alpha_1, \ldots, \alpha_{i-1}, 0, \alpha_{i+1}, \ldots, \alpha_n) \neq f(\alpha_1, \ldots, \alpha_{i-1}, 1, \alpha_{i+1}, \ldots, \alpha_n).$$

Переменная $x_i$ в этом случае называется *существенной*. В противном случае $x_i$ называется *несущественной* или *фиктивной*.

Рассмотрим следующий пример таблиц истинности двух функций $f_1$ и $f_2$:

| № | $x$ | $y$ | $f_1$ | $f_2$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 2 | 1 | 0 | 0 | 1 |
| 3 | 1 | 1 | 0 | 0 |

Фиктивной является для функции $f_1$ переменная $y$, для функции $f_2$ — переменная $x$.

Определение. Булевы функции $f_1$ и $f_2$ называются *равными*, если функцию $f_1$ можно получить из $f_2$ способом введения и (или) удаления фиктивных переменных.

### 1.2.5. Интерпретация булевых функций

В контексте рассматриваемых вопросов *интерпретацию* следует понимать как соотношение двух систем или вложение смысла. В основе интерпретации как соотношения двух систем лежит *интерпретирующий словарь* — список соответствующих понятий, терминов, определений указанных систем. Предлагается рассмотреть релейно-контактную интерпретацию и смысловую.

#### Релейно-контактная интерпретация

Объектами релейно-контактной интерпретации являются *контакты* (*реле*) и *контактные схемы*. Поведение контакта рассмотрим на примере работы физического устройства — реле и обычной кнопки дверного звонка. Схема реле представлена на рис. 1.3. Опишем кратко работу реле.

Суть работы реле — это его способность находиться в одном из двух состояний. Одно состояние связывается с наличием напряжения на обмотке сердечника, а другое — с его отсутствием. Упругая пластина реле имеет свойство отклоняться вверх при отсутствии напряжения на обмотке сердечника. При наличии на обмотке напряжения сердечник намагничивается и притягивает упругую пластину, устанавливая ее в горизонтальное положение. Именно это свойство реле изменять положение упругой пластины используется в электрических цепях в качестве переключателя.


Рис. 1.3. Реле

Различают два способа включения реле в электрические цепи. При первом способе реле используется как нормально разомкнутый контакт (рис. 1.4), а при втором — как нормально замкнутый контакт (рис. 1.5).

### Нормально разомкнутый контакт (рис. 1.4)

Нормальное состояние *нормально разомкнутого контакта* (реле) — это отсутствие напряжения на *обмотке сердечника* реле (рис. 1.4). Включение такого контакта в электрическую цепь устанавливает ее в непроводящее состояние.

При наличии напряжения на *обмотке сердечника* реле упругая пластина притягивается сердечником вниз и цепь замыкается (рис. 1.4).

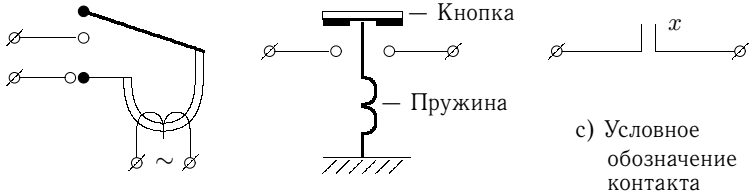



Рис. 1.4. Нормально разомкнутый контакт

### Нормально замкнутый контакт (рис. 1.5)

Нормальное состояние *нормально замкнутого контакта* (реле) — это отсутствие напряжения на *обмотке сердечника* реле (рис. 1.5). При включении такого контакта в электрическую

цепь его упругая пластина заводится за контакт цепи (рис. 1.5). Такое включение устанавливает электрическую цепь в проводящее состояние.

При наличии напряжения на *обмотке сердечника* реле упругая пластина притягивается сердечником вниз и цепь разрывается (рис. 1.5).

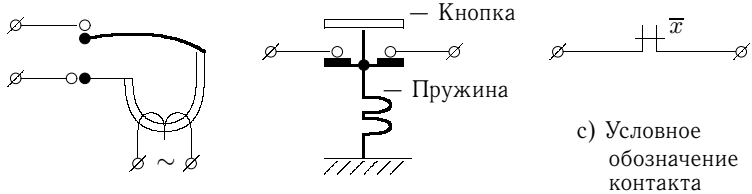



Рис. 1.5. Нормально замкнутый контакт

Вывод. Таким образом, объектами релейно-контактных схем являются *контакты и контактные схемы*. Различают два вида контактов — нормально разомкнутые и нормально замкнутые. Контактные схемы также имеют два состояния — проводящее и непроводящее.

## Интерпретирующий словарь соответствия объектов булевой алгебры и объектов релейно-контактных схем

| Объекты булевой алгебры | Объекты релейно-контактных схем |
|---|---|
| Булева переменная<br>Значение переменной $\left\{\begin{array}{c}0\\1\end{array}\right\}$ | Нормально разомкнутый контакт<br>$\left\{\begin{array}{c}\text{НЕТ}\\\text{ЕСТЬ}\end{array}\right\}$ — напряжение на обмотке |
| Булева функция<br>Значение функции $\left\{\begin{array}{c}0\\1\end{array}\right\}$ | Контактная цепь<br>$\left\{\begin{array}{c}\text{НЕТ}\\\text{ДА}\end{array}\right\}$ — проводящее состояние цепи |

## Примеры интерпретаций булевых функций

Предложенные примеры интерпретаций в дальнейшем будем использовать как строительные кирпичики для построении контактных схем любой сложности. Пример контактной схемы булевой функции $x \oplus y$ не должен вводить в заблуждение. За-

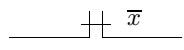
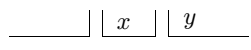
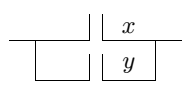
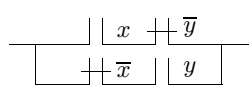
| Булевы функции | Контактные схемы |
|---|---|
| 1). $x$ |  |
| 2). $\overline{x}$ |  |
| 3). $x \cdot y$ |  |
| 4). $x \vee y$ |  |
| 5). $x \oplus y$ |  |

Рис. 1.6. Примеры интерпретаций

интересованный читатель далее обнаружит, что построить контактную схему произвольной булевой функции не сложнее, чем вычислить ее таблицу истинности.

### Смысловая интерпретация

Объектами смысловой интерпретации являются *высказывания*. Различают *простые и сложные высказывания*.

Определение. *Высказывание* — это предложение (или набор слов, или утверждение, или суждение), о котором имеет смысл говорить, что оно истинно либо ложно.

Определение. *Простым* называется высказывание, которое нельзя разбить на части так, чтобы каждая из них являлась высказыванием.

Определение. *Сложным* называется высказывание, состоящее из нескольких простых, связанных логическими связками.

Примеры высказываний. «Сахалин — остров». «Платон — грек». «2 меньше 3». «Святогор — богатырь». «Если $A \subset B$, а $B \subset C$, то $A \subset C$».

Замечание. Истинные высказывания еще называют *тождественно-истинными*, а ложные — *тождественно-ложными* высказываниями.

### Интерпретирующий словарь соответствия объектов булевой алгебры и смысловых объектов

| Объекты булевой алгебры | Смысловые объекты |
|---|---|
| Булева переменная<br>Значение переменной $\left\{\begin{array}{c}0\\1\end{array}\right\}$ | Высказывание<br>$\left\{\begin{array}{c}\text{ЛОЖЬ}\\\text{ИСТИНА}\end{array}\right\}$ — значение высказывания |
| Булева функция<br>Значение функции $\left\{\begin{array}{c}0\\1\end{array}\right\}$ | Сложное высказывание<br>$\left\{\begin{array}{c}\text{ЛОЖЬ}\\\text{ИСТИНА}\end{array}\right\}$ — значение высказывания |

### Примеры смысловой интерпретации булевых функций

Обозначим через $A$ и $B$ произвольные высказывания, которые интерпретируются как булевы переменные. Приведем базовый (не исчерпывающий) список высказываний, рассуждений (*справа*), которые могут быть заменены (интерпретированы) логическими формулами, стоящими *слева*.

| **Булевы функции** | **Смысловая интерпретация** |
|---|---|
| $\neg A$ — | Не $A$. |
| | $A$ не имеет смысла. |
| | $A$ не верно. |
| $A\&B$ — | $A$ и $B$. |
| | Как $A$, так и $B$. |
| | $A$ вместе с $B$. |
| | $A$, в то время как $B$. |
| | $B$, хотя и $A$. |
| | Не только $A$, но и $B$. |
| $A \vee B$ — | $A$ или $B$ или оба. |
| | $A$ или $B$. |
| | $A$, если не $B$ |
| $A \supset B$ — | Если $A$, то $B$. |
| | Для $A$ необходимо $B$. |
| | Для $B$ достаточно $A$. |
| | В случае $A$ имеет место $B$. |
| $A \sim B$ — | Если $A$, то $B$ и обратно. |
| | Для $A$ необходимо и достаточно $B$. |
| | $A$ тогда и только тогда, когда $B$. |
| | $A$ равносильно $B$. |
| | $A$ эквивалентно $B$. |
| $A \oplus B$ — | $A$ или $B$, но не оба. |
| | $A$, кроме случая, когда $B$. |
| | Либо $A$, либо $B$. |

Пример. Переведем следующие рассуждения в логическую символику (формулы). Проверьте правильность полученных ответов, вычислив для них таблицы истинности.

1. Любой друг Коли $[K]$ — друг Васи $[B]$. Петя не друг Васи $[\neg B]$, значит Петя не друг Коли $[\neg K]$. *Ответ*: $((K \supset B) \times \times \overline{B}) \supset \overline{K}$.
2. Я заплатил бы за работу по ремонту телевизора $[Z]$, только если бы он стал работать $[R]$. Он же не работает $[\neg R]$. Поэтому я платить не буду $[\neg Z]$. *Ответ*: $((Z \supset R) \cdot \overline{R}) \supset \overline{Z}$.

Задача *Кислера*. Браун, Джонс и Смит обвиняются в подделке сведений о подлежащих налоговому обложению доходах. Они дают под присягой следующие показания.

*Браун* : Джонс виновен, а Смит невиновен.

*Джонс*: Если Браун виновен, то виновен и Смит.

*Смит* : Я невиновен, но хотя бы один из них двоих виновен.

Обозначим через Б, Д и С высказывания: «Браун виновен», «Джонс виновен», «Смит виновен». Выразите показания каждого из обвиняемых логической формулой. Постройте таблицы истинности трех полученных формул. Затем ответьте на следующие вопросы.

1. Совместны ли показания всех троих заподозренных (т. е. могут ли они быть верны одновременно)?
2. Показания одного из обвиняемых следуют из показаний другого; о чьих показаниях идет речь?
3. Если все трое невиновны, то кто совершил лжесвидетельство?
4. Предполагая, что показания всех обвиняемых верны, укажите, кто невиновен, а кто виновен?
5. Если невиновный говорит истину, а виновный лжет, то кто невиновен, а кто виновен?

### 1.2.6. Алгебра Буля

Объектами алгебры Буля (Д. Буль, 1847) является множество $P_2$ всех булевых функций (п. 1.2.1). Основные операции алгебры: $\neg$, $\wedge$, $\vee$.

#### Аксиомы алгебры

Пусть $x, y, z$ — произвольные булевы функции.

I. *Аксиомы для особых элементов* 0 и 1:

1. $x \vee \overline{x} = 1$;
2. $x \cdot \overline{x} = 0$.

II. *Аксиомы для отрицания*:

1. $\overline{1} = 0$;
2. $\overline{0} = 1$;
3. $\overline{\overline{x}} = x$ — снятие двойного отрицания.

III. *Аксиомы для конъюнкции*:

1. $x \cdot y = y \cdot x$ — коммутативность;
2. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ — ассоциативность;
3. $x \cdot x = x$ — иденпотентность;
4. $x \cdot 1 = x$;
5. $x \cdot 0 = 0$.

IV. *Аксиомы для дизъюнкции*:

1. $x \vee y = y \vee x$ — коммутативность;
2. $(x \vee y) \vee z = x \vee (y \vee z)$ — ассоциативность;
3. $x \vee x = x$ — иденпотентность;
4. $x \vee 1 = 1$;
5. $x \vee 0 = x$.

V. *Аксиомы для конъюнкции и дизъюнкции*:

1. $x \cdot (y \vee z) = x \cdot y \vee x \cdot z$ — дистрибутивность $\cdot$ относительно $\vee$;
2. $x \vee (y \cdot z) = (x \vee y) \cdot (x \vee z)$ — дистрибутивность $\vee$ относительно $\cdot$;
3. $x \vee x \cdot y = x$ — поглощение конъюнктивного члена;
4. $x \cdot (x \vee y) = x$ — поглощение дизъюнктивного члена.

VI. *Аксиомы для отрицания, конъюнкции и дизъюнкции*:

1. $\overline{x \cdot y} = \overline{x} \vee \overline{y}$ — 1-й закон А. де Моргана (XIX в.);
2. $\overline{x \vee y} = \overline{x} \cdot \overline{y}$ — 2-й закон А. де Моргана.

Замечание. Аксиомы не доказываются в рамках той же теории, однако справедливость их всегда можно проверить в рамках другой теории. В нашем случае — с помощью таблиц истинности. Проверим справедливость *первого закона де Моргана*: $\overline{x \cdot y} = \overline{x} \vee \overline{y}$, вычислив таблицы истинности для его левой и правой частей.

| № | $x\ y$ | $x\,y$ | $\overline{x}$ | $\overline{y}$ | $\overline{x\,y}$ | $\overline{x} \vee \overline{y}$ |
|---|---|---|---|---|---|---|
| 0 | 0 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 1 | 0 | 1 | 0 | 1 | 1 |
| 2 | 1 0 | 0 | 0 | 1 | 1 | 1 |
| 3 | 1 1 | 1 | 0 | 0 | 0 | 0 |

В данной системе аксиом не все аксиомы являются независимыми. Это означает, что имеется возможность вывода каких-либо приведенных аксиом из других аксиом. Как правило, авторы сознательно идут на такую избыточность с целью упростить процесс вывода и преобразования формул.

Пример. Выведем аксиому V.2: $x \vee (y \cdot z) = (x \vee y) \cdot (x \vee z)$.

Решение. В качестве исходного выражения для преобразования возьмем правую часть указанной аксиомы и приведем ее к выражению левой части этой аксиомы. Итак, имеем $(x \vee y) \cdot (x \vee z) \overset{\text{V.1}}{=} (x \vee y) \cdot x \ \vee (x \vee y) \cdot z \overset{\text{III.1}}{=} x \cdot (x \vee y) \vee z \times \times (x \vee y) \overset{\text{V.1}}{=} xx \vee xy \vee zx \vee zy \overset{\text{V.1}}{=} \left| xx \overset{\text{III.3}}{=} x \right| = x \vee xy \vee zx \vee zy = = \left| x \vee xy \overset{\text{V.3}}{=} x \right| = x \ \vee zx \vee zy = \left| x \vee zx \overset{\text{V.3}}{=} x \right| = x \ \vee zy.$

Замечание. Система аксиом дает возможность приводить произвольные булевы функции к стандартному их виду — это дизъюнктивные и конъюнктивные формы булевых функций.

### 1.2.7. Совершенные дизъюнктивная и конъюнктивная нормальные формы (СДНФ и СКНФ) булевых функций

В данном разделе рассматриваются стандартные формы булевых функций. Отметим, что приведение к стандартному виду выполняется в рамках алгебры Буля (п. 1.2.6). По ходу изложения вводится ряд понятий и определений с целью упрощения и ясности изложения материала.

Определение. *Элементарным произведением* (ЭП) булевых переменных $x_1, x_2, \ldots, x_n \in X$ называется конъюнкция указанных переменных, взятых по одному с отрицанием или без него. *Например*, $x \cdot \overline{y} \cdot \overline{z}$ — элементарное произведение, $x \cdot \overline{y \cdot z}$ — неэлементарное произведение.

Определение. *Дизъюнктивной нормальной формой* (ДНФ) булевой функции называется дизъюнкция элементарных произведений. *Например*, $f(x, y, z, w) = x\,\overline{y}\,\overline{z} \vee \overline{y}\,z \vee \overline{x}\,z\,w \vee \overline{y}\,\overline{w}$.

Определение. *Конституентой* 1 набора значений $(0\,1\,1\,0 \ldots 0)$ булевых переменных $x_1, x_2, \ldots, x_n \in X$ называется конъюнкция всех указанных переменных, взятых по одному с отрицанием или без него. Переменная входит с отрицанием, если ее значение в наборе равно 0. Для исходного набора *конституента* 1 имеет вид $\overline{x}_1\, x_2\, x_3 \overline{x}_4 \ldots \overline{x}_n$. Ясно, что *конституента* 1 является элементарным произведением. *Например*,

$(0\,1\,0)$ — набор значений переменных $x, y, z$, тогда $\overline{x}\,y\,\overline{z}$ — конституента 1 этого набора.

Утверждение 1.2.1. *Конституента* 1 *как булева функция равна* 1 *лишь на том наборе, для которого она составлена.*

Справедливость следует непосредственно из определения конституенты 1.

Определение. *Конституентой* 0 набора значений $(0\,1\,1\,0\ldots0)$ булевых переменных $x_1, x_2, \ldots, x_n \in X$ называется дизъюнкция всех указанных переменных, взятых по одному с отрицанием или без него. Переменная входит с отрицанием, если ее значение в наборе равно 1. Для исходного набора *конституента* 0 имеет вид $x_1 \vee \overline{x}_2 \vee \overline{x}_3 \vee x_4 \vee \ldots \vee x_n$. *Например*, если $(0\,1\,0)$ — набор значений переменных $x, y, z$, то $x \vee \overline{y} \vee z$ — конституента 0 этого набора.

Утверждение 1.2.2. *Конституента* 0 *как булева функция равна* 0 *лишь на том наборе, для которого она составлена.*

Справедливость следует непосредственно из определения конституенты 0.

Утверждение 1.2.3. *В соответствии с законами де Моргана* (*п.* 1.2.6) *конституента* 1 *и конституента* 0 *являются отрицанием друг друга.*

### Совершенная дизъюнктивная нормальная форма булевых функций

Заполним таблицу истинности (табл. 1.7) булевой функции $f(x_1, x_2, \ldots, x_n)$, $n$ — число аргументов.

Таблица 1.7

| № | $x_1$ | $x_2$ | … | $x_n$ | $f$ | $K^1$ | $K^0$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | … | 0 | $f_0$ | $K^1_0$ | $K^0_0$ |
| 1 | 0 | 0 | … | 1 | $f_1$ | $K^1_1$ | $K^0_1$ |
| 2 | 0 | 1 | … | 0 | $f_2$ | $K^1_2$ | $K^0_2$ |
| ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ |
| $\ell$ | 0 | 1 | … | 1 | $f_\ell$ | $K^1_\ell$ | $K^0_\ell$ |
| ⋮ | ⋮ | ⋮ | | ⋮ | ⋮ | ⋮ | ⋮ |
| $2^n-1$ | 1 | 1 | … | 1 | $f_{2^n-1}$ | $K^1_{2^n-1}$ | $K^0_{2^n-1}$ |

В табл. 1.7 приняты следующие обозначения: $f$ — указывает столбец значений функции $f(x_1, x_2, \ldots, x_n)$; $f_i \in \{0, 1\}$ — значение функции $f(x_1, x_2, \ldots, x_n)$ на $i$-м наборе; $K^1$ — столбец конституент 1; $K^1_i$ — конституента 1 $i$-го набора; $K^0$ — столбец конституент 0; $K^0_i$ — конституента 0 $i$-го набора. *Например*, $K^1_i = \overline{x}_1 \cdot x_2 \cdot \overline{x}_3 \cdot \ldots \cdot \overline{x}_n$, а $K^0_i = x_1 \vee \overline{x}_2 \vee x_3 \vee \ldots \vee x_n$. По табл. 1.7 составим формулу

$$F_1(x_1, x_2, \ldots, x_n) = \\ = f_0 K^1_0 \vee f_1 K^1_1 \vee \ldots \vee f_{2^n-1} K^1_{2^n-1} = \bigvee_{i=0}^{2^n-1} f_i K^1_i. \quad (1.15)$$

Теорема 1.2.1 (о СДНФ). *Формула* $F_1(x_1, x_2, \ldots, x_n)$ (1.15) *совпадает с исходной булевой функцией* $f(x_1, x_2, \ldots, x_n)$.

Доказательство. Функция $f$ и формула $F_1$ совпадают, если совпадают их таблицы истинности. Для этого достаточно проверить, что они равны на произвольном $\ell$-м наборе:

$$F_1\Big|_\ell = f_0 K^1_0\Big|_\ell \vee f_1 K^1_1\Big|_\ell \vee \ldots \vee f_{2^n-1} K^1_{2^n-1}\Big|_\ell = \\ = \left| \text{ Из утверждения (1.2.1) } K^1_i|_\ell = \begin{cases} 0, & \text{если } i \neq \ell, \\ 1, & \text{если } i = \ell. \end{cases} \right| = \\ = f_0 \cdot 0 \vee f_1 \cdot 0 \vee \ldots \vee f_\ell \cdot 1 \vee \ldots \vee f_{2^n-1} \cdot 0 = f_\ell. \qquad \square$$

### Упрощение формулы $F_1(x_1, x_2, \ldots, x_n)$

Так как $f_i K^1_i = \begin{cases} 0, & \text{если } f_i = 0, \\ K^1_i, & \text{если } f_i = 1, \end{cases}$ то

$$F_1(x_1, x_2, \ldots, x_n) = \bigvee_{i=0}^{2^n-1} f_i K^1_i = K^1_{m_1} \vee K^1_{m_2} \vee \ldots \vee K^1_{m_r},$$

где $K^1_{m_j}$ — конституенты 1 тех наборов, на которых $f$ равна 1.

Таким образом, справедливо представление

$$f(x_1, x_2, \ldots, x_n) = K^1_{m_1} \vee K^1_{m_2} \vee \ldots \vee K^1_{m_r}. \quad (1.16)$$

Определение. *Совершенная дизъюнктивная нормальная форма* (СДНФ) булевой функции $f(x_1, x_2, \ldots, x_n)$ есть (1.16) — дизъюнкция конституент 1 всех тех наборов, на которых данная булева функция равна 1.

### Совершенная конъюнктивная нормальная форма булевых функций

По табл. 1.7 составим формулу

$$F_2(x_1, x_2, \ldots, x_n) = (f_0 \vee K_0^0) \cdot (f_1 \vee K_1^0) \cdot \ldots \cdot (f_{2^n-1} \vee K_{2^n-1}^0) = \bigwedge_{i=0}^{2^n-1} (f_i \vee K_i^0). \quad (1.17)$$

Теорема 1.2.2 (о СКНФ). *Формула* $F_2(x_1, x_2, \ldots, x_n)$ (1.17) *совпадает с исходной булевой функцией* $f(x_1, x_2, \ldots, x_n)$.

Доказательство. Функция $f$ и формула $F_2$ совпадают, если совпадают их таблицы истинности. Для этого достаточно проверить, что они равны на произвольном $\ell$-м наборе.

$$F_2|_\ell = (f_0 \vee K_0^0|_\ell) \cdot (f_1 \vee K_1^0|_\ell) \cdot \ldots \cdot (f_{2^n-1} \vee K_{2^n-1}^0|_\ell) =$$

$$= \left| \text{ Из утверждения (1.2.2) } K_i^0|_\ell = \begin{cases} 1, & \text{если } i \neq \ell, \\ 0, & \text{если } i = \ell, \end{cases} \right| =$$

$$= (f_0 \vee 1) \cdot (f_1 \vee 1) \cdot \ldots \cdot (f_\ell \vee 0) \cdot \ldots \cdot (f_{2^n-1} \vee 1) = f_\ell. \qquad \square$$

### Упрощение формулы $F_2(x_1, x_2, \ldots, x_n)$

Так как $f_i \vee K_i^0 = \begin{cases} 1, & \text{если } f_i = 1, \\ K_i^0, & \text{если } f_i = 0, \end{cases}$ то

$$F_2(x_1, x_2, \ldots, x_n) = \bigwedge_{i=0}^{2^n-1} (f_i \vee K_i^0) = K_{n_1}^0 \cdot K_{n_2}^0 \cdot \ldots \cdot K_{n_s}^0,$$

где $K_{n_j}^0$ — конституенты 0 тех наборов, на которых $f$ равна 0.

Таким образом, справедливо представление

$$f(x_1, x_2, \ldots, x_n) = K_{n_1}^0 \cdot K_{n_2}^0 \cdot \ldots \cdot K_{n_s}^0. \quad (1.18)$$

Определение. *Совершенная конъюнктивная нормальная форма* (СКНФ) булевой функции $f(x_1, x_2, \ldots, x_n)$ есть (1.18) — конъюнкция конституент 0 всех тех наборов, на которых данная булева функция равна 0.

Следствие. *Теоремы об* СДНФ *и* СКНФ *позволяют записать в аналитическом виде любую булеву функцию множества* $P_2$, *заданную таблично.*

Следствие. *Набор булевых функций* $\sigma = \{\neg, \wedge, \vee\}$ *является полным, так как с помощью этого набора операций можно представить любую булеву функцию* $P_2$.

Пример. Составим СДНФ и СКНФ следующих булевых функций: $x \to y$, $x \oplus y$, $x \nrightarrow y$ и $x \downarrow y$. Соответствующая им таблица истинности (табл. 1.5) приобретает следующий вид:

| № | $x\ y$ | $x \to y$ | $x \oplus y$ | $x \nrightarrow y$ | $x \downarrow y$ |
|---|---|---|---|---|---|
| 0 | 0 0 | 1 | 0 | 0 | 1 |
| 1 | 0 1 | 1 | 1 | 1 | 0 |
| 2 | 1 0 | 0 | 1 | 0 | 0 |
| 3 | 1 1 | 1 | 0 | 0 | 0 |

Отсюда

$$
\begin{aligned}
(x \to y)_{\text{СДНФ}} &= \overline{x}\,\overline{y} \vee \overline{x}\,y \vee x\,y,\\
(x \oplus y)_{\text{СДНФ}} &= \overline{x}\,y \vee x\,\overline{y},\\
(x \nrightarrow y)_{\text{СДНФ}} &= \overline{x}\,y,\\
(x \downarrow y)_{\text{СДНФ}} &= \overline{x}\,\overline{y}.
\end{aligned}
$$

$$
\begin{aligned}
(x \to y)_{\text{СКНФ}} &= \overline{x} \vee y,\\
(x \oplus y)_{\text{СКНФ}} &= (x \vee y)\,(\overline{x} \vee \overline{y}),\\
(x \nrightarrow y)_{\text{СКНФ}} &= (x \vee y)\,(\overline{x} \vee y)\,(\overline{x} \vee \overline{y}),\\
(x \downarrow y)_{\text{СКНФ}} &= (x \vee \overline{y})\,(\overline{x} \vee y)\,(\overline{x} \vee \overline{y}).
\end{aligned}
$$

Замечание. Вернемся к интерпретации булевых функций релейно-контактными схемами (п. 1.2.5). Представление функций в форме СДНФ и СКНФ снимает все вопросы относительно составления контактных схем булевых функций любой сложности. Достаточно представлять себе контактные схемы булевых функций $\neg, \wedge, \vee$ (п. 1.2.5).

## § 1.3. Минимизация булевых функций

Под *минимизацией* булевых функций будем понимать их представление в самом экономичном коротком виде. Задачу минимизации будем решать в области дизъюнктивных нормальных форм (ДНФ) булевых функций (п. 1.2.7). В области ДНФ минимизация доведена до конечного алгоритма.

### 1.3.1. Классификация двоичных наборов

Пусть $A = (a_1, a_2, \ldots, a_n)$, $B = (b_1, b_2, \ldots, b_n)$, где $a_i, b_i \in \{0, 1\}$, — произвольные двоичные наборы.

1. Наборы $A$ и $B$ — *соседние*, если отличаются ровно одной координатой. *Например*, $A = (1101110011)$, $B = (1101110001)$ — соседние наборы.
2. Наборы $A$ и $B$ — *несоседние*, если отличаются более, чем одной координатой. *Например*, $A = (1100010011)$, $B = (1101110001)$ — несоседние наборы.
3. Наборы $A$ и $B$ — *противоположные*, если являются отрицанием друг друга. *Например*, $A = (1110000001)$, $B = (0001111110)$ — противоположные наборы.
4. Наборы $A$ и $B$ — *сравнимые*, если одновременно для всех $i = 1, 2, \ldots, n$ выполняется $a_i \geqslant b_i$ или $a_i \leqslant b_i$. *Например*, $A = (1110001111)$, $B = (1010001100)$ — сравнимые наборы.
5. Сравнимые наборы делятся на *большие* и *меньшие*. Пусть наборы $A$ и $B$ — *сравнимые* и для всех $i = 1, 2, \ldots, n$ выполняется $a_i \geqslant b_i$, тогда записывают $A \geqslant B$. Если же для всех $i = 1, 2, \ldots, n$ выполняется $a_i > b_i$, то записывают $A > B$.

### 1.3.2. Геометрическая интерпретация задачи минимизации булевых функций

Рассмотрим пример минимизации булевой функции трех переменных $f(x, y, z)$, заданной таблично (табл. 1.8).

Таблица 1.8

| $A$ | $x$ $y$ $z$ | $f(x, y, z)$ | $K^1$ |
|---|---|---|---|
| $A_0$ | 0 0 0 | 0 | $\overline{x}\,\overline{y}\,\overline{z}$ |
| $A_1$ | 0 0 1 | 0 | $\overline{x}\,\overline{y}\,z$ |
| $A_2$ | 0 1 0 | 1 | $\overline{x}\,y\,\overline{z}$ |
| $A_3$ | 0 1 1 | 0 | $\overline{x}\,y\,z$ |
| $A_4$ | 1 0 0 | 1 | $x\,\overline{y}\,\overline{z}$ |
| $A_5$ | 1 0 1 | 1 | $x\,\overline{y}\,z$ |
| $A_6$ | 1 1 0 | 1 | $x\,y\,\overline{z}$ |
| $A_7$ | 1 1 1 | 1 | $x\,y\,z$ |

В таблице $A$ — столбец обозначений наборов; $A_i$ — $i$-й порядковый набор в таблице истинности; $K^1$ — столбец конституент 1 для каждого набора таблицы истинности.

Рассмотрим графическое представление (п. 1.2.1) выбранной функции $f(x, y, z)$. Область определения $f(x, y, z)$ — множество наборов $\{A_0, A_1, \ldots, A_7\}$ координат точек вершин единичного трехмерного куба (рис. 1.7). В соответствии с таблицей истинности (табл. 1.8) отметим вершины, в которых булева функция равна 1. *Представленный таким образом куб — это объемная таблица истинности булевой функции. Объемное представление таблицы истинности позволяет визуально оценивать распределение вершин по кубу, в которых функция равна* 1.

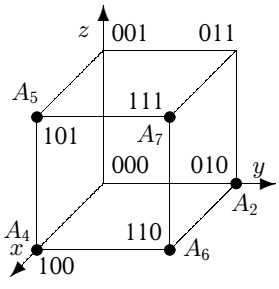


Рис. 1.7

Запишем рассматриваемую функцию в СДНФ. При записи конституент 1 в СДНФ будем придерживаться такой их последовательности, где учитывается отношение соседства вершин на кубе (рис. 1.7), в которых функция равна 1. Полагаем, что две вершины соседние, если их наборы являются соседними, т. е. отличаются одной координатой. На кубе соседние вершины — это вершины одного ребра. Итак,

$$f(x,y,z) = \underbrace{x\,\overline{y}\,\overline{z}}_{A_4} \vee \underbrace{x\,\overline{y}\,z}_{A_5} \vee \underbrace{x\,y\,\overline{z}}_{A_6} \vee \underbrace{x\,y\,z}_{A_7} \vee \underbrace{\overline{x}\,y\,\overline{z}}_{A_2}.$$

Используя аксиомы булевой алгебры, упростим $f(x, y, z)$. Заметим, что каждая пара соседних вершин имеет свою общую часть, чем и воспользуемся. Тогда

$$f(x,y,z) = \underbrace{x\,\overline{y}\,(\overline{z} \vee z)}_{A_4A_5} \vee \underbrace{x\,y\,(\overline{z} \vee z)}_{A_6A_7} \vee \underbrace{\overline{x}\,y\,\overline{z}}_{A_2} = \underbrace{x\,\overline{y}}_{A_4A_5} \vee \underbrace{x\,y}_{A_6A_7} \vee \underbrace{\overline{x}\,y\,\overline{z}}_{A_2}.$$

После «склеивания» соседних вершин их общую часть можно интерпретировать ребрами, которые их *покрывают*. Ребро представляется конъюнкцией двух переменных. Значит, последнее выражение для $f(x, y, z)$ включает ребро $A_4A_5$, ребро $A_6A_7$ и вершину $A_2$.

Ребра $A_4A_5$ и $A_6A_7$ также являются соседними, соответствующие им выражения (наборы) отличаются одной координатой. Выполним склеивание таких ребер:

$$f(x,y,z)=\underbrace{x\,(\overline{y}\vee y)}_{A_4A_5A_6A_7}\vee\underbrace{\overline{x}\,y\,\overline{z}}_{A_2}=\underbrace{x}_{A_4A_5A_6A_7}\vee\underbrace{\overline{x}\,y\,\overline{z}}_{A_2}.$$

Общую часть соседних ребер будем интерпретировать гранью, которая их покрывает. Грань представляется одной переменной. Теперь выражение для $f(x,y,z)$ включает одну грань $A_4A_5A_6A_7$ и вершину $A_2$.

*Геометрическая интерпретация* выполненных аналитических преобразований, направленных на упрощение исходной функции $f(x,y,z)$, дает основание заключить, что уменьшение длины выражения $f(x,y,z)$ возможно лишь в том случае, если имеется в наличие хотя бы одна пара соседних объектов (вершин, ребер и т. д.).

Вернемся к объемной таблице истинности (рис. 1.7) исходной булевой функции $f(x,y,z)$. Здесь вершина $A_2$ имеет соседнюю вершину $A_6$, в которой функция равна 1. Склеиванием указанных вершин, с целью уменьшения длины $f(x,y,z)$, мы не воспользовались, так как вершина $A_6$ участвовала в склеивании с вершиной $A_7$.

Однако и в этом случае длина $f(x,y,z)$ еще может быть уменьшена. Воспользуемся свойством операции дизъюнкции: $g\vee h\vee h=\ g\vee(h\vee h)=\ g\vee h$, где $g$ и $h$ — произвольные функции. Это означает, что булева функция не изменяется при повторном многократном включении ее дизъюнктивных членов.

Воспользуемся этим свойством и добавим повторно в $f(x,y,z)$ ее же вершину $A_6$ как соседнюю с $A_2$. Склеивание этих вершин уменьшит длину последнего выражения $f(x,y,z)$ на 1:

$$f(x,y,z)=\underbrace{x}_{A_4A_5A_6A_7}\vee\underbrace{\overline{x}\,y\,\overline{z}}_{A_2}\vee\underbrace{x\,y\,\overline{z}}_{A_6}=$$

$$=\underbrace{x}_{A_4A_5A_6A_7}\vee\underbrace{y\,\overline{z}\,(\overline{x}\vee x)}_{A_2A_6}=\underbrace{x}_{A_4A_5A_6A_7}\vee\underbrace{y\,\overline{z}}_{A_2A_6}.$$

Последнее выражение позволяет заключить, что вершина $A_6$ оказалась покрытой дважды: гранью $A_4A_5A_6A_7$ и ребром $A_2A_6$.

*Сформулируем общие правила аналитического описания геометрических объектов* (*вершин*, *ребер*, *граней и т. д.*).

1. Выписывается конъюнкция тех переменных, которые не изменяются в пределах данного геометрического объекта.

2. Переменная берется с отрицанием, если на данном геометрическом объекте она равна 0 и без отрицания, если равна 1.

**Основные правила минимизации булевых функций в геометрической интерпретации**

1. Пусть $f(x_1, x_2, \ldots, x_n)$ — исходная булева функция, которую необходимо минимизировать. Рассмотрим ее графическое представление. Множество наборов области определения функции $f(x_1, x_2, \ldots, x_n)$ — это множество координат точек вершин единичного $n$-мерного куба. Отметим вершины такого куба, в которых функция равна 1. Построенный таким образом $n$-мерный куб является объемной таблицей истинности исходной функции.
2. *Минимизация* $f(x_1, x_2, \ldots, x_n)$ в геометрической интерпретации заключается в том, чтобы *покрыть* (склеить) все вершины единичного $n$-мерного куба, в которых булева функция равна 1, геометрическими объектами (вершинами, ребрами, гранями и т. д.) *максимально возможной размерности*. Покрывать следует таким образом, чтобы ни одна из вершин, в которых функция равна 0, не оказалась покрытой.
3. Вершины куба можно покрывать произвольное число раз. Значение функции при этом не изменяется.
4. Минимальная ДНФ функции $f(x_1, x_2, \ldots, x_n)$ — это дизъюнкция геометрических объектов (их аналитических представлений), участвовавших в покрытии вершин куба, в которых функция равна 1.

Пример. Найти минимальную ДНФ булевой функции $f(x, y, z)$, заданной таблично (табл. 1.9).

Таблица 1.9

| $A$ | $x$ $y$ $z$ | $f(x,y,z)$ | $K^1$ |
|---|---|---|---|
| $A_0$ | 0 0 0 | 0 | $\overline{x}\,\overline{y}\,\overline{z}$ |
| $A_1$ | 0 0 1 | 1 | $\overline{x}\,\overline{y}\,z$ |
| $A_2$ | 0 1 0 | 1 | $\overline{x}\,y\,\overline{z}$ |
| $A_3$ | 0 1 1 | 0 | $\overline{x}\,y\,z$ |
| $A_4$ | 1 0 0 | 1 | $x\,\overline{y}\,\overline{z}$ |
| $A_5$ | 1 0 1 | 0 | $x\,\overline{y}\,z$ |
| $A_6$ | 1 1 0 | 1 | $x\,y\,\overline{z}$ |
| $A_7$ | 1 1 1 | 1 | $x\,y\,z$ |

На рис. 1.8 представлен куб пространственной таблицы истинности рассматриваемой функции. Точками выделены вершины, в которых $f(x,y,z)$ равна 1. Геометрический объект максимально возможной размерности, которым можно покрывать вершины куба данного примера, — это ребро. Покрываем только те вершины, в которых функция равна 1. Вершины можно покрыть тремя ребрами $A_4A_6$, $A_7A_6$, $A_2A_6$ и одной вершиной $A_1$. Отсюда минимальная ДНФ имеет вид

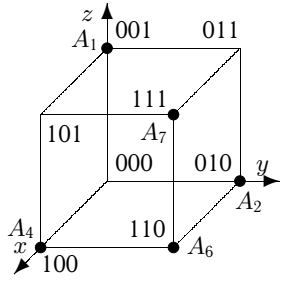


Рис. 1.8

$$f(x,y,z)_{\min\limits_{\text{ДНФ}}} = \underbrace{x\,\overline{z}}_{A_4A_6} \vee \underbrace{x\,y}_{A_7A_6} \vee \underbrace{y\,\overline{z}}_{A_2A_6} \vee \underbrace{\overline{x}\,\overline{y}\,z}_{A_1}.$$

### 1.3.3. Метод Карно минимизации булевых функций 4-х переменных

В основе геометрической интерпретация задачи минимизации (п. 1.3.2) булевых функций $f(x_1,x_2,\ldots,x_n)$ лежит пространственная структура единичного $n$-мерного куба. Успешное использование геометрической интерпретации в задаче минимизации функций 3-х переменных обусловлено наглядностью представления отношения соседства вершин в *трехмерном* кубе. С увеличением размерности $n$ задачи возрастает число связей между вершинами и существенно падает наглядность представления. Так, для функций 4-х переменных $f(x,y,z,w)$ решение указанным способом уже становится не очевидным. Например, на рис. 1.9 представлен *четырехмерный* куб. Такой куб строится перемещением трехмерного куба ($x,y,z$ — измерения) по четвертому измерению ($w$).

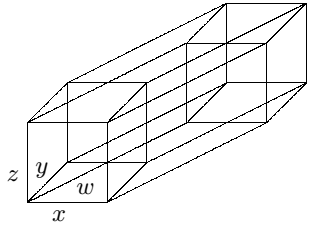


Рис. 1.9

Метод Карно также является геометрическим, минимизация выполняется в рамках основных правил минимизации булевых функций в геометрической интерпретации. В основу метода положено удобное представление четырехмерного куба на плоскости, которое называется картой Карно (рис. 1.10, слева). Представление *сохраняет отношение соседства вершин*.

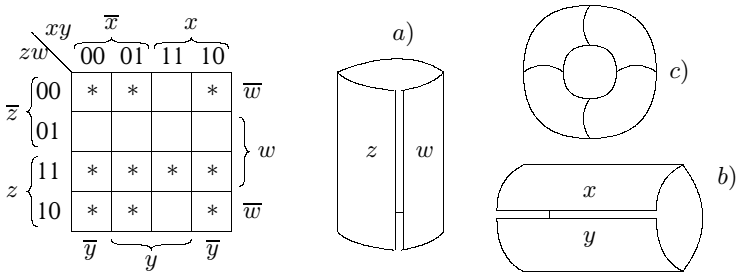



Рис. 1.10. Карта Карно

### Структура карты Карно (рис. 1.10)

1. *Одна клетка* карты соответствует вершине (нульмерная грань) четырехмерного куба (рис. 1.9).
2. Соседним вершинам четырехмерного куба соответствую *соседние клетки* карты Карно. *Соседние клетки карты имеют общую сторону*. Отношение соседства клеток на карте Карно имеет несколько особенностей, что *очень важно* для минимизации функций. При определении соседних клеток следует представлять, что горизонтальные и вертикальные края карты склеены в цилиндры так, как это показано на рис. 1.10, где a) — вертикальное склеивание и b) — горизонтальное склеивание. Если же выполнить горизонтальное склеивание карты, которую ранее склеили вертикально, то получим геометрическую фигуру — *тор* (бублик, двойной цилиндр) (рис. 1.10, c). Таким образом, у карты Карно нет крайних клеток, все они равнозначны. *Соседство клеток* оценивается по их относительному расположению на *торе* (двойной цилиндр).
3. *Две* соседних клетки карты соответствуют *ребру* (одномерная грань).
4. *Четыре* соседних клетки карты соответствуют *двумерной грани*. На карте двумерные грани составляются из двух ребер (две пары соседних клеток). Расположение таких ребер возможно либо в один ряд, либо квадратом. На рис. 1.10 можно найти примеры конфигураций такого рода соседства клеток, отмеченых звездочками $(*)$. Если вы все правильно поняли, то должны опознать здесь *четыре* двумерные грани.
5. *Восемь* соседних клеток карты соответствуют *трехмерной грани* (трехмерный куб). На карте трехмерные грани составляются из двух соседних двухмерных граней.
6. *Шестнадцать* соседних клеток карты соответствуют *четырехмерной грани* (четырехмерный куб). На карте четырех-

мерные грани составляются из двух соседних трехмерных граней.

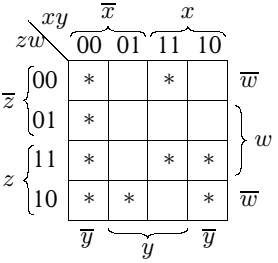



Рис. 1.11

Пример. Найти минимальную ДНФ булевой функции, карта Карно (таблица истинности) которой представлена на рис. 1.11.

Решение. Отмеченные клетки карты соответствуют координатам вершин единичного четырехмерного куба, в которых функция равна 1. Максимальная *размерность грани*, которой можно покрывать клетки указанной карты, равна *двум*. Таких граней здесь две — $\overline{x}\,\overline{y}$ и $\overline{y}\,z$. Оставшиеся вершины покрываем двумя ребрами — $\overline{x}\,z\,\overline{w}$, $x\,z\,w$, и одной оставшейся вершиной — $x\,y\,\overline{z}\,\overline{w}$. Отсюда $f(x,y,z,w)_{\min\limits_{\text{ДНФ}}} = \overline{x}\,\overline{y} \vee \overline{y}\,z \vee \overline{x}\,z\,\overline{w} \vee \vee x\,z\,w \vee x\,y\,\overline{z}\,\overline{w}$.

Пример. Найти минимальную ДНФ булевой функции, карта Карно (таблица истинности) которой представлена на рис. 1.10.

Решение. Отмеченные клетки карты соответствуют координатам вершин единичного четырехмерного куба, в которых функция равна 1. Максимальная *размерность грани*, которой можно покрывать клетки указанной карты, равна *двум*. Все отмеченные клетки покрываются тремя такими гранями. Отсюда минимальная ДНФ равна $f(x,y,z,w)_{\min\limits_{\text{ДНФ}}} = z\,w \vee \overline{x}\,\overline{w} \vee \overline{y}\,\overline{w}$.

### 1.3.4. Аналитический метод Куайна минимизации булевых функций

Введем определения и понятия, используемые при описании метода Куайна минимизации булевых функций. Отметим, что в итоге данного раздела излагается конечный алгоритм минимизации произвольной булевой функции. Алгоритм вполне пригоден для его программной реализации.

#### Классификация дизъюнктивных нормальных форм булевых функций

Рассматриваемая класификация булевых функций привязывается к этапам минимизации метода Куайна. Последовательность форм булевых функций этапов минимизации представляется следующим их списком:

1. ДНФ;
2. СДНФ — исходная форма булевой функции;
3. Сокращенная ДНФ;
4. Тупиковая ДНФ;
5. Минимальная ДНФ.

Определение. *Импликантой* $f_i$ булевой функции $f$ называется *элементарное произведение*, которое равно нулю на всех тех наборах, где $f$ равна 0. Это эквивалентно тому, что импликация $f_i \to f$ является тождественно истинной формулой (следует из определения импликанты).

Утверждение 1.3.1. *Пусть* $f = A \vee B$ *и* $A, B, C$ *— элементарные произведения. Тогда* $A$, $B \cdot C$ *— импликантны.*

Доказательство. Достаточно проверить утверждение для конъюнкции $B \cdot C$, т. е. показать, что импликация $B \times \times C \to f$ — тождественно истинная формула. Запишем импликацию $B \cdot C \to f$ в СКНФ. Тогда $B \cdot C \to f = \overline{B \cdot C} \vee f = \overline{B} \cdot \overline{C} \vee (A \vee B) = (\overline{B} \vee \overline{C}) \vee (A \vee B) = (\overline{B} \vee B) \vee A \vee \overline{C} = 1 \vee A \vee \overline{C} = 1$. □

Итак, а) *дизъюнктивный член* $A$ функции является ее *импликантой*; б) импликантой функции является *элементарное произведение* $B \cdot C$, если часть его $B$ является дизъюнктивным членом функции. В соответствии с утверждением 1.3.1 дадим следующее определение.

Определение. *Импликанта* функции называется *простой*, если сама она является дизъюнктивным членом функции, а никакая ее часть не является таковой.

Пример. Пусть $f(x, yz) = x\,\overline{z} \vee x\,y\,z \vee \overline{y}\,\overline{z} \vee \overline{y}$. Тогда
$x\,\overline{z}$, $x\,y\,z$, $\overline{y}$ — простые импликанты,
$\overline{y}\,\overline{z}$ — непростая импликанта,
$x\,y\,\overline{z}$ — непростая импликанта,
$x\,y\,z\,\overline{w}$ — непростая импликанта.

Замечание. От непростых импликант функций можно избавляться. Избавимся в последнем примере от непростой импликанты $\overline{y}\,\overline{z}$. Имеем $f(x, y, z) = x\,\overline{z} \vee x\,y\,z \vee \overline{y}\,\overline{z} \vee \overline{y} = x\,\overline{z} \vee x\,y\,z \vee \overline{y}\,(\overline{z} \vee 1) = x\,\overline{z} \vee x\,y\,z \vee \overline{y}$.

Утверждение 1.3.2. ДНФ *булевой функции может быть составлена из простых импликант.*

Доказательство следует из последних определений.

Определение. *Сокращенная* ДНФ функции есть дизъюнкция ее простых импликант.

Определение. *Тупиковая* ДНФ функции есть сокращенная ДНФ, в которой нет ни одной лишней простой импликанты.

Определение. *Минимальная* ДНФ функции есть наименьшая из тупиковых ДНФ.

### Операции упрощения метода Куайна

Пусть $A$ и $B$ — элементарные произведения.

1. *Операция полного склеивания.*
   $Ax \vee A\overline{x} = A\,(x \vee \overline{x}) = A.$
   *Геометрическая интерпретация.* Операция полного склеивания соответствует объединению соседних геометрических объектов $Ax$ и $A\overline{x}$ — вершин, ребер, граней и т. д.
2. *Операция неполного склеивания.*
   $$Ax \vee A\overline{x} = \begin{cases} A \vee Ax \vee A\overline{x}, \\ A \vee Ax, \\ A \vee A\overline{x}. \end{cases}$$
   *Геометрическая интерпретация.* Операция неполного склеивания соответствует повторному покрытию геометрических объектов $Ax$ и $A\overline{x}$ — вершин, ребер, граней и т. д.
3. *Операция элементарного поглощения.*
   $A \vee AB = A\,(1 \vee B) = A.$
   *Геометрическая интерпретация.* Операция поглощения соответствует удалению ранее покрытых геометрических объектов $AB$ объектами большей размерности $A$.

### Алгоритм минимизации метода Куайна

1. *Исходная форма булевой функции* — это ее СДНФ.
2. *Получение сокращенной* ДНФ. Для всех членов исходной булевой функции выполнить все возможные операции полного и неполного склеивания. Затем произвести все возможные операции элементарного поглощения.
3. *Получение тупиковой* ДНФ. Все тупиковые ДНФ булевой функции получаются из сокращенной ДНФ с помощью импликативной матрицы Куайна.
4. *Получение минимальной* ДНФ. Из всех найденных тупиковых ДНФ выбирается ДНФ с минимальной длиной по числу вхождения всех переменных. Минимальных ДНФ у булевой функции может быть несколько, если их длины совпадают.

Изложение каждого из пунктов алгоритма Куайна рассмотрим на конкретном примере минимизации булевой функции:

$$f(x,y,z,w)_{\text{СДНФ}} = x\,y\,\overline{z}\,\overline{w} \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee \overline{x}\,\overline{y}\,\overline{z}\,w \vee \overline{x}\,y\,\overline{z}\,w \vee x\,y\,\overline{z}\,w \vee \overline{x}\,y\,z\,w \vee \overline{x}\,y\,z\,\overline{w} \vee x\,y\,z\,\overline{w} \vee x\,\overline{y}\,z\,\overline{w}. \quad (1.19)$$

**Получение сокращенной ДНФ**

1. Нумеруем все слагаемые булевой функции (1.19) и пытаемся выполнить операцию полного склеивания каждого слагаемого с каждым:

$$f = \underbrace{x\,y\,\overline{z}\,\overline{w}}_{1} \vee \underbrace{x\,\overline{y}\,\overline{z}\,\overline{w}}_{2} \vee \underbrace{\overline{x}\,\overline{y}\,\overline{z}\,w}_{3} \vee \underbrace{\overline{x}\,y\,\overline{z}\,w}_{4} \vee \underbrace{x\,y\,\overline{z}\,w}_{5} \vee \underbrace{\overline{x}\,y\,z\,w}_{6} \vee \underbrace{\overline{x}\,y\,z\,\overline{w}}_{7} \vee \underbrace{x\,y\,z\,\overline{w}}_{8} \vee \underbrace{x\,\overline{y}\,z\,\overline{w}}_{9}.$$

2. Формируем новый список булевой функции. Сначала записываем результаты полного склеивания и отмечаем слагаемые, участвовавшие в операциях. Затем дописываем слагаемые, не принимавшие участия в операциях склеивания. Сохраняем нумерацию слагаемых исходного списка функции:

$$f = \underbrace{x\,\overline{z}\,\overline{w}}_{1,2} \vee \underbrace{x\,y\,\overline{z}}_{1,5} \vee \underbrace{x\,y\,\overline{w}}_{1,8} \vee \underbrace{x\,\overline{y}\,\overline{w}}_{2,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{y\,\overline{z}\,w}_{4,5} \vee \underbrace{\overline{x}\,z\,w}_{4,6} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{y\,z\,\overline{w}}_{7,8} \vee \underbrace{x\,z\,\overline{w}}_{8,9}.$$

3. Если возможно, то повторяем многократно п. 1 и п. 2 алгоритма с новым списком слагаемых функции:

$$f = \underbrace{x\,\overline{w}}_{(1,2),(8,9)} \vee \underbrace{x\,\overline{w}}_{(1,8),(2,9)} \vee \underbrace{x\,y\,\overline{z}}_{1,5} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{y\,\overline{z}\,w}_{4,5} \vee \underbrace{\overline{x}\,z\,w}_{4,6} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{y\,z\,\overline{w}}_{7,8}.$$

4. Для последнего списка слагаемых выполняем операцию *элементарного поглощения*. Получили сокращенную ДНФ:

$$f = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{x\,y\,\overline{z}}_{1,5} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{y\,\overline{z}\,w}_{4,5} \vee \underbrace{\overline{x}\,z\,w}_{4,6} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{y\,z\,\overline{w}}_{7,8}.$$

### Получение тупиковой ДНФ

Тупиковые ДНФ булевой функции строятся с помощью *импликативной матрицы Куайна* и найденной сокращенной ДНФ функции. *Столбцы матрицы* идентифицируются слагаемыми исходной булевой функции $f(x, y, z, w)$. А так как исходная форма функции — СДНФ, то столбцы — *конституенты* 1. *Строки матрицы* Куайна идентифицируются простыми импликантами найденной сокращенной ДНФ булевой функции.

| | Импликанты | | + | ✓ | ✓ | + | | | | + | ✓ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| ✓ | $x\overline{w}$ | 1289 | * | * | | | | | | * | * |
| | $x\,y\,\overline{z}$ | 15 | * | | | | * | | | | |
| ✓ | $\overline{x}\,\overline{z}\,w$ | 34 | | | * | * | | | | | |
| | $y\,\overline{z}\,w$ | 45 | | | | * | * | | | | |
| | $\overline{x}\,z\,w$ | 46 | | | | * | | * | | | |
| | $\overline{x}\,y\,z$ | 67 | | | | | | * | * | | |
| | $y\,z\,\overline{w}$ | 78 | | | | | | | * | * | |

Рис. 1.12. Импликативная матрица Куайна

*Геометрическая интерпретация матрицы Куайна*. Столбцы матрицы — это вершины куба, в которых функция равна 1. Строки матрицы являются простыми импликантами. Импликаты — это геометрические объекты (вершины, ребра, грани и т. д.), которые покрывают указанные вершины куба.

При построении сокращенной ДНФ устанавливалась нумерация исходных слагаемых СДНФ булевой функции и нумерация импликант сокращенной ДНФ. Пусть номера слагаемых СДНФ соответствуют номерам столбцов матрицы Куайна. Тогда составной номер импликанты — это список номеров столбцов, которые покрывает данная импликанта-строка матрицы. Такие номера столбцов отмечаются звездочкой для каждой импликанты матрицы.

Нетрудно заметить, что импликативная матрица Куайна — это представление пространственной таблицы истинности на плоскости, где вместо сохранения отношения соседства вершин представлены уже геометрические объекты-импликанты, покрывающие вершины, в которых функция равна 1.

*Алгоритм поиска тупиковой* ДНФ *заключается в том, чтобы из перечня простых импликант сокращенной* ДНФ *вы-*

*брать минимальный по длине их состав так, чтобы выбранные импликанты покрывали все столбцы матрицы Куайна.*

1. Определяются столбцы с одной звездочкой (∗). В матрице Куайна (рис. 1.12) они отмечены символом ✓. Каждый из таких столбцов покрывается лишь одной импликантой. Значит, такие импликанты всегда должны входить в тупиковые ДНФ. В матрице Куайна (рис. 1.12) строки этих импликант отмечены символом ✓.
2. Далее логика минимизации подсказывает, что найденные обязательные импликанты помимо столбцов с одной звездочкой могут покрывать и другие столбцы. Их также следует отметить как покрытые. Такие столбцы в матрице Куайна (рис. 1.12) отмечены символом +.
3. В итоге могут остаться непокрытыми лишь столбцы, каждый из которых содержит две и более звездочки. Множество таких столбцов покрывается множеством импликант со звездочками в указанных столбцах. Из этого множества импликант необходимо выбрать те, которые имеют наименьшую суммарную длину и покрывают оставшиеся столбцы. Для небольшого числа переменных задачу всегда можно решить полным перебором.

Вернемся к примеру. Непокрытыми остались столбцы $5, 6, 7$. Они покрываются множеством импликант $(1,5)$, $(4,5)$, $(4,6)$, $(6,7)$, $(7,8)$. Видно, что нет одной импликанты, которая покрывала бы все три столбца. Две такие импликанты можно подобрать двумя способами. Первый способ — $(6,7)$, $(1,5)$, и второй — $(6,7)$, $(4,5)$. Им соответствуют тупиковые ДНФ

$$f(x,y,z,w)_{\text{туп}_1} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{x\,y\,\overline{z}}_{1,5},$$

$$f(x,y,z,w)_{\text{туп}_2} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{y\,\overline{z}\,w}_{4,5}.$$

Тремя импликантами столбцы $5, 6, 7$ можно покрыть также двумя способами. Первый способ — $(4,6)$, $(7,8)$, $(1,5)$ и второй — $(4,6)$, $(7,8)$, $(4,5)$:

$$f(x,y,z,w)_{\text{туп}_3} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,z\,w}_{4,6} \vee \underbrace{y\,z\,\overline{w}}_{7,8} \vee \underbrace{x\,y\,\overline{z}}_{1,5},$$

$$f(x,y,z,w)_{\text{туп}_4} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,z\,w}_{4,6} \vee \underbrace{y\,z\,\overline{w}}_{7,8} \vee \underbrace{y\,\overline{z}\,w}_{4,5}.$$

### Получение минимальной ДНФ

Из всех найденных тупиковых ДНФ выбирается та, которая имеет минимальную длину по числу вхождения всех переменных. Минимальных ДНФ у булевой функции может быть несколько, если их длины совпадают. Так, в примере из найденных четырех тупиковых две имеют равную наименьшую длину. Значит, имеем две минимальных ДНФ:

$$f_{min_1} = f_{\text{туп}_1} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{x\,y\,\overline{z}}_{1,5},$$

$$f_{min_2} = f_{\text{туп}_2} = \underbrace{x\,\overline{w}}_{1,2,8,9} \vee \underbrace{\overline{x}\,\overline{z}\,w}_{3,4} \vee \underbrace{\overline{x}\,y\,z}_{6,7} \vee \underbrace{y\,\overline{z}\,w}_{4,5}.$$

## § 1.4. Функциональная полнота булевых функций

Определение. Система булевых функций $F = \{f_1, f_2, \ldots, f_m\}$ из $P_2$ называется (*функционально*) *полной*, если любая булева функция может быть реализована формулой над этой системой, т. е. может быть выражена (представлена) через $f_1, f_2, \ldots, f_m$ с помощью операций замены переменных и суперпозиции (п. 1.2.2).

Утверждение 1.4.1. *Пусть $F = \{f_1, f_2, \ldots, f_m\}$ и $G = \{g_1, g_2, \ldots, g_n\}$ две системы из $P_2$. Известно, что система $F$ — полная и каждая ее функция может быть реализована формулой над системой из $G$. Тогда и система $G$ — полная.*

Пример. Система булевых функций $\sigma = \{\neg, \wedge, \vee\}$ — полная. Действительно, в теоремах об СДНФ (1.2.1) и СКНФ (1.2.2) речь идет именно об этом наборе булевых функций, посредством которого можно записать в аналитическом виде произвольную булеву функцию из $P_2$.

### 1.4.1. Алгебра Жегалкина

Объектами алгебры является множество $P_2$ всех булевых функций (п. 1.2.1). Основные операции алгебры: 1, $\wedge$, $\oplus$.

#### Аксиомы алгебры

Пусть $x, y, z$ — произвольные булевы функции.

I. *Аксиомы для конъюнкции*:

1. $x \cdot y = y \cdot x$ — коммутативность;
2. $(x \cdot y) \cdot z = x \cdot (y \cdot z)$ — ассоциативность;
3. $x \cdot x = x$ — иденпотентность;

4. $x \cdot 1 = x$;
5. $x \cdot 0 = 0$.

II. *Аксиомы для суммы по модулю* 2 ($\oplus$):

1. $x \oplus y = y \oplus x$ — коммутативность;
2. $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ — ассоциативность;
3. $x \oplus x = 0$;
4. $x \oplus 1 = \overline{x}$;
5. $x \oplus 0 = x$.

III. *Аксиома для конъюнкции и суммы по модулю* 2:

1. $x \cdot (y \oplus z) = x \cdot y \oplus x \cdot z$ — дистрибутивность $\cdot$ относительно $\oplus$.

Замечание. Изложение данного раздела повторяет схему изложения алгебры Буля и основной его теоремы об СДНФ (п. 1.2.6, п. 1.2.7).

### Полиномы Жегалкина

Определение. *Полиномом Жегалкина* $P_{\text{ж}}(x_1, x_2, \ldots, x_n)$ от $n$ переменных называется выражение вида

$$\begin{aligned} P_{\text{ж}} = {} & a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_n x_n \oplus \\ & \oplus a_{1,2} x_1 x_2 \oplus a_{1,3} x_1 x_3 \oplus \ldots \oplus a_{n-1,n} x_{n-1} x_n \oplus \\ & \oplus a_{1,2,3} x_1 x_2 x_3 \oplus \ldots \oplus a_{1,2,\ldots,n} x_1 x_2 \cdot \ldots \cdot x_n, \end{aligned} \tag{1.20}$$

где $a_{i_1 i_2 \ldots i_k} \in \{0, 1\}$ — коэффициенты, $x_i$ — булевы переменные.

Определение. Полином Жегалкина называется *линейным*, если его степень не превышает 1. Такой полином имеет следующий вид:

$$P_{\text{ж}}(x_1, x_2, \ldots, x_n) = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus \ldots \oplus a_n x_n. \tag{1.21}$$

Утверждение 1.4.2. *Количество полиномов Жегалкина $n$ переменных* (1.20) *совпадает с числом булевых функций $n$ переменных и равно* $2^{2^n}$.

Доказательство. Пусть $m$ — число слагаемых в полиноме Жегалкина (1.20). Определим это число. Для этого рассмотрим алгебраическое выражение

$$\begin{aligned} (1 + x_1)(1 + x_2) \cdot \ldots \cdot (1 + x_n) = {} & 1 + x_1 + x_2 + \ldots + x_n + \\ & + x_1 x_2 + x_1 x_3 + \ldots + x_{n-1} x_n + \\ & + x_1 x_2 x_3 + \ldots + x_1 x_2 \cdot \ldots \cdot x_n. \end{aligned} \tag{1.22}$$

Число слагаемых в правой части последнего выражения совпадает с числом слагаемых в полиноме Жегалкина (1.20). Число слагаемых же в правой части (1.22) легко сосчитать. Положим в (1.22) все $x_i = 1$, $i = 1, 2, \ldots, n$, тогда в правой части каждое слагаемое обратится в 1 и в сумме дадут число таких слагаемых $m$. С другой стороны, левая часть выражении (1.22) при $x_i = 1$ равна $2 \cdot 2 \cdot \ldots \cdot 2 = 2^n$. Отсюда $m = 2^n$ — число слагаемых в полиноме Жегалкина (1.20). А так как коэффициенты $a_{i_1,i_2\ldots,i_k}$ слагаемых в (1.20) могут принимать два значения: 0 или 1, то количество полиномов Жегалкина определяется как $2^m = 2^{2^n}$ (правило прямого произведения п. 2.2). □

Теорема 1.4.1 (Жегалкина). *Любая булева функция $f(x_1, x_2, \ldots, x_n)$ из $P_2$ может быть представлена единственным образом своим полиномом Жегалкина $P_{\text{ж}}(x_1, x_2, \ldots, x_n)$* (1.20).

Доказательство. Используя таблицу истинности функции $f(x_1, x_2, \ldots, x_n)$ (табл. 1.7), составим формулу $F_3(x_1, x_2, \ldots$ $\ldots, x_n)$ по шаблону $F_1(x_1, x_2, \ldots, x_n)$ (1.15) теоремы о СДНФ (1.2.1), где вместо знака операции дизъюнкции ($\vee$) используется знак операции $\oplus$ — суммы по модулю 2:

$$F_3(x_1, x_2, \ldots, x_n) =$$

$$= f_0 K_0^1 \oplus f_1 K_1^1 \oplus \ldots \oplus f_{2^n-1} K_{2^n-1}^1 = \bigoplus_{i=0}^{2^n-1} f_i K_i^1, \quad (1.23)$$

где $f_i$ — значение $f$ на $i$-м наборе таблицы истинности; $K_i^1$ — конституента 1 $i$-го набора. Покажем совпадение исходной функции $f$ и формулы $F_3$. Для этого достаточно проверить их равенство на произвольном $\ell$-м наборе таблицы истинности функции $f$:

$$F_3|_\ell = f_0 K_0^1|_\ell \oplus f_1 K_1^1|_\ell \oplus \ldots \oplus f_{2^n-1} K_{2^n-1}^1|_\ell =$$

$$= \left| \text{Из утверждения (1.2.1)} \quad K_i^1|_\ell = \begin{cases} 0, & \text{если } i \neq \ell, \\ 1, & \text{если } i = \ell. \end{cases} \right| =$$

$$= f_0 \cdot 0 \oplus f_1 \cdot 0 \oplus \ldots \oplus f_\ell \cdot 1 \oplus \ldots \oplus f_{2^n-1} \cdot 0 = f_\ell.$$

Итак, $f$ и $F_3$ совпадают. Приведем формулу $F_3$ к полиному Жегалкина. Для этого достаточно избавиться от возможных операций отрицания в $K_i^1$ конституентах 1. Замена отрицания выполняется по аксиоме (II.4) $x \oplus 1 = \overline{x}$. Далее, согласно аксиоме

(III.1) $x \cdot (y \oplus z) = x \cdot y \ \oplus x \cdot z$, раскрываем скобки и, пользуясь аксиомами (II.3) $x \oplus x \ = x$ и (II.5) $x \oplus 0 = 0$, приводим подобные.

Единственность представления функций полиномами Жегалкина следует из утверждения 1.4.2, что количество таких полиномов от $n$ переменных совпадает с числом булевых функций того же числа аргументов и равно $2^{2^n}$. □

Замечание. СДНФ булевой функции $f$ не изменяется после замены в ней операций дизъюнкции $\vee$ на операции $\oplus$ — сумма по модулю 2, так как показано, что формулы $F_3$ и $F_1$ совпадают с исходной функцией $f$, где $F_1$ — СДНФ функции $f$.

Следствие. *Система булевых функций* $\sigma = \{1, \wedge, \oplus\}$ — *полная*.

Действительно, данный набор операций используется при построении полиномов Жегалкина булевых функций из $P_2$.

Пример. Построить полином Жегалкина функции $x \to y$.

Решение. Строить будем согласно теореме Жегалкина:

$$\begin{aligned}(x \to y)_{\text{ж}} &= (\overline{x}\,\overline{y} \vee \overline{x}\,y \vee x\,y)_{\text{СДНФ}} = \overline{x}\,\overline{y} \oplus \overline{x}\,y \oplus x\,y = \\ &= (x \oplus 1)(y \oplus 1) \oplus (x \oplus 1)\,y \oplus x\,y = \\ &= x\,y \oplus x \oplus y \ \oplus 1 \oplus x\,y \oplus y \oplus x\,y = \\ &= x\,y \oplus x \ \oplus 1.\end{aligned}$$

Утверждение 1.4.3. *Пусть* $P(x_1, x_2, \ldots, x_n)$ — *полином Жегалкина степени* $k \geqslant 3$. *С помощью замены переменных* (*их отождествления*) *данный полином можно привести к полиному степени* $k - 1$.

Доказательство. Без ограничения общности можно считать, что элементарное произведение $C = x_1 x_2 \ldots x_k$ из $k$ переменных является слагаемым полинома $P(x_1, x_2, \ldots, x_n)$. Отождествив в $P$ переменные $x_k = x_{k+1} = \ldots = x_n$, получим полином $Q(x_1, x_2, \ldots, x_k)$. Нетрудно заметить, что в полиноме $Q$ слагаемое $C = x_1 x_2 \ldots x_k$ останется *единственным* степени $k$. Полином $Q$ допускает следующие варианты по составу слагаемых $x_{m_1} x_{m_2} \cdot \ldots \cdot x_{m_{k-1}}$ степени $k - 1$.

1. Полином $Q$ содержит все слагаемые $x_{m_1} x_{m_2} \cdot \ldots \cdot x_{m_{k-1}}$ степени $k - 1$. Число таких слагаемых $C_k^{k-1} = k$. Среди слагаемых будут $A = x_1 x_2 \cdot \ldots \cdot x_{k-1}$ и $B = x_1 x_2 \cdot \ldots \cdot x_{k-2} x_k$, так как $k \geqslant 3$. Отождествив переменные $x_{k-1} = x_k$, получим равные слагаемые $A = B = C = x_1 x_2 \cdot \ldots \cdot x_{k-1}$. Их сумма $A \oplus B \oplus C = A$. Таким

образом, полученный полином $Q(x_1, x_2, \ldots, x_{k-1})$ будет иметь степень $k-1$.

2. Полином $Q$ содержит ровно $k-1$ слагаемое степени $k-1$: $x_{m_1}x_{m_2} \cdot \ldots \cdot x_{m_{k-1}}$. Одно слагаемое такой степени *отсутствует*. Пусть отсутствует слагаемое $A = x_1x_2 \cdot \ldots \cdot x_{k-1}$. Отождествив $x_1 = x_k$, получим $C = A$. Так как $A$ не входит в $Q$, то новое $C = x_1x_2 \cdot \ldots \cdot x_{k-1}$ степени $k-1$ войдет в $Q$. Таким образом, полученный полином $Q(x_1, x_2, \ldots, x_{k-1})$ будет иметь степень $k-1$.

3. В полиноме $Q$ *отсутствуют* два слагаемых степени $k-1$: $A = x_1x_2 \cdot \ldots \cdot x_{i-1}x_{i+1} \cdot \ldots \cdot x_k$ и $B = x_1x_2 \cdot \ldots \cdot x_{j-1}x_{j+1} \cdot \ldots \cdot x_k$, где $1 \leqslant i < j \leqslant k$. Отождествив $x_i = x_j$, получим $C = B$. Так как слагаемое $B$ отсутствует в полиноме $Q$, то новое $C = x_1x_2 \cdot \ldots \cdot x_{j-1}x_{j+1} \cdot \ldots \cdot x_k$ степени $k-1$ войдет в $Q$. Таким образом, и в последнем случае полином $Q(x_1, x_2, \ldots, x_{k-1})$ будет иметь степень $k-1$. □

### 1.4.2. Классы Поста булевых функций

В основу рекурсивного правила получения новых формул (п. 1.2.2) из элементарных булевых функций положены *операции замены переменных и суперпозиции*.

Класс (множество) булевых функций называется (*функционально*) *замкнутым*, если вместе с функциями из этого класса он содержит и все функции, полученные из них с помощью указанных операций.

Очевидно, что для доказательства замкнутости класса *достаточно* проверить его замкнутость лишь относительно элементарных операций суперпозиции.

#### Класс функций, сохраняющих константу 0

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ *сохраняет константу* 0, если $f(0, 0, \ldots, 0) = 0$. Класс функций, сохраняющих 0, обозначают через $T_0$.

Теорема 1.4.2. *Класс функций $T_0$ замкнут.*

Доказательство. Пусть $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ — функции, сохраняющие 0. Покажем, что и суперпозиция их $f_3(x_1, x_2, \ldots, x_n) = f_1(x_1, f_2(x_1, x_2, \ldots, x_n), \ldots, x_n)$ сохраняет 0. Действительно, $f_3(0, 0, \ldots, 0) = f_1(0, f_2(0, 0, \ldots, 0), \ldots, 0) = f_1(0, 0, \ldots, 0) = 0$. □

Замечание. Число функций $n$ переменных, сохраняющих 0, равно половине общего числа функций: $|T_0| = 2^{2^n}/2 = 2^{2^n-1}$.

### Класс функций, сохраняющих константу 1

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ *сохраняет константу* 1, если $f(1, 1, \ldots, 1) = 1$. Класс функций, сохраняющих 1, обозначают через $T_1$.

Теорема 1.4.3. *Класс функций $T_1$ замкнут.*

Доказательство. Пусть $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ — функции, сохраняющие 1. Покажем, что и суперпозиция их $f_3(x_1, x_2, \ldots, x_n) = f_1(x_1, f_2(x_1, x_2, \ldots, x_n), \ldots, x_n)$ сохраняет 1. Действительно, $f_3(1, 1, \ldots, 1) = f_1(1, f_2(1, 1, \ldots, 1), \ldots, 1) = f_1(1, 1, \ldots, 1) = 1$. □

Замечание. Число функций $n$ переменных, сохраняющих 1, равно половине общего числа функций: $|T_1| = 2^{2^n}/2 = 2^{2^n-1}$.

### Класс монотонных функций

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется *монотонной*, если для любых наборов значений $A = (a_1, a_2, \ldots, a_n)$ и $B = (b_1, b_2, \ldots, b_n)$ списка переменных функций таких, что $A > B$, выполняется $f(A) \geqslant f(B)$. Класс монотонных функций обозначают через $M$.

Теорема 1.4.4. *Класс монотонных функций $M$ замкнут.*

Доказательство. Пусть $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ — монотонные функции. Покажем, что и суперпозиция их $f_3(x_1, x_2, \ldots, x_n) = f_1(x_1, f_2(x_1, x_2, \ldots, x_n), \ldots, x_n)$ есть монотонная функция. Пусть $A > B$ — произвольные наборы. Обозначим наборы, соответствующие суперпозиции функций, через $\widetilde{A} = (a_1, f_2(a_1, a_2, \ldots, a_n), \ldots, a_n)$ и $\widetilde{B} = (b_1, f_2(b_1, b_2, \ldots, b_n), \ldots, b_n)$. Так как сравнение наборов $A > B$ выполняется по координатам и $f_2(A) \geqslant f_2(B)$, то $\widetilde{A} \geqslant \widetilde{B}$ и, значит, $f_1(\widetilde{A}) \geqslant f_1(\widetilde{B})$. Монотонность $f_3$ следует из соотношения $f_3(A) = f_1(\widetilde{A}) \geqslant f_1(\widetilde{B}) = f_3(B)$. □

### Класс линейных функций

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется *линейной*, если степень ее полинома Жегалкина не выше первой. Класс линейных функций обозначают через $L$.

Теорема 1.4.5. *Класс линейных функций $L$ замкнут.*

Доказательство. Пусть $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ — линейные функции. Покажем, что и суперпозиция их $f_3(x_1, x_2, \ldots, x_n) = f_1(x_1, f_2(x_1, x_2, \ldots, x_n), \ldots, x_n)$ есть линейная функция. Имеем $f_1 = a_0 \oplus a_1x_1 \oplus a_2x_2 \oplus \ldots \oplus a_nx_n$ и $f_2 = b_0 \oplus b_1x_1 \oplus b_2x_2 \oplus \ldots \oplus b_nx_n$. Очевидно, что $f_3 = a_0 \oplus a_1x_1 \oplus (b_0 \oplus b_1x_1 \oplus b_2x_2 \oplus \ldots \oplus b_nx_n)x_2 \oplus \ldots \oplus a_nx_n$ есть линейная функция. Степень суперпозиции линейных функций может только уменьшиться. □

Замечание. Несложно определить, что число линейных функций $n$ переменных равно $|L| = 2^{n+1}$.

### Класс самодвойственных функций

Определение. Функция $f(x_1, x_2, \ldots, x_n)$ называется *самодвойственной*, если для любого набора значений $A = (a_1, a_2, \ldots, a_n)$ списка переменных выполняется $f(A) = \overline{f}(\overline{A})$. Это означает, что самодвойственная функция на противоположных наборах принимает противоположные значения. Класс самодвойственных функций обозначают через $S$.

Замечание. Преобразование $\overline{f}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ обозначают как $f^*(x_1, x_2, \ldots, x_n)$. Тогда $f$ — самодвойственная, если $f(x_1, x_2, \ldots, x_n) = f^*(x_1, x_2, \ldots, x_n)$.

Теорема 1.4.6. *Класс функций $S$ замкнут.*

Доказательство. Пусть $f_1(x_1, x_2, \ldots, x_n)$, $f_2(x_1, x_2, \ldots, x_n)$ — самодвойственные функции. Покажем, что и суперпозиция их $f_3(x_1, x_2, \ldots, x_n) = f_1(x_1, f_2(x_1, x_2, \ldots, x_n), \ldots, x_n)$ есть самодвойственная функция. Пусть $A$ — произвольный набор. Тогда $f_3(A) = f_1(a_1, f_2(a_1, a_2, \ldots, a_n), \ldots, a_n) = \overline{f}_1(\overline{a}_1, \overline{f}_2(a_1, a_2, \ldots, a_n), \ldots, \overline{a}_n) = \overline{f}_1(\overline{a}_1, \overline{\overline{f}}_2(\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_n), \ldots, \overline{a}_n) = \overline{f}_1(\overline{a}_1, f_2(\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_n), \ldots, \overline{a}_n) = \overline{f}_3(\overline{A})$. Получили, что для произвольного набора $A$ выполняется $f_3(A) = \overline{f}_3(\overline{A})$. Значит, $f_3$ — самодвойственная функция. □

Замечание. Определим число |S| самодвойственных функций $n$ переменных. Самодвойственная функция определяется значениями на половине $2^n/2 = 2^{n-1}$ общего числа наборов $2^n$. На другой противоположной половине наборов ее значения устанавливаются как противоположные первой половине. Следовательно, $|S| = 2^{2^{n-1}}$.

Замечание. Классы булевых функций $T_0, T_1, M, L, S$ неполные и попарно различные. Так, можно привести примеры функций, не принадлежащих ни одному из выделенных классов, а для любой пары из этого множества классов можно указать функцию, которая одному из них принадлежит, а другому нет.

Пример. Приведем таблицу (табл. 1.10) перечня функций с оценкой распределения их по выделенным классам $T_0, T_1, M, L, S$. Таблица 1.10 заполняется непосредственно по таблице истинности (табл. 1.5) приведенного перечня функций. Для заполнения столбца $L$ линейности функций необходимо составить их полиномы Жегалкина.

Таблица 1.10

| Функции | $T_0$ | $T_1$ | $M$ | $L$ | $S$ |
|---|---|---|---|---|---|
| 0 | + | − | + | + | − |
| 1 | − | + | + | + | − |
| $x$ | + | + | + | + | + |
| $\neg x$ | − | − | − | + | + |
| $x \cdot y$ | + | + | + | − | − |
| $x \vee y$ | + | + | + | − | − |
| $x \nrightarrow y$ | + | − | − | − | − |
| $x \oplus y$ | + | − | − | + | − |
| $x \to y$ | − | + | − | − | − |
| $x \sim y$ | − | + | − | + | − |
| $x \downarrow y$ | − | − | − | − | − |
| $x \mid y$ | − | − | − | − | − |

### 1.4.3. Теоремы Поста о функциональной полноте

Рассматриваемые здесь теоремы доказаны Э. Постом в 1921 г. Полученные результаты функциональной полноты булевых функций заложили прочную основу развития современных средств вычислительной техники, алгоритмизации и т. п.

Теорема 1.4.7 (ослабленная, Э. Пост, 1921). *Система булевых функций $\sigma$, содержащая константу* 0 *и константу* 1, *полна тогда и только тогда, когда система $\sigma$ содержит хотя бы одну немонотонную и хотя бы одну нелинейную функции.*

Замечания. 1. В теореме речь идет о системе функций $\sigma = \{f_1, f_2, f_3, f_4\}$, где

$f_1$ — константа 0,
$f_2$ — константа 1,
$f_3$ — немонотонная,
$f_4$ — нелинейная.

Отметим, что множество $\sigma$ объединяет функции со свойствами. Поэтому количество функций в $\sigma$ зависит от наличия у них указанных свойств. Так, $\sigma$ может состоять и из одной функции, если у этой функции будут все необходимые свойства.

2. Отметим, что $f_3 \neq f_1$, $f_3 \neq f_2$, $f_4 \neq f_1$ и $f_4 \neq f_2$, поскольку $f_1$ и $f_2$ — линейные и монотонные функции. Это означает, что $\sigma$ содержит не менее трех функций, две из которых есть $f_1$ и $f_2$.

3. Теорема называется *ослабленной*, так как система $\sigma$ включает фиксированые функции — это константа 0 и константа 1.

Доказательство. *Необходимость.* Имеем $\sigma$ — полный набор и $f_1$, $f_2 \in \sigma$. Необходимо показать, что и $f_3$, $f_4 \in \sigma$. Если предположить, что $f_3 \notin \sigma$, тогда множество $\sigma$ содержит только монотонные функции, так как $f_1$ и $f_2$ — монотонные. Класс монотонных функций замкнут, значит, в условиях предположения в рамках $\sigma$ нельзя получить ни одной немонотонной функции. Это противоречит условию, что $\sigma$ есть полный набор. Значит, предположение неверно и $f_3 \in \sigma$. Подобным образом доказывается, что и $f_4 \in \sigma$.

*Достаточность.* Имеем систему функций $\sigma = \{f_1, f_2, f_3, f_4\}$. Доказать, что $\sigma$ — полный набор. Для доказательства достаточно показать, что из набора $\sigma$ можно получить функции множества $\{\overline{x}, x\,y, x \vee y\}$. Поскольку система $\{\overline{x}, x\,y, x \vee y\}$ есть полный набор, то и набор функций $\sigma$ будет полным. *Важно, что доказательство имеет конструктивный характер, т. е. выполняется явное построение функций множества $\{\overline{x}, x\,y, x \vee y\}$ из набора $\sigma$.* Это позволит использовать предложенную технику в условиях конкретного набора функций $\sigma$.

*1. Получение отрицания $\overline{x}$ из $f_1$, $f_2$ и $f_3$.* По условию $f_3$ — немонотонная, значит, существуют наборы $A$ и $B$ списка переменных такие, что $A > B$ и $f_3(A) < f_3(B)$ — нарушается условие монотонности. Покажем, что существуют и соседние наборы $A > B$, для которых $f_3(A) < f_3(B)$. Действительно, свяжем наборы $A$ и $B$ цепочкой соседних наборов

$$A = \alpha_1 > \alpha_2 > \alpha_3 > \ldots > \alpha_r = B, \tag{1.24}$$

где $\alpha_i$ — соседние наборы. Например, если $A = (11111)$ и $B = (00001)$, то цепочка соседних наборов может иметь такой вид:

$$A = (11111) > (01111) > (00111) > (00011) > (00001) = B.$$

Среди цепочки соседних наборов (1.24) существует пара соседних $\alpha_\ell > \alpha_{\ell+1}$, для которых $f_3(\alpha_\ell) < f_3(\alpha_{\ell+1})$. В противном случае можно составить цепочку

$$f_3(A) = f_3(\alpha_1) \geqslant f_3(\alpha_2) \geqslant f_3(\alpha_3) \geqslant \ldots \geqslant f_3(\alpha_r) = f_3(B).$$

Отсюда $f_3(A) \geqslant f_3(B)$, что противоречит условию $f_3(A) < f_3(B)$. Итак, далее полагаем, что $A$ и $B$ — соседние наборы:

$$A = (a_1, a_2, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n),$$
$$B = (a_1, a_2, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n).$$

Рассмотрим функцию одного аргумента $\varphi(x)$, полученную заменой переменных:

$$\varphi(x) = f_3(a_1, a_2, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_n). \tag{1.25}$$

Для нее выполняется условие $\varphi(0) = f_3(B)$ и $\varphi(1) = f_3(A)$ или $\varphi(0) = f_3(B) > f_3(A) = \varphi(1)$. Отсюда $\varphi(0) > \varphi(1)$ или $\varphi(0) = 1$ и $\varphi(1) = 0$. Значит, $\varphi(x) = \overline{x}$.

*2. Получение конъюнкции $xy$ и дизъюнкции $x \vee y$ из $f_1$, $f_2$, $f_4$ и $\overline{x}$.* По условию $f_4$ — нелинейная. Можно считать, что степень функции $f_4$ равна двум (см. утверждение 1.4.3) и

$$f_4(x) = xy \oplus \alpha x \oplus \beta y \oplus \gamma. \tag{1.26}$$

Избавимся от линейной части $\alpha x \oplus \beta y$. Для этого рассмотрим суперпозицию функций

$$\begin{aligned}\psi(x, y) &= f_4(x \oplus \beta, y \oplus \alpha) = \\ &= (x \oplus \beta)(y \oplus \alpha) \oplus \alpha(x \oplus \beta) \oplus \beta(y \oplus \alpha) \oplus \gamma = \\ &= xy \oplus \alpha x \oplus \beta y \oplus \alpha\beta \oplus \alpha x \oplus \beta y \oplus \alpha\beta \oplus \alpha\beta \oplus \gamma = \\ &= xy \oplus \alpha\beta \oplus \gamma.\end{aligned}$$

Таким образом,

$$\psi(x, y) = \begin{cases} xy, & \text{если } \alpha\beta \oplus \gamma = 0, \\ xy \oplus 1, & \text{если } \alpha\beta \oplus \gamma = 1. \end{cases}$$

Для второго случая выполним преобразования

$$\psi(x, y) = \begin{cases} xy, \\ xy \oplus 1, \end{cases} = \begin{cases} xy, \\ \overline{xy}, \end{cases} = \begin{cases} xy, \\ \overline{x} \vee \overline{y}. \end{cases}$$

Отсюда $\psi(x,y) = xy$ или $\psi(x,y) = \overline{x} \vee \overline{y}$. Последнее выражение приведем к виду $\psi(\overline{x},\overline{y}) = x \vee y$. Итак, из нелинейной функции $f_4$ можно получить $xy$ или $x \vee y$.

В том случае, если получили конъюнкцию $xy = \psi(x,y)$, то дизъюнкция определяется как суперпозиция фунций $\overline{x}$ и найденной $xy$:

$$x \vee y = \overline{\overline{x}\,\overline{y}} = \overline{\psi}(\overline{x},\overline{y}). \tag{1.27}$$

Если же получили дизъюнкцию $x \vee y = \psi(\overline{x},\overline{y})$, то конъюнкция есть суперпозиция $\overline{x}$ и найденной $x \vee y$:

$$xy = \overline{\overline{x} \vee \overline{y}} = \overline{\psi}(x,y). \tag{1.28}$$

□

Пример. Рассмотрим набор булевых функций $\sigma = \{0, 1, \to\}$. Импликация $\to$ является немонотонной и нелинейной функцией (табл. 1.10). Значит, набор $\sigma$ — полный. Получим, согласно теореме Поста 1.4.3, функции $\neg$, $\wedge$, и $\vee$.

*1. Получение $\overline{x}$ из* $\{0, 1, f_3 = x \to y\}$. По таблице истинности функции $x \to y$ (табл. 1.5) определяем соседние наборы $A = (1,0)$ и $B = (0,0)$, для которых $f_3(A) < f_3(B)$ т. е. $0 = 1 \to 0 < 0 \to \to 0 = 1$ — нарушается условие монотонности. Тогда из (1.25) находим отрицание $\overline{x} = f_3(x,0) = x \to 0$.

*2. Получение $xy$ и $x \vee y$ из* $\{0, 1, f_4 = x \to y$ *и* $\overline{x}\}$. Как в теореме Поста 1.4.3 составим полином Жегалкина $f_4(x,y) = x \to y = xy \oplus x \oplus 1$. С помощью подстановки $\overline{y} = y \oplus 1$ вместо $y$ избавимся от линейной части $x$ в последнем выражении $\psi(x,y) = f_4(x,\overline{y}) = x \to \overline{y} = x(y \oplus 1) \oplus x \oplus 1 = xy \oplus 1 = \overline{xy} = \overline{x} \vee \overline{y}$. Подстановка $\overline{x}$ и $\overline{y}$ в $\psi(x,y)$ дает $x \vee y = \overline{\overline{x}} \vee \overline{\overline{y}} = \overline{x} \to y = \psi(\overline{x},\overline{y})$. Итак, дизъюнкция $x \vee y = (x \to 0) \to y$.

Конъюнкцию находим из (1.28) $xy = \overline{\overline{x} \vee \overline{y}} = \overline{\psi}(x,y) = \overline{x \to \overline{y}} = (x \to \overline{y}) \to 0 = (x \to (y \to 0)) \to 0$. Сведем вычисления в таблицу результатов:

$$\begin{aligned} \overline{x} &= x \to 0, \\ x \vee y &= (x \to 0) \to y, \\ xy &= (x \to (y \to 0)) \to 0. \end{aligned} \tag{1.29}$$

Замечание. Выражения (1.29) позволяют заключить, что набор из двух функций $\sigma = \{0, \to\}$ является полным.

Теорема 1.4.8 (основная, Э. Пост, 1921). *Система булевых функций $\sigma$ полна тогда и только тогда, когда система $\sigma$ содержит*

*хотя бы одну* $f_1$ *не сохраняющую* 0,
*хотя бы одну* $f_2$ *не сохраняющую* 1,
*хотя бы одну* $f_3$ *немонотонную*,
*хотя бы одну* $f_4$ *нелинейную и*
*хотя бы одну* $f_5$ *несамодвойственную*
*функции*.

Доказательство. В теореме речь идет о системе функций $\sigma = \{f_1, f_2, f_3, f_4, f_5\}$. Отметим, что множество $\sigma$ объединяет функции со свойствами. Поэтому количество функций в $\sigma$ зависит от наличия у них указанных свойств. Так, $\sigma$ может состоять и из одной функции, если у этой функции будут все необходимые свойства.

*Доказательство необходимости* повторяет доказательство этого условия в ослабленной теореме Поста 1.4.7, где можно с ним и ознакомиться.

*Доказательство достаточности*. Имеем систему функций $\sigma = \{f_1, f_2, f_3, f_4, f_5\}$. Доказать, что $\sigma$ — полный набор. Для этого сведем условие теоремы к условию ослабленной теоремы Поста 1.4.7. Для выполнения условий ослабленной теоремы не достает функций константа 0 и константа 1, которые получим из предложенного набора функций $\sigma = \{f_1, f_2, f_3, f_4, f_5\}$. Сравнение условий ослабленной теоремы и настоящей позволяет заключить, что функции константу 0 и константу 1 следует получать из $f_1$, $f_2$, и $f_5$. *Получение указанных функций носит конструктивный характер*, *т. е. выполняется их явное построение*.

*1. Получение константы* 1 *или* $\overline{x}$ *из* $f_1$, *не сохраняющей* 0. Рассмотрим функцию $\varphi(x) = f_1(x, x, \ldots, x)$. Поведение функции $f_1$ на единичном наборе $(1, 1, \ldots, 1)$ не оговорено. Поэтому возможны следующие два случая: a) и b).

a). $\varphi(0) = f_1((0, 0, \ldots, 0)) = 1$, $\varphi(1) = f_1((1, 1, \ldots, 1)) = 1$. | Здесь $\varphi(x) = 1$ — константа 1.

b). $\varphi(0) = f_1((0, 0, \ldots, 0)) = 1$, $\varphi(1) = f_1((1, 1, \ldots, 1)) = 0$. | Здесь $\varphi(x) = \overline{x}$ — отрицание.

*2. Получение константы* 0 *или* $\overline{x}$ *из* $f_2$, *не сохраняющей* 1. Рассмотрим функцию $\psi(x) = f_2(x, x, \ldots, x)$. Поведение функции $f_2$ на нулевом наборе $(0, 0, \ldots, 0)$ не оговорено. Поэтому возможны следующие два случая: c) и d).

c). $\psi(0) = f_2((0, 0, \ldots, 0)) = 0$, $\psi(1) = f_2((1, 1, \ldots, 1)) = 0$. | Здесь $\psi(x) = 0$ — константа 0.

d). $\psi(0) = f_2((0, 0, \ldots, 0)) = 1$, $\psi(1) = f_2((1, 1, \ldots, 1)) = 0$. | Здесь $\psi(x) = \overline{x}$ — отрицание.

Комбинации случаев a), b), c) и d) дают четыре варианта.

a)+c). Имеем $\varphi(x) = 1$ и $\psi(x) = 0$. Условия ослабленной теоремы выполнены, *теорема доказана*.

a)+d). Имеем $\varphi(x) = 1$ и $\psi(x) = \overline{x}$, суперпозиция которых дает $0 = \overline{1}$. Располагаем 0 и 1, *теорема доказана*.

b)+c). Имеем $\varphi(x) = \overline{x}$ и $\psi(x) = 0$, суперпозиция которых дает $1 = \overline{0}$. Располагаем 0 и 1, *теорема доказана*.

b)+d). Имеем $\varphi(x) = \overline{x}$ и $\psi(x) = \overline{x}$. В этом случае из $f_1$ и $f_2$ смогли получить лишь отрицание $\overline{x}$. Однако теперь для выполнения условий ослабленной теоремы достаточно получить одну из двух констант 0 или 1. Другую определим как отрицание найденной. Тогда и в этом случае будем располагать 0 и 1, *теорема доказана*.

*Получение константы* 0 *или* 1 *из* $f_5$ *несамодвойственной функции*. Так как $f_5$ — несамодвойственная, то существует набор $A = (a_1, a_2, \ldots, a_n)$, на котором нарушается условие самодвойственности, т. е. выполняется условие $f_5(A) = f_5(\overline{A})$. Рассмотрим суперпозицию функций

$$\xi(x) = f_5(x_1, x_2, \ldots, x_n) = f_5(x, \overline{x}, \ldots, x). \quad \left| \begin{array}{l} \overline{x} \to x_i, \text{ если } a_i = 0, \\ x \to x_i, \text{ если } a_i = 1. \end{array} \right. \tag{1.30}$$

Тогда $\xi(1) = f_5(A)$ и $\xi(0) = f_5(\overline{A})$ и так как выполняется $f_5(A) = f_5(\overline{A})$, то $\xi(1) = \xi(0)$. Следовательно, $\xi(x)$ — это константа либо 0, либо 1. □

Замечание. Результаты теорем Поста дают важные основания к систематическому изучению и подбору полных систем булевых функций.

Пример. Набор $\sigma = \{x \downarrow y\}$ из одной функции (стрелка Пирса) является полным. Данная функция обладает всеми указанными свойствами (табл.1.10), обозначенными в теореме Поста. Получим из этого набора $\sigma$ функции $\{0, 1, \overline{x}, xy, x \vee y\}$.

*Будем следовать основной теореме Поста*. Проверим поведение $x \downarrow y$ на нулевом и единичном наборах. Так, $0 \downarrow 0 = 1$ и $1 \downarrow 1 = 0$. Это значит, что мы находимся в условиях варианта b)+d) основной теоремы Поста, что позволяет получить отрицание $\overline{x} = x \downarrow x$. Одну из констант необходимо получить из свойства нарушения самодвойственности функции $x \downarrow y$.

*Получение констант*. Для набора $A = (0, 1)$ нарушается условие самодвойственности. Из (1.30) имеем $\overline{x} \downarrow x = 0$ или $(x \downarrow x) \downarrow x = 0$. Отсюда $1 = \overline{0} = 0 \downarrow 0 = ((x \downarrow x) \downarrow x) \downarrow ((x \downarrow x) \downarrow x)$.

*Получение конъюнкции и дизъюнкции*. Правило получения из нелинейной функции указанных функций рассматривается в ослабленной теореме Поста и предусматривает составление полинома Жегалкина (1.26). Составим $(x \downarrow y)_{\text{СДНФ}} = \overline{x}\,\overline{y}$. Отсюда конъюнкция $xy = \overline{x} \downarrow \overline{y} = (x \downarrow x) \downarrow (y \downarrow y)$. Вернемся к $(x \downarrow y)_{\text{СДНФ}} = \overline{x}\,\overline{y}$. Отсюда дизъюнкция $x \vee y = \overline{\overline{x}\,\overline{y}} = \overline{x \downarrow y} = (x \downarrow y) \downarrow (x \downarrow y)$. Сведем вычисления в таблицу результатов:

$$\begin{aligned} 0 &= (x \downarrow x) \downarrow x, \\ 1 &= ((x \downarrow x) \downarrow x) \downarrow ((x \downarrow x) \downarrow x), \\ \overline{x} &= x \downarrow x, \\ xy &= (x \downarrow x) \downarrow (y \downarrow y), \\ x \vee y &= (x \downarrow y) \downarrow (x \downarrow y). \end{aligned} \tag{1.31}$$

Замечание. Функция называется *шефферовой*, если она одна составляет полную систему (базис в $P_2$). Примеры таких функций двух переменных можно найти в таблице 1.10.

## § 1.5. Исчисление высказываний

*Исчисление высказываний* — это раздел математической логики, который посвящен изучению связей между *высказываниями*. Указанные связи определяются исключительно тем, каким образом одни высказывания строятся из других. Другое название исчисления высказываний — это логика высказываний.

Исчисление высказываний рассматривается как формальная аксиоматическая теория. Формальному введению предварим объяснение отдельных вопросов перехода от булевой алгебры к исчислению высказываний.

### 1.5.1. Основные понятия

*Высказывание* — это предложение (или набор слов, или утверждение, или суждение), о котором имеет смысл говорить, что оно истинно либо ложно.

*Простым* называется высказывание, которое нельзя разбить на части, чтобы каждая из них являлась высказыванием.

Примеры. (1) Тайфун — это грозное синоптическое явление. (2) Небольшие осадки — благо. (3) Большие осадки — бедствие.

*Сложным* называется высказывание, состоящее из нескольких простых, соединенных логическими связками: *и*, *или*, *если ...*, *то*, *нет* (*не*) и подобные им.

Примеры. (1) С приходом тайфуна может выпасть до годовой нормы осадков. (2) Если траектория перемещения тайфуна повторит траекторию кривой его прогноза, то возможно избежать его разрушительных последствий.

Содержательное знакомство с формулами логики высказываний можно найти в разделе смысловой интерпретации булевых функций (п. 1.2.5). Таблицы истинности булевых функций дают основание заключить, что в рамках раздела булевой алгебры допустимо решение многих задач, касающихся формул логики высказываний. Существуют более сложные задачи исчисления высказываний, которые не могут быть решены в рамках булевой алгебры. В этом отношении важной причиной является и огромная трудоемость табличного решения задач большой размерности.

### 1.5.2. Задачи исчисления высказываний

Основными задачами исчисления высказываний являются:
1) формализация записи простых и сложных высказываний;
2) установление истинности высказываний.

#### Формализация записи простых и сложных высказываний

1. Для обозначения высказываний используют прописные буквы латинского алфавита: $A, B, C_1, C_2, X, Y_1, Y_2, Y_3, Z$, которые называют *пропозициональными* переменными.

2. Вводят пропозициональные связки (символы, операции): $\neg, \wedge, \vee, \supset, \sim$, эквивалентные логическим связкам: не ($\neg$), и ($\wedge$), или ($\vee$), если ... то ($\supset$), эквивалентно (равносильно) ($\sim$).

3. Вводят понятие *формулы*. Формулами объявляются простые высказывания. Другие формулы строятся посредством пропозициональных связок и ранее построенных формул. Так, если $A$ и $B$ — это формулы, то формулами считаются выражения (сложные высказывания): $\neg A$, $A \wedge B$, $A \vee B$, $A \supset B$, $A \sim B$.

4. Вводят скобки «()» для установления порядка действия пропозициональных связок. *Например*, $(A \supset (B \supset A)) \vee (\neg C)$.

5. Назначают приоритет операциям: $\neg, \wedge, \vee, \supset, \sim$ для определения последовательности вычислений в бесскобочных формулах. *Например*, последовательность вычислений в формуле $A \supset B \supset A \ \vee \neg C$ соответствует следующей ее скобочной записи $((A \supset B) \supset (A \vee (\neg C)))$.

Определение. Формула называется *тождественно истинной*, если она истинна всегда независимо от истинности или лжи составляющих ее формул. Тождественно истинные формулы (ТИФ) еще называют *общезначимыми* и *тавтологиями*.

**Установление истинности высказываний**

Решение задачи установления истинности высказываний можно проводить двумя способами:

1) содержательно на основе таблиц истинности булевой алгебры;
2) формально с помощью аксиоматико-дедуктивного метода.

Замечание. В исчислении высказываний при решении задачи установления истинности предпочтение отдается дедуктивному методу.

### 1.5.3. Соотношение булевой алгебры и исчисления высказываний. Содержательный аспект

Определение формулы как объекта исчисления высказываний ничем не отличается от формул булевой алгебры. Поэтому вполне допустимо проводить анализ подобных формул средствами булевой алгебры, к чему часто и прибегают.

Исчисление высказываний как формальная система предлагает свои средства анализа формул, направленные на решение основной ее задачи — установление истинности формул. С этой целью в систему включаются средства дедуктивного анализа логических выражений, когда на основе ряда формул, истинность которых установлена, возможно получение новых истинностных формул.

Если в *булевой алгебре* отношение равенства ($=$) выполняло роль перехода к эквивалентным формулам в процессе их преобразования, то в *исчислении высказываний* понятие равенства как отношения между формулами удалено за ненадобностью. Здесь формулы оцениваются лишь на их *истинность*. В исчислении высказываний в качестве *основных операций* вводят импликацию ($\supset$) и эквивалентность ($\sim$).

Анализ формул булевой алгебры возможно проводить средствами исчисления высказываний. Для этого необходимо выполнить замену символа равенства ($=$) на символ эквивалентности ($\sim$). Теперь эквивалентные формулы булевой алгебры становятся тождественно истинными формулами в исчислении высказываний, верно и обратно. Так, система аксиом булевой алгебры из

системы равенств преобразуются в систему тождественно истинных формул.

### 1.5.4. Содержательный аспект тождественно истинных формул

Основными объектами преобразования в исчислении высказываний являются тождественно истинные формулы (с учетом контекста решения задачи). С этой точки зрения необходимо иметь какие-либо правила, закономерности, свойства распознавания тождественно истинных формул. Приведем наиболее важные правила содержательного аспекта распознавания тождественно истинных формул.

1. Тождественно истинные формулы состоят из эквивалентных равносильных частей, соединенных операцией эквивалентности ($\sim$). Равносильными называются формулы (высказывания), принимающие одно и то же значение при одинаковых значениях переменных. *Например*, $x \vee y \ \sim y \vee x$.
2. Тождественно истинные формулы соответствуют схемам правильного логического вывода.
3. Тождественно истинные формулы соответствуют законам формальной логики.

#### Схемы правильного логического вывода

Перечень рассмотренных схем правильного логического вывода, конечно же, является не всеобъемлющим. Приводятся схемы, наиболее часто используемые в математических рассуждениях, доказательствах. Схемы являются тождественно истинными формулами. Однако каждая из них допускает использование ее как схемы получения новых формул, а значит, и новых знаний. Истинность таких формул, как правило, определяется истинностью посылок соответствующих схем вывода.

Будем полагать, что $x, y, z$ — это произвольные формулы (высказывания). Каждое правило будем записывать в виде тождественно истинной формулы и в формате схемы вывода с указанием посылок и заключения.

1. *Правило заключения по импликации*, или *modus ponens* (*mp*), или правило отделения:

$$(x \wedge (x \supset y)) \supset y \quad \text{или} \quad \frac{x, x \supset y}{y}. \tag{1.32}$$

Схема (1.32) понимается как правило перехода от двух формул вида $x$ и $x \supset y$ к одной формуле $y$. В *выводе* этого правила $x$ и $x \supset y$ являются *посылками*, $y$ — *заключением*.

*Интерпретация*. Если верно условие $x$ и известно, что верна теорема $x \supset y$ (из $x$ следует $y$), то верно и условие $y$. Отметим, что заключение правила modus ponens истинно настолько, насколько истинны посылки.

2. *Правило двойственности*, или *modus tollens*, или правило отрицания:

$$((x \supset y) \wedge \neg y\ ) \supset \neg x \quad \text{или} \quad \frac{x \supset y, \neg y}{\neg x}. \tag{1.33}$$

В *выводе* этого правила $x \supset y$ и $\neg y$ являются *посылками*, $\neg x$ — *заключением*. *Интерпретация*. Если верна теорема $x \supset y$ (из $x$ следует $y$) и известно, что условие $y$ не выполняется (верно условие $\neg y$), то тогда и условие $x$ не выполняется (верно $\neg x$).

Данная схема вывода лежит в основе доказательства от противного. Допустим, необходимо доказать теорему $x \supset y$. Для ее доказательства предполагают, что она неверна. Это означает, что выполняется условие $\neg y$, противоположное заключению теоремы. Далее из условия $\neg y$ получают, что выполняется $\neg x$. А так как при доказательстве теоремы $x \supset y$ условие $x$ предполагается истинным, то делают вывод, что полученное заключение $\neg x$ противоречит условию теоремы и, значит, предположение $\neg y$ неверное. Откуда заключают истинность теоремы $x \supset y$.

3. *Правило транзитивности*, или цепное правило, или *закон силлогизма* (*syllogismos*):

$$((x \supset y) \wedge (y \supset z)) \supset (x \supset z) \quad \text{или} \quad \frac{x \supset y,\ y \supset z}{x \supset z}. \tag{1.34}$$

В *выводе* этого правила $x \supset y$ и $y \supset z$ являются *посылками*, $x \supset z$ — *заключением*. *Интерпретация*. Если из условия $x$ следует $y$, а из условия $y$ следует $z$, то из условия $x$ следует $z$.

4. *Правило приведения к абсурду* или *reductio ad absurdum*:

$$((\neg x \supset y) \wedge (\neg x \supset \neg y)) \supset x \qquad \text{или} \qquad \frac{\neg x \supset y,\ \neg x \supset \neg y}{x}. \tag{1.35}$$

В *выводе* этого правила формулы $\neg x \supset y$ и $\neg x \supset \neg y$ являются *посылками*, $x$ — *заключением*. *Интерпретация*. Если из условия $\neg x$ следует $y$ и противоположное ему условие $\neg y$, то посылка $\neg x$ является неверной. Значит, выполняется условие $x$.

5. *Правило истины из чего угодно* или *verum ex quadlibet*:

$$x\ \supset (y \supset x) \quad \text{или} \quad \frac{x}{y \supset x}. \tag{1.36}$$

В *выводе* этого правила $x$ является *посылкой*, $y \supset x$ — *заключением*. *Интерпретация*. Если условие $x$ выполняется,

то верна любая теорема $y \supset x$ вне зависимости от условия $y$. Таким образом, истинное условие $x$ можно доказать *из чего угодно* $y$.

6. *Правило из лжи что угодно* или *ex falco quadlibet*:

$$\neg x \ \supset (x \supset y) \quad \text{или} \quad \frac{\neg x}{x \supset y}. \tag{1.37}$$

В *выводе* этого правила формула $\neg x$ является *посылкой*, $x \supset y$ — *заключением*. *Интерпретация*. Если условие $x$ неверное (ложное), то доказать можно любую теорему вида $x \supset y$ вне зависимости от условия $y$. Таким образом, из неверной посылки $x$ можно доказать *что угодно* $y$.

**Законы формальной логики**

При любом выборе формулы $x$ тождественно истинными являются следующие формулы.

1. $x \supset x$ — *закон тождества*. Значение истинности высказывания не должно изменяться в течение всего времени до нового отрицания.
2. $\neg(x \wedge \overline{x})$ — *закон противоречия* (*абсурда*). Не верно, что одновременно $x$ и $\neg x$ — абсурд.
3. $x \vee \overline{x}$ — *закон исключенного третьего*. Истинно либо $x$, либо $\neg x$, другого не бывает.
4. $\neg\neg x \supset x$ — *закон двойного отрицания*. Двойное отрицание $x$ тождественно исходному.

### 1.5.5. Формальное введение в исчисление высказываний

Определение исчисления высказываний как формальной системы требует удовлетворения определенных положений (правил) с ее стороны. Перечень и содержание такого рода положений приводится ниже.

1. Определяются все формальные объекты исчисления высказываний и правила их образования (построения).
2. Из формального определения исключается все, что нельзя выразить на языке формального описания.
3. Вводится система аксиом, представляющая конечный список первоначальных формул.
4. Вводятся основные операции (правила вывода), только в соответствии с которыми можно преобразовывать формулы.
5. Вводятся формальные понятия доказуемости и выводимости.
6. Рассматриваются общие (внешние) оценки исчисления высказываний — непротиворечивость, полнота, независимость системы аксиом, разрешимость системы в целом.

### (1) Формальные объекты и правила их образования

Формальные объекты делятся на три категории.

I. *Алфавит* (*символы*):

1. Пропозициональные буквы: $A, B, X_1, X_2, X_3, \ldots$.
2. Пропозициональные связки (операции): $\neg, \wedge, \vee, \supset, \sim$.
3. Скобки: ().

II. *Формулы*. Формулой называется конечная последовательность вхождения *символов алфавита*. Выделяют правильно и неправильно построенные формулы. *Например*, $A \wedge B$, $A \supset B \sim \neg D$ — формулы.

III. *Правильное* (*рекурсивное*) *построение формул*:

1. Любая пропозициональная буква является формулой.
2. Если $A$ и $B$ — формулы, то формулами являются: $(\neg A)$, $(\neg B)$, $(A \wedge B)$, $(A \vee B)$, $(A \supset B)$, $(A \sim B)$.
3. Никаких других формул не существует.
4. Приоритет операций: $\neg$, $\wedge$, $\vee$, $\supset$, $\sim$.

### (2) Исключение из системы невыразимых понятий

1. Из системы исключаются прежде всего содержание и смысл, остаются лишь последовательность букв, цифр, знаков.

2. Исключаются понятия *истины* и *лжи*. Формальная система помогает сохранить в процессе доказательства и вывода первоначальный *смысл*, не уменьшая его и не увеличивая.

3. Исключается знак $=$, который означает (a) одинаковость объектов, (b) одинаковость результатов действия над некоторыми объектами. Потребность в этом знаке равенства ($=$) отпадает, так как содержание включается непосредственно в объекты, например, в $A$ и $B$, и во время выполнения над ними операций, например, $A \vee B$, нового содержания и смысла в полученные формулы не привносится.

### (3) Система аксиом

Системой аксиом называется конечная совокупность начальных формул и каким-то образом образующих всю совокупность формул исчисления высказываний. С точки зрения содержательного аспекта аксиомы — это совокупность тождественно истинных формул, которые характеризуют все множество тождественно истинных формул. Существует более десятка систем аксиом. Приведем некоторые из них, которые наиболее известны и которыми будем пользоваться.

### Система аксиом расширенного исчисления высказываний

В данной системе используется максимальное число операций — пять: $\neg$, $\cdot$, $\vee$, $\supset$, $\sim$. Данная система получена из системы аксиом булевой алгебры путем замены символа равенства ($=$) на символ эквивалентность ($\sim$), где $x, y, z$ — формулы.

1. $x \cdot y \sim y \cdot x$.
2. $(x \cdot y) \cdot z \sim x \cdot (y \cdot z)$.
3. $x \cdot x \sim x$.
4. $x \vee y \sim y \vee x$.
5. $(x \vee y) \vee z \sim x \vee (y \vee z)$.
6. $y \vee y \sim y$.
7. $x \cdot (y \vee z) \sim x \cdot y \ \vee x \cdot z$.
8. $x \vee (y \cdot z) \sim (x \vee y) \cdot (x \vee z)$.
9. $x \vee x \cdot y \ \sim x$.
10. $x \cdot (x \vee y) \sim x$.
11. $\neg(x \cdot y) \sim \neg x \vee \neg y$.
12. $\neg(x \vee y) \sim \neg x \cdot \neg y$.
13. $\neg\neg x \sim x$.
14. $x \supset y \ \sim \neg x \vee y$.
15. $x \sim y \ \sim x \cdot y \vee \neg x \cdot \neg y$.
16. $x \vee (y \cdot \neg y) \sim x$.
17. $x \vee (y \vee \neg y) \sim y \vee \neg y$.
18. $x \cdot \neg x \ \cdot y \ \vee z \ \sim z$ .
19. $x \vee \neg x$.
20. $\neg(x \cdot \neg x)$.
21. $\dfrac{x, x \supset y}{y}$ — modus ponens.

### Система аксиом исчисления Новикова

Операции: $\neg$, $\wedge$, $\vee$, $\supset$.

Аксиомы:
1. $A \supset (B \supset A)$.
2. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$.
3. $A \wedge B \supset A$.
4. $A \wedge B \supset B$.
5. $(A \supset B) \supset ((A \supset C) \supset (A \supset B \wedge C))$.
6. $A \supset A \vee B$.
7. $B \supset A \vee B$.
8. $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$.
9. $(A \supset B) \supset (\neg B \supset \neg A)$.
10. $A \supset \neg\neg A$.
11. $\neg\neg A \supset A$.
12. $\dfrac{A, A \supset B}{B}$ — modus ponens.

### Система аксиом исчисления Клини

Операции: $\neg$, $\wedge$, $\vee$, $\supset$.

Аксиомы:
1. $A \supset (B \supset A)$.
2. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$.
3. $A \supset (B \supset A \wedge B)$.

4. $A \wedge B \supset A$.
5. $A \wedge B \supset B$.
6. $A \supset A \vee B$.
7. $B \supset A \vee B$.
8. $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$.
9. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$.
10. $\neg\neg A \supset A$.
11. $A \supset \neg\neg A$.
12. $\dfrac{A, A \supset B}{B}$ — modus ponens.

**Система аксиом исчисления Фреге**

Операции: $\neg$, $\supset$.

Аксиомы:
1. $A \supset (B \supset A)$.
2. $(A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$.
3. $(A \supset (B \supset C)) \supset (B \supset (A \supset C))$.
4. $(A \supset B) \supset (\neg B \supset \neg A)$.
5. $A \supset \neg\neg A$.
6. $\neg\neg A \supset A$.
7. $\dfrac{A, A \supset B}{B}$ — modus ponens.

Добавляются в неявном виде.

8. $A \wedge B \ \sim \neg(A \supset \neg B)$.
9. $A \vee B \ \sim \neg A \supset B$.

**(4) Формальные операции, правила вывода**

Применяются *основные операции* вывода и *дополнительные*. *Основные операции* применимы во всех системах исчисления. В основу *дополнительных операций* вывода положены эквивалентные преобразования, а, значит, соответствующие исчисления должны применять каким-то образом операцию эквивалентности ($\sim$). Лишь это является ограничением для использования дополнительных операций вывода.

**Правило подстановки — *основная операция***

Определение. *Подстановка* — это операция, при которой некоторая пропозициональная буква $X$ в данной формуле $A(X)$ на всех местах ее вхождения заменяется на некоторую новую формулу $B$. Формальная запись: $A(X)_{|B \to X} = A(B)$.

Замечание (важное). Операция *подстановки* применяется только в тождественно истинные формулы $A(X)$. Смысл замечания раскрывает следующий пример. Пусть $A(X) = X \vee Y$

и $B = \neg Y$. Тогда $A(X)_{|B \to X} = \neg Y \vee Y$. Здесь исходная формула $X \vee Y$ — условно истинная, а $\neg Y \vee Y$ — тождественно истинная, т. е. совершенно изменилось содержание формулы. Применение же операции подстановки в тождественно истинные формулы не изменяет их содержания.

**Правило заключения по импликации modus ponens — *основная операция***

$\dfrac{A, A \supset B}{B}$ — формальная запись операции modus ponens. Применяется к любым формулам $A$ и $B$, при этом результат операции $B$ является настолько же истинным, насколько истинны посылки $A$ и $A \supset B$.

**Правило подстановки по эквивалентности — *дополнительная операция***

$G(A)_{|A \sim B} = G(B)$ — формальная запись подстановки по эквивалентности. Если в формуле $G(A)$ имеется подформула $A$, эквивалентная формуле $B$, то подстановкой по эквивалентности получается формула $G(B)$ из $G(A)$, в которой подформула $A$ заменена эквивалентной формулой $B$.

**Правило замены по эквивалентности — *дополнительная операция***

$\dfrac{A, A \sim B}{B}$ и $\dfrac{B, A \sim B}{A}$ — формальная запись замены по эквивалентности. Очевидно, что истинность заключения $B$ (и $A$) определяется истинностью посылки $A$ (и $B$).

**(5) Формализация доказательства и вывода**

Определение. *Доказательством* некоторой формулы $A$ называется конечная последовательность формул, каждая из которых может быть:

1) аксиомой;
2) подстановкой в тождественно истинную формулу: (a) в аксиому, (b) в формулу результата операции modus ponens;
3) формулой как результатом применения операции modus ponens, посылками которой выступают предыдущие формулы списка доказательства;
4) доказуемой формулой $A$ — последней формулой списка доказательства.

Замечание. Конечная формула $A$ называется *доказуемой* или формальной *теоремой*. Доказательство формулы $A$ обозначают как $\vdash A$.

Задача. Доказать формулу

$$\vdash A \supset A. \tag{1.38}$$

Доказательство в системе аксиом Клини (mp = modus ponens).

1. $A \supset (B \supset A)$, a.1.
2. $A \supset (A \supset A)$, a.1, где $A \to B$ — подстановка.
3. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$, a.2.
4. $(A \supset (A \supset A)) \supset ((A \supset ((A \supset A) \supset A)) \supset (A \supset A))$, a.2, где $(A \supset A) \to B$ и $A \to C$ — подстановка.
5. $(A \supset ((A \supset A) \supset A)) \supset (A \supset A)$, mp 2,4.
6. $A \supset ((A \supset A) \supset A)$, a.1, где $(A \supset A) \to B$ — подстановка.
7. $A \supset A$, mp 6,5. □

Замечание. Понятие *формального* вывода является обобщением *доказательства*, где кроме аксиом дополнительно дается перечень условно истинных формул (УИФ) $\Gamma = \{G_1, G_2, \ldots$ $\ldots, G_n\}$, которые не могут быть получены из аксиом формально, но являются (принимаются) истинными на протяжении всего вывода. *Например*, $A \supset B$ — УИФ.

Замечание. В УИФ нельзя делать подстановки. Содержание таких формул может измениться на тождественно истинное или ложное. *Например*, $\neg A \supset B \underset{\neg(Y \vee \neg Y) \to B}{\overset{(X \vee \neg X) \to A}{}} =$ $= \neg(X \vee \neg X) \vee \neg(Y \vee \neg Y))$ — тождественно ложная функция.

Определение. *Выводом* некоторой формулы $Z$ называется конечная последовательность формул, каждая из которых может быть:

1) аксиомой;
2) УИФ — условно истинной формулой из перечня $\Gamma = \{G_1, G_2, \ldots, G_n\}$ без подстановки;
3) подстановкой в тождественно истинную формулу: (a) в аксиому, (b) в формулу результата операции modus ponens.
4) формулой как результатом применения операции modus ponens, посылками которой выступают предыдущие формулы списка вывода;
5) выводимой формулой $Z$ — последней формулой списка вывода.

Замечание. Конечная формула $Z$ называется *выводимой*. Вывод формулы $Z$ при условии $\Gamma$ обозначают как $\Gamma \vdash Z$ или $G_1, G_2, \ldots, G_n \vdash Z$.

Задача. Доказать $A \wedge B \supset C, A, B \vdash C$.

Доказательство в системе аксиом Клини.

1. $A \supset (B \supset A \wedge B)$, а.3.
2. $A$, УИФ.
3. $B \supset A \wedge B$, mp 2,1.
4. $B$, УИФ.
5. $A \wedge B$, mp 4,3.
6. $A \wedge B \supset C$, УИФ.
7. $C$, mp 5,6. □

Замечание. Формальный вывод — это строгая последовательность формул. В связи с этим становятся важными любые указания на существование вывода. Теорема о *дедукции* позволяет делать заключения о существовании *вывода* какой-либо формулы на основании уже существующих *выводов* других формул.

Теорема 1.5.1 (о дедукции, 1930). *Пусть* $A, \Gamma$ — *совокупность* УИФ, *где* $\Gamma = \{G_1, G_2, \ldots, G_n\}$. *Формула* $B$ *выводима из совокупности* $A, \Gamma$ *тогда и только тогда, когда из* $\Gamma$ *выводима формула* $A \supset B$. *Итак*,

$$A, \Gamma \vdash B \;\Leftrightarrow \Gamma \vdash A \supset B.$$

Доказательство *необходимости*. По условию имеем вывод формулы $\Gamma \vdash A \supset B$ . Требуется показать, что тогда существует и вывод $A, \Gamma \vdash B$. Построим вывод этой формулы:

1. $A$, УИФ из условия $A, \Gamma \vdash B$.
2. $A \supset B$, УИФ из вывода $\Gamma \vdash A \supset B$.
3. $B$, mp 1,2.

*Доказательство достаточности* — это индукция по длине $k$ существующего по условию вывода $A, \Gamma \vdash B$. Все выводы будем строить в системе аксиом Клини.

*Установочный шаг индукции*. Проверим справедливость условия теоремы $A, \Gamma \vdash B \Rightarrow \Gamma \vdash A \supset B$, если длина существующего вывода $A, \Gamma \vdash B$ будет равна 1, т. е. $k = 1$. Построим в этом случае вывод формулы $\Gamma \vdash A \supset B$.

Итак, длина списка вывода $A, \Gamma \vdash B$ состоит из одной формулы $B$. В этом случае $B$ может быть лишь одной из формул совокупности $A$, $\Gamma$ или *аксиомой*.

*Случай* 1. Пусть $B$ — это одна из формул $\Gamma$ или аксиома. Следовательно, при построении вывода $\Gamma \vdash A \supset B$ можно пользоваться и формулой $B$. Построим этот вывод:

1. $B$, одна из формул $\Gamma$ или *аксиома*.
2. $A \supset (B \supset A)$, a.1.
3. $B \supset (A \supset B)$, a.1, где $A \to B$, $B \to A$ — подстановка.
4. $A \supset B$, mp 1,3.

*Случай* 2. Пусть $B$ — это формула $A$. Тогда следует построить вывод $\Gamma \vdash A \supset A$ и формулой $A$ нельзя пользоваться, ее нет в списке $\Gamma$. Доказательство тождественно истинной формулы $\vdash A \supset A$ дано выше в *примере* (1.38).

*Шаг предположения индукции*. Предположим, что условие теоремы $A, \Gamma \vdash B \Rightarrow \Gamma \vdash A \supset B$ выполняется для всех случаев, когда длина вывода $A, \Gamma \vdash B$ не превышает $k$.

*Шаг распространения индукции*. Рассмотрим некоторый вывод $A, \Gamma \vdash B$ длиной $k+1$. Покажем, что и в этом случае существует вывод $\Gamma \vdash A \supset B$. В выводе $A, \Gamma \vdash B$ формула $B$ может быть: (случай 1) либо одной из списка $\Gamma$ или аксиомой, (случай 2) либо формулой $A$, (случай 3) либо *новой формулой*, как результатом операции заключения по импликации modus ponens.

Доказательство *случаев* 1 и 2 повторяет доказательство *установочного шага индукции*.

*Случай* 3. Полагаем, что в условии вывода $A, \Gamma \vdash B$ формула $B$ является новой и получена как результат операции modus ponens, примененной к предыдущим формулам списка вывода. Без ограничения общности можно полагать, что вывод формулы $B$ из $A, \Gamma \vdash B$ представляется списком

$$\underbrace{\overbrace{A_1, A_2, \ldots, A_m}^{m \leqslant k-2}, P, A_{m+1}, A_{m+2}, \ldots, A_{k-2}, P \supset B}_{k}, \underbrace{B}_{k+1}, \qquad (1.39)$$

где формула $B$ — последняя $k+1$ формула списка. Из списка (1.39) имеем выводы $A, \Gamma \vdash P$ и $A, \Gamma \vdash P \supset B$, длины которых не превышают значения $k$. Таким образом, для них выполняется *предположение индукции* и, значит, существуют выводы формул $\Gamma \vdash A \supset P$ и $\Gamma \vdash A \supset (P \supset B)$. В этих условиях продолжим вывод формулы $\Gamma \vdash A \supset B$:

1. $A \supset P$, УИФ.
2. $A \supset (P \supset B)$, УИФ.
3. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$, a.2.
4. $(A \supset P) \supset ((A \supset (P \supset B)) \supset (A \supset B))$, a.2, где $P \to B, B \to C$.

5. $(A \supset (P \supset B)) \supset (A \supset B)$, mp 1,4.
6. $A \supset B$, mp 2,5.

Теорема о дедукции (ТОД) доказана. □

Следствие ТОД. *Если формула $A$ выводима из совокупности формул $G_1, G_2, \ldots, G_n \vdash A$, то доказуемой является следующая формула* $\vdash G_1 \supset (G_2 \supset \ldots (G_n \supset A))$.

Пример. Построим вывод формулы

$$A, B, A \supset (B \supset C) \vdash C. \tag{1.40}$$

1. $A$, УИФ.
2. $B$, УИФ.
3. $A \supset (B \supset C)$, УИФ.
4. $B \supset C$, mp 1,3.
5. $C$, mp 2,4.

По теореме о дедукции из существования вывода (1.40) заключаем, что существуют следующие выводы и доказательства:

$$\begin{array}{ll}
1. & B, A \supset (B \supset C) \vdash A \supset C. \\
2. & A, A \supset (B \supset C) \vdash B \supset C. \\
3. & A, B \vdash (A \supset (B \supset C)) \supset C. \\
4. & A \supset (B \supset C) \vdash B \supset (A \supset C). \\
5. & A \supset (B \supset C) \vdash A \supset (B \supset C). \\
6. & \vdash (A \supset (B \supset C)) \supset (B \supset (A \supset C)). \\
7. & \vdash (A \supset (B \supset C)) \supset (A \supset (B \supset C)).
\end{array} \tag{1.41}$$

Пример. Построим вывод $B, A \supset (B \supset C) \vdash A \supset C$, (1.41.1).

1. $B \supset (A \supset B)$, a.1, где $A \to B$, $B \to A$.
2. $B$, УИФ.
3. $A \supset B$, mp 2,1.
4. $(A \supset B) \supset ((A \supset (B \supset C)) \supset (A \supset C))$, a.2.
5. $(A \supset (B \supset C)) \supset (A \supset C)$, mp 3,4.
6. $A \supset (B \supset C)$, УИФ.
7. $A \supset C$, mp 6,5.

### Характерные выводимости

$$\begin{array}{lll}
1. & A \supset B, B \supset C \vdash A \supset C & \text{— транзитивность.} \\
2. & A \supset (B \supset C) \vdash B \supset (A \supset C) & \text{— перестановка посылок.} \\
3. & A \supset (B \supset C) \vdash A \wedge B \supset C & \text{— объединение посылок.} \\
4. & A \wedge B \supset C \vdash A \supset (B \supset C) & \text{— разъединение посылок.} \\
5. & A \supset B \vdash \neg B \supset \neg A & \text{— контрапозиция.}
\end{array} \tag{1.42}$$

Если пользоваться теоремой о дедукции (ТОД), то нетрудно установить существование приведенных характерных выводимостей, построить же сам вывод, как правило, — не такая простая задача. Далее покажем лишь существование указанных выводов, что следует рассматривать в качестве хорошей практики применения теоремы о дедукции. Используются аксиомы Клини.

*(1) Пример вывода.* $A \supset B, B \supset C \vdash A \supset C$.

1. $A \supset B, B \supset C \vdash A \supset C$ — исходная задача.
2. $A, A \supset B, B \supset C \vdash C$, тод 1 — будем решать.
3. $A$, уиф.
4. $A \supset B$, уиф.
5. $B$, mp 3,4.
6. $B \supset C$, уиф.
7. $C$, mp 5,6.

*(2) Пример вывода.* $A \supset (B \supset C) \vdash B \supset (A \supset C)$.

1. $A \supset (B \supset C) \vdash B \supset (A \supset C)$ — исходная задача.
2. $B, A \supset (B \supset C) \vdash A \supset C$, тод 1.
3. $A, B, A \supset (B \supset C) \vdash C$, тод 2 — будем решать.
4. $A$, уиф.
5. $A \supset (B \supset C)$, уиф.
6. $B \supset C$, mp 4,5.
7. $B$, уиф.
8. $C$, mp 6,7.

*(3) Пример вывода.* $A \supset (B \supset C) \vdash A \wedge B \supset C$.

1. $A \supset (B \supset C) \vdash A \wedge B \supset C$ — исходная задача.
2. $A \wedge B, A \supset (B \supset C) \vdash C$, тод 1 — будем решать.
4. $A \wedge B$, уиф.
5. $A \wedge B \supset A$, а.4.
6. $A \wedge B \supset B$, а.5.
7. $A$, mp 4,5.
8. $B$, mp 4,6.
9. $A \supset (B \supset C)$, уиф.
10. $B \supset C$, mp 7,9.
11. $C$, mp 8,10.

*(4) Пример вывода.* $A \wedge B \supset C \vdash A \supset (B \supset C)$.

1. $A \wedge B \supset C \vdash A \supset (B \supset C)$ — исходная задача.
2. $A, A \wedge B \supset C \vdash B \supset C$, тод 1.
3. $B, A, A \wedge B \supset C \vdash C$, тод 2 — будем решать.
4. $A \supset (B \supset A \wedge B)$, а.3.
5. $A$, уиф.

6. $B \supset A \wedge B$, mp 5,4.
7. $B$, УИФ.
8. $A \wedge B$, mp 7,6.
9. $A \wedge B \supset C$, УИФ.
10. $C$, mp 8,9.

*(5) Пример вывода.* $A \supset B \vdash \neg B \supset \neg A$.

1. $A \supset B \vdash \neg B \supset \neg A$ — исходная задача.
2. $\neg B, A \supset B \vdash \neg A$, ТОД 1 — будем решать.
3. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$, а.9.
4. $A \supset B$, УИФ.
5. $(A \supset \neg B) \supset \neg A$, mp 4,3.
6. $A \supset (B \supset A)$, а.1.
7. $\neg B \supset (A \supset \neg B)$, а.1, где $\neg B \to A$, $A \to B$.
8. $\neg B$, УИФ.
9. $A \supset \neg B$, mp 8,7.
10. $\neg A$, mp 9,5.

### Операции введения и удаления логических символов

Отметим, что упрощение формул или получение из них новых, как правило, связано с включением или удалением операций и операндов (логических связок, символов и букв). Далее рассматривается перечень таких преобразований формул для каждого символа операции $\supset, \wedge, \vee, \neg$. Пусть $\Gamma = \{G_1, G_2, \ldots, G_n\}$ — произвольный список УИФ, $A, B, C$ — произвольные формулы.

1. *Введение импликации.*
   Если $\Gamma, A \vdash B$, то $\Gamma \vdash A \supset B$ — ТОД.

1а. *Удаление импликации.*
   $A, A \supset B \vdash B$ — mp.

2. *Введение конъюнкции* $A, B \vdash A \wedge B$.
   1. $A \supset (B \supset A \wedge B)$, а.3.
   2. $A$, УИФ.
   3. $B \supset A \wedge B$, mp 2,1.
   4. $B$, УИФ.
   5. $A \wedge B$, mp 4,1.

2а. *Удаление конъюнкции* $A \wedge B \vdash A$ и $A \wedge B \vdash B$.
   1. $A \wedge B \supset A$, а.4.
   2. $A \wedge B$, УИФ.
   3. $A$, mp 2,1.

3. *Введение дизъюнкции* $A \vdash A \vee B$ и $B \vdash A \vee B$.
   1. $A \supset A \vee B$, а.6.
   2. $A$, УИФ.
   3. $A \vee B$, mp 2,1.

3а. *Удаление дизъюнкции*. Если $A \vdash C$ и $B \vdash C$, то $A \vee B \vdash C$.
1. $A \vdash C$. $1^* \vdash A \supset C$, тод 1.
2. $B \vdash C$. $2^* \vdash B \supset C$, тод 2.
3. $A \vee B \vdash C$. $3^* \vdash (A \vee B) \supset C$, тод 3.
4. $A \supset C, B \supset C \vdash (A \vee B) \supset C$ — выводим.
5. $(A \supset C) \supset ((B \supset C) \supset ((A \vee B) \supset C))$, а.8.
6. $A \supset C$, уиф.
7. $(B \supset C) \supset ((A \vee B) \supset C)$, mp 6,5.
8. $B \supset C$, уиф.
9. $(A \vee B) \supset C$, mp 8,7.

4. *Введение отрицания*. Если $A \vdash B$ и $A \vdash \neg B$, то $\vdash \neg A$.
1. $A \vdash B$. $1^* \vdash A \supset B$, тод 1.
2. $A \vdash \neg B$. $2^* \vdash A \supset \neg B$, тод 2.
3. $A \supset B, A \supset \neg B \vdash \neg A$ — выводим.
4. $(A \supset B) \supset ((A \supset \neg B) \supset \neg A)$, а.9.
5. $A \supset B$, уиф.
6. $(A \supset \neg B) \supset \neg A$, mp 8,7.
7. $A \supset \neg B$, уиф.
8. $\neg A$, mp 5,4.

4а. *Слабое удаление отрицания* $A, \neg A \vdash B$.
1. $A \supset (\neg B \supset A)$, а.1, где $\neg B \to B$.
2. $A$, уиф.
3. $\neg B \supset A$, mp 2,1.
4. $(\neg B \supset A) \supset ((\neg B \supset \neg A) \supset \neg\neg B)$, а.9, где $\neg B \to A$, $A \to B$.
5. $(\neg B \supset \neg A) \supset \neg\neg B$, mp 3,4.
6. $\neg A \supset (\neg B \supset \neg A)$, а.1, где $\neg A \to A$, $\neg B \to B$.
7. $\neg A$, уиф.
8. $\neg B \supset \neg A$, mp 7,6.
9. $\neg\neg B$, mp 8,5.
10. $\neg\neg B \supset B$, а.10.
11. $B$, mp 9,10.

4b. *Сильное удаление отрицания* $\neg\neg A \vdash A$.
1. $\neg\neg A$, уиф.
2. $\neg\neg A \supset A$, а.10.
3. $A$, mp 1,2.

### (6) Общие внешние оценки исчисления высказываний

В любой формальной теории исчисления основными внешними ее оценками являются:

1) непротиворечивость исчисления;
2) полнота системы аксиом;
3) независимость системы аксиом;
4) разрешимость исчисления.

### (1) Непротиворечивость исчисления

Определение. Любая формальная система исчисления высказываний, имеющая символ $\neg$ для отрицания, называется *непротиворечивой*, если в ней одновременно недоказуема некоторая формула $A$ и $\neg A$.

Замечание. Для доказательства *непротиворечивости* исчисления высказываний необходимо найти некоторое свойство аксиом и правил вывода такое, что

1) все аксиомы обладают этим свойством;
2) правила вывода сохраняют это свойство в формулах, если оно имелось в последних;
3) любая пара формул $A$ и $\neg A$ одновременно этим свойством обладать не могут.

Искомым свойством является *свойство тождественной истинности* при нормальной интерпретации. Нормальная булева интерпретация подразумевает, что все пропозициональные переменные могут принимать два значения — 0 и 1, а все пропозициональные связки определяются таблицей истинности булевых функций.

Замечание. Формальное исчисление высказываний является *непротиворечивой* системой. Действительно, ее система аксиом содержательно является совокупностью ТИФ и правила вывода сохраняют это свойство в новых формулах. Поэтому ни для какой формально выводимой формулы $A$ (она ТИФ) нельзя вывести противоположную ей $\neg A$ (она не ТИФ). Можно сказать, что формальное исчисление высказываний является замкнутым в области ТИФ.

### (2) Полнота системы аксиом

Определение содержательной полноты. Исчисление высказываний называется *содержательно* полным, если из его системы аксиом выводима любая *тождественно истинная формула* (ТИФ).

Теорема 1.5.2 (о содержательной полноте). *Исчисление высказываний является содержательно полным.*

Доказательство выполним на примере функций двух переменных для аксиом *расширенного исчисления высказываний*. Идея доказательства легко распространяется на другие системы аксиом исчисления высказываний и функции произвольного числа аргументов.

Пусть $F(x,y)$ — произвольная ТИФ двух переменных. Необходимо показать, что она доказуема $\vdash F(x,y)$. Таблица истинности такой функции на всех значениях переменных $x,y$ равна 1 (истина в интерпретации). Тогда ее СДНФ имеет вид $\overline{x}\,\overline{y} \vee \overline{x}\,y \vee x\,\overline{y} \vee x\,y$. СДНФ является полной алгебраической формой относительно двух переменных $x,y$. Нетрудно заметить, что эта форма представима в виде $\overline{x}\,\overline{y} \vee \overline{x}\,y \vee x\,\overline{y} \vee x\,y = (x \vee \overline{x})(y \vee \overline{y})$. Так как $F(x,y)$ и ее СДНФ — это равные функции, то они эквивалентны. Таким образом, эквивалентность $F(x,y) \sim (x \vee \overline{x})(y \vee \overline{y})$ по построению является тождественно истинной формулой. Следующая последовательность формул является искомым доказательством $\vdash F(x,y)$:

1. $F(x,y) \sim (x \vee \overline{x})(y \vee \overline{y})$, ТИФ — *по построению*.
2. $x \vee \overline{x}$, а.19.
3. $y \vee \overline{y}$, а.19, где $y \to x$.
4. $(x \vee \overline{x})(y \vee \overline{y})$ — *введение конъюнкции* 2,3.
5. $F(x,y)$ — *правило замены по эквивалентности* 4,1. □

Определение формальной полноты. Система аксиом исчисления высказываний называется *формально* полной, если при добавлении к ней любой невыводимой из нее формулы система становится противоречивой.

Теорема 1.5.3 (о полноте). *В исчислении высказываний формальная полнота и содержательная полнота являются эквивалентными понятиями.*

Доказательство. Докажем, что из содержательной полноты следует формальная полнота исчисления высказываний. Содержательная полнота говорит о том, что любая ТИФ является выводимой из системы аксиом. С другой стороны, из системы аксиом можно получать только тождественно истинные формулы. Поэтому, по определению, для проверки формальной полноты исчисления высказываний следует включить в систему ее аксиом произвольную *не тождественно истинную* формулу и показать, что в этом случае система становится противоречивой. Пункты доказательства отмечаем порядковыми номерами.

1. Имеем, что из системы аксиом выводима любая ТИФ.
2. Пусть $B(x_1,x_2,\ldots,x_n)$ — некоторая невыводимая формула, $x_i \in \{0,1\}$. Значит, она не ТИФ.
3. Следовательно, существует такой набор $(a_1,a_2,\ldots,a_n)$ значений переменных, что $B(a_1,a_2,\ldots,a_n) = 0$.
4. Присоединим формулу $B(x_1,x_2,\ldots,x_n)$ к системе аксиом.

5. Выполним подстановку

$$B(x_1, x_2, \ldots, x_n) = C(x) = B(a_1, a_2, \ldots, a_n) = 0,$$
$$\begin{vmatrix} x \vee \overline{x} \to x_i, & \text{если } a_i = 1, \\ x \wedge \overline{x} \to x_i, & \text{если } a_i = 0. \end{vmatrix}$$

где $C$ — тождественно *ложная* формула или $\neg C$ — ТИФ.
6. Имеем правило (1.37) *из лжи что угодно* $\neg x \supset (x \supset y)$ — ТИФ. Выполним подстановку $C \to x$ и $A \to y$. Тогда $\neg C \supset \supset (C \supset A)$ — ТИФ, где $A$ — произвольная формула.
7. Применяя к тождественно истинным формулам $\neg C$ и $\neg C \supset \supset (C \supset A)$ операцию modus ponens, получим $C \supset A$ — ТИФ.
8. Выполним подстановку $\neg C \to C$ в формулу $C \supset A$ — ТИФ. Имеем $\neg C \supset A$ — ТИФ.
9. Применив к формулам $\neg C$ и $\neg C \supset A$ операцию modus ponens, получим $A$.
10. Выполним подстановку $\neg A \to A$ в формулу $\neg C \supset A$ — ТИФ. Имеем $\neg C \supset \neg A$ — ТИФ.
11. Применив к формулам $\neg C$ и $\neg C \supset \neg A$ операцию modus ponens, получим $\neg A$.

Таким образом, включение в систему аксиом исчисления высказываний произвольной не тождественно истинной формулы приводит к тому, что такая система позволяет получить пару формул $A$ и $\neg A$. Такая система аксиом является *противоречивой*. Исходная же система аксиом была непротиворечивой. Это доказывает, что из содержательной полноты следует формальная.

*Докажем*, что из формальной полноты следует содержательная полнота. Доказательство выполним от противного. Допустим, что система формально полна, однако содержательной полноты нет.

1. Так как содержательная полнота не выполняется, значит, существует тождественно истинная формула $B$, которая не выводима из системы аксиом.
2. Добавим указанную $B$ — ТИФ к системе аксиом.
3. Замечая, что добавляемая формула $B$ и аксиомы — это совокупность ТИФ, получаем, что вывод из этой совокупности возможен лишь тождественно истинных формул.
4. По условию исходная система аксиом является формально полной. Поэтому добавление к ней формулы $B$ должно обратить систему в противоречивую, из которой возможен вывод пары формул $A$ и $\neg A$. Согласно п. 3 эти формулы должны быть ТИФ. Однако $A$ и $\neg A$ не могут быть одновременно ТИФ. Следовательно, предположение, что при формальной полноте содержательной полноты нет, является неверным. □

### (3) Независимость системы аксиом

Определение. Система аксиом исчисления высказываний называется *независимой*, если никакая из аксиом этой системы не может быть получена из других с использованием допустимых правил вывода.

Замечание. С точки зрения использования теории независимость системы аксиом — менее важное свойство, чем полнота. Независимость аксиом важна при переходе от одной теории к другой.

Замечание. Определение независимости системы аксиом является непростой задачей. При решении ее используют тот же подход, что и при установлении непротиворечивости системы. Пусть необходимо установить независимость какой-либо *выделенной* аксиомы, тогда последовательность решений этой задачи состоит в следующем.

1. Находят некоторое *свойство* формул, которое *избирательно* выполняется в аксиомах всего списка.
2. Это свойство должно быть в наличии у всех аксиом, кроме *выделенной*. У выделенной аксиомы это *свойство* должно отсутствовать.
3. Данное свойство должно сохраняться правилами вывода.
4. Допустим, что это все выполнено и свойство найдено, тогда исключение *выделенной* аксиомы из всего перечня их списка приведет к следующему эффекту.
5. Оставшаяся часть аксиом позволит нам получать лишь формулы, у которых всегда в наличии *выделенное* свойство. И, как выше сказано, это свойство сохраняется правилами вывода.
6. С другой стороны, существуют формулы, которые в нормальной булевой интерпретации являются ТИФ и не могут быть получены из оставшейся части списка аксиом. Это те тождественно истинные формулы, которые не обладают *выделенным* свойством. Например, такой формулой является удаленная аксиома.
7. Таким образом, удалив обозначенную аксиому, мы в целом нарушили полноту системы аксиом.

Замечание. Как правило, в качестве указанного *выделенного* свойства аксиом предлагается использовать их свойство быть тождественно истинными формулами. И это свойство должно сохраняться правилами вывода. Используется нормальная бу-

лева интерпретация: 0 — ложь, 1 — истина. Далее пытаются изменить таблицы истинности некоторых операций (связок) так, чтобы в новой интерпретации пропозициональных связок все аксиомы остались ТИФ, однако *выделенная* аксиома *не должна* остаться ТИФ. Это и будет доказательством *независимости* выделенной аксиомы.

Пример. Рассмотрим систему аксиом Клини (п. 1.5.5). Покажем независимость ее четвертой аксиомы $A \wedge B \supset A$. Определим таблицу истинности конъюнкции $A \wedge B$ как $B$. Итак, в новой интерпретации $A \wedge B = B$. Отметим, что аксиомы, в которые не входит операция $A \wedge B$, остались и в новой интерпретации тождественно истинными формулами. Оценим теперь истинность аксиом, в которые входит операция $A \wedge B$.

*Аксиома* 3. $A \supset (B \supset A \wedge B)$. Так как в новой интерпретации $A \wedge B = B$, то $A \supset (B \supset A \wedge B) = A \supset (B \supset B)$ — ТИФ.

*Аксиома* 4. $A \wedge B \supset A$. Так как имеем $A \wedge B = B$, то $A \wedge B \supset A = B \supset A$ — условно истинная формула (не ТИФ).

*Аксиома* 5. $A \wedge B \supset B$. Так как имеем $A \wedge B = B$, то $A \wedge B \supset B = B \supset B$ — ТИФ.

Видно, что в предложенной интерпретации $A \wedge B = B$ все аксиомы являются ТИФ, кроме выделенной $A \wedge B \supset A$, которая не ТИФ. Следовательно, можем заключить, что аксиома $A \wedge B \supset A$ является независимой. Ее нельзя получить из других аксиом, которые являются ТИФ в рассматриваемой интерпретации.

### (4) Разрешимость исчисления высказываний

Определение. Формальная теория исчисления высказываний называется *разрешимой*, если имеется эффективная (значит, конечная) процедура, дающая ответ на вопрос — выводима ли формула $A$ из заданного списка формул $G_1, G_2, \ldots, G_n$.

Теорема о дедукции позволяет записать условие выводимости

$$G_1, G_2, \ldots, G_n \vdash A \tag{1.43}$$

в форме доказательства

$$\vdash G_1 \supset (G_2 \supset (\ldots (G_n \supset A))). \tag{1.44}$$

Из эквивалентности содержательной и формальной полноты исчисления высказываний заключаем, что $G_1 \supset (G_2 \supset (\ldots \ldots (G_n \supset A)))$ — ТИФ. Следовательно, в качестве эффективной

процедуры ответа на вопрос выводимости какой-либо формулы (1.43) может выступать процедура установления тождественной истинности соответствующей ей доказуемой формулы (1.44).

## § 1.6. Исчисление предикатов

Исчисление высказываний является довольно узкой логической теорией. Несложно привести логические рассуждения, истинность которых не может быть объяснена в рамках исчисления высказываний.

Примеры рассуждений.

1). *Имеем* $2 < 5$
*и* $5 < 9$.
*Значит* $2 < 9$.
Рассуждение — истинное. Его заключение является следствием посылок.

2). *Имеем* $2 < 5$
*и* $7 < 9$.
*Значит* $2 < 7$.
Рассуждение — неверное. Его заключение не является следствием посылок.

3). *Все птицы — животные.*
*Все воробьи — птицы.*
*Значит все воробьи — животные.*
Рассуждение — истинное. Его заключение является следствием посылок.

4). *Все птицы — животные.*
*Никакая рыба — не птица.*
*Значит никакая рыба — не животное.*
Рассуждение — неверное. Его заключение не является следствием посылок.

5). *Простое число* 2 *является четным.*
*Следовательно, существуют простые четные числа.*
Рассуждение — истинное. Его заключение является следствием посылки.

Корректность (или некорректность) предложенных умозаключений основана на внутренней структуре самих предложений и на смысле слов *всякий* и *существует*. Если же рассматривать их формализацию в рамках теории исчисления высказываний,

то все они укладываются в одну схему — истинные посылки и истинное (или ложное) заключение, что не может отразить действительного содержания рассуждений.

### 1.6.1. Понятие предиката

С целью введения понятия предиката рассмотрим следующие примеры простых высказываний: *«Платон есть грек»*, *«7 есть простое число»*, *«Михаил есть брат Владимира»*.

В каждом простом высказывании утверждается тот факт, что некоторые объекты обладают некоторыми свойствами. Содержание таких высказываний является разным, однако форма их всех одинаковая. Возьмем высказывание *«Платон есть грек»*. *Обобщением* его является высказывание вида *«X есть грек»*. Предложения такого типа называются *предикатами*, сказуемыми (*predicate* — сказуемое). Действительно, обобщение распространяется на подлежащее (*subject* — предмет, субъект). Сказуемая часть остается без изменения и фиксирует некоторое свойство предиката. Нетрудно заметить, что именно *свойства объектов* находятся между собой в некотором отношении.

Последнее высказывание *«X есть грек»* можно обозначить как функцию одной переменной $\text{Гр}(x)$. Такая функция каждому значению переменной $x$ ставит в соответствие некоторое высказывание. *Например*, $\text{Гр}(\textit{Аристотель}) =$ *«Аристотель есть грек»* или $\text{Гр}(\textit{Наполеон}) =$ *«Наполеон есть грек»*. Здесь предикат выступает в роли *пропозициональной функции*, областью значений которой служат высказывания. Такие высказывания могут быть как истинными, так и ложными. Аргументами (значениями переменных) предикатов также выступают высказывания.

*Например*, обобщением высказывания *«Михаил есть брат Владимира»* может служить предикат *«X есть брат Y»*, который можно обозначить *пропозициональной* функцией двух переменных $\text{Бр}(x, y)$. На такой предикат можно смотреть и как на некоторое отношение между объектами $x$ и $y$.

Определение. В общем случае под *предикатом* понимают некоторое отношение $P(x_1, x_2, \ldots, x_n)$ (или пропозициональную функцию), которое устанавливает связь между определенным числом предметных переменных (объектов) $x_1, x_2, \ldots, x_n \in D$. В рамках классического исчисления предикатов полагают, что высказывание $P(x_1, x_2, \ldots, x_n)$, соответствующее произвольным значениям предметных переменных $x_1, x_2, \ldots, x_n \in$

$\in D$, непременно является либо истинным, либо ложным. По количеству переменных отношение $P(x_1, x_2, \ldots, x_n)$ называют $n$-местным предикатом.

### 1.6.2. Операции над предикатами

В исчислении высказываний с помощью операций (логических связок) $\neg$, $\wedge$, $\vee$, $\supset$, $\sim$ формировались сложные высказывания. Подобным же образом в *исчислении предикатов* посредством указанных операций формируются новые предикаты. Например, $S(x, z) \supset P(x, y) \vee Q(x, y, z)$.

Примеры операций.

1). Рассмотрим два предиката

$$\text{Чет}(x) = \{x - \textit{четное число}\} \text{ и}$$

$$\text{Пр}(x) = \{x - \textit{простое число}\}.$$

На основе обозначенных предикатов построим новые предикаты.

$$\text{Чет}(x) \wedge \text{Пр}(x) = \{x - \textit{четное и простое число}\}.$$

$$\text{Чет}(x) \vee \text{Пр}(x) = \{x - \textit{четное или простое число}\}.$$

2). Пусть предикат $D(x, y) = \{x \textit{ делит } y\}$.
Новый предикат $D(x, y) \wedge D(y, x) = \{x = y \neq 0\}$.

3). Определим следующие предикаты:

$$\text{М}(x, y) = \{x - \textit{мать } y\},$$

$$\text{Ж}(x, y) = \{x - \textit{жена } y\},$$

$$\text{Б}(x, y) = \{x - \textit{бабушка } y\},$$

$$\text{Тщ}(x, y) = \{x - \textit{теща } y\}.$$

Новые предикаты:

$$\text{М}(x, y) \wedge \text{М}(y, z) = \text{Б}(x, z),$$

$$\text{Ж}(x, y) \wedge \text{М}(z, x) = \text{Тщ}(z, y).$$

### 1.6.3. Определение кванторов

Пусть $Q(x, y, z)$ — произвольный предикат и $x, y, z \in D$.

1. Фиксация всех $x, y, z$ на множестве $D$ значений предметных переменных превращает предикат в единичное высказывание.
2. Суждение в предикате $P(x, y, z)$ обо всем множестве значений какой-либо из переменных $x, y, z \in D$ превращает его в обобщенное высказывание.

Определение. Переход от предиката к обобщенному высказыванию называется *операцией квантификации*.

*Квантор общности* $\forall$. Пусть $P(x)$ — некоторый предикат, $x \in D$. Тогда под выражением $\forall x\ P(x)$ будем понимать обобщенное высказывание, которое истинно, если высказывание $P(x)$ истинно для каждого значения $x \in D$.

*Квантор существования* $\exists$. Под выражением $\exists x\ P(x)$ будем понимать обобщенное высказывание, которое истинно, если существует такое значение $x \in D$, для которого $P(x)$ истинно.

Если положить $D = \{a_1, a_2, a_3\}$, тогда определение обобщенных высказываний $\forall x\ P(x)$ и $\exists x\ P(x)$ представляется в виде

$$\begin{array}{l} \forall x\ P(x) = P(a_1) \wedge P(a_2) \wedge P(a_3), \\ \exists x\ P(x) = P(a_1) \vee P(a_2) \vee P(a_3). \end{array} \tag{1.45}$$

Утверждение 1.6.1. *Пусть $P(x)$ — произвольный предикат. Тогда справедливы тождества* (*законы де Моргана*)

$$\begin{array}{l} \neg\forall x\ \ P(x) = \exists x\ \ \neg P(x), \\ \neg\exists x\ \ P(x) = \forall x\ \ \neg P(x). \end{array} \tag{1.46}$$

Доказательство. Пусть $x \in D = \{a_1, a_2, a_3\}$. Тогда

$$\begin{aligned} \neg\forall x\ P(x) &= \neg(P(a_1) \wedge P(a_2) \wedge P(a_3)) = \\ &= \neg P(a_1) \vee \neg P(a_2) \vee \neg P(a_3) = \exists x\ \neg P(x). \end{aligned}$$

Подобным образом

$$\begin{aligned} \neg\exists x\ P(x) &= \neg(P(a_1) \vee P(a_2) \vee P(a_3)) = \\ &= \neg P(a_1) \wedge \neg P(a_2) \wedge \neg P(a_3) = \forall x\ \neg P(x). \end{aligned}$$ □

Замечание. Доказанные свойства (1.46) можно рассматривать в качестве правила переноса кванторов через символ отрицания.

### Свободные и связанные переменные

Пусть $\forall y\ Q(x, y, z)$ — произвольный предикат и $x, y, z \in D$.

1. Переменная $x$ называется *свободной* в предикате $\forall y\ Q(x, y, z)$, если ее можно заменить на любое значение $x \in D$.
2. Переменная $y$ в предикате $\forall y\ Q(x, y, z)$ называется *связанной*, если она квантифицирована. Такая переменная характеризует все множество ее значений $y \in D$.

3. Свободные переменные при их квантификации переходят в разряд связанных. Связывание (квантифицирование) у предиката каждой переменной понижает на единицу его «местность». Так, квантификация переменной $y$ в трехместном предикате $Q(x, y, z)$ превращает его в двухместный предикат $\forall y\ Q(x, y, z)$.

### Область действия кванторов

Приведем пример предиката

$$(\forall x\ \ Q(x)) \vee (\exists x\ \ P(x) \wedge \neg R(x)) \supset \underbrace{\neg R(x)}_{x\ -\ \text{свободная}} \ \vee \ \underbrace{S(x, y)}_{x, y\ -\ \text{свободные}}.$$

1. Область действия квантора определяется *скобками*.
2. Если скобок нет, то действие квантора распространяется на всю часть предиката, которая стоит после такого квантора.

Так, в приведенном примере обозначены свободные переменные, на которые не распространяется действие кванторов. Остальные переменные выражений (составляющие предиката) являются связанными.

## 1.6.4. Строение исчисления предикатов

Приведем основной перечень признаков, который используется при классификации составных частей исчисления предикатов.

1. Количество элементов в множестве значений предикатов $P(x_1, x_2, \ldots, x_n)$. В рамках классического исчисления предикатов полагают, что значение непременно должно быть либо истинным, либо ложным.
2. Количество предметных переменных в предикате. Так, $P(x_1, x_2, \ldots, x_n)$ — $n$-местный предикат.
3. Количество предметных областей. В общем случае для предиката $P(x_1, x_2, \ldots, x_n)$ каждая из таких переменных принадлежит свой области $x_i \in D_i$.
4. Количество ступеней подчинения в иерархии переменных предиката.

Дадим более подробное определение каждого из указанных пунктов.

### (1) Классификация по количеству элементов в множестве значений предикатов

1. Теория двузначных предикатов.
2. Теория трехзначных предикатов.
3. Теория $k$-значной логики.

4. Теория бесконечнозначной логики (модальная, вероятностная и др.)

**(2) Классификация по количеству предметных переменных**

1. Теория нульместных предикатов. Это исчисление высказываний.
2. Теория одноместных предикатов или исчисление классов аристотелевых силлогизмов.
3. Теория $n$-местных предикатов.

**(3) Классификация по количеству предметных областей переменных**

1. Исчисление предикатов с одной предметной областью. *Например*, $P(x, y, z)$, где $x, y, z \in D$.
2. Исчисление предикатов с несколькими разнородными предметными областями. *Например*, $P(x, y, z)$, где $x \in D_1$, $y \in D_2$, $z \in D_3$.

**(4) Классификация по количеству ступеней подчинения в иерархии переменных предикатов**

1. Исчисление предикатов 1-й ступени (*узкое исчисление предикатов*), одна ступень подчинения. Определим понятие ступени подчинения переменных. Пусть $P(x)$, $Q(x, y)$, $S(y, z)$ — произвольные предикаты. Переменные $x, y, z \in D$ называются *предметными* переменными. Если же непосредственно сами предикаты выступают в роли переменных, то такие переменные называются *предикатными*. В исчислении предикатов 1-й ступени квантифицироваться могут лишь *предметные* переменные. Пусть $P(x)$ — некоторый предикат. Тогда предикатом 1-й ступени будет предикат

$$\forall x \forall y (\, x = y \,) \supset (\, P(x) = P(y) \,) \ \text{— предикат 1-й ступени,}$$
$$\forall x \forall y \forall Q \, (\, x = y \,) \supset (\, Q(x) = Q(y) \,) \ \text{— предикат НЕ 1-й ступени.}$$

2. Исчисление предикатов 2-й ступени (*расширенное исчисление предикатов*), две ступени подчинения. Вводятся понятия предикатных переменных первого ранга и второго. У предикатов первого ранга переменными могут выступать лишь предметные переменные. Предикаты второго ранга в качестве переменных имеют уже предикаты первого ранга. В исчислении предикатов 2-й ступени квантифицироваться уже могут *предметные* переменные и предикатные переменные первого ранга.

3. По аналогии вводят исчисление предикатов $n$-й ступени, $n$ ступеней подчинения. Рассматриваются предикатные переменные рангов $1, 2, \ldots, n$. У предикатов большего ранга в качестве переменных могут выступать предикаты меньшего ранга. В исчислении предикатов $n$-й ступени квантифицироваться уже могут предметные переменные и предикатные всех рангов до $n-1$.

### 1.6.5. Предикаты свойств

Определение. *Предикат свойства* — это логическая функция одной переменной $y = P(x)$, где $x \in D$ — область значений предметных переменных, $y \in \{\textit{истина}, \textit{ложь}\}$ — множество значений функции. Пользуются интерпретацией значений *«истина»* и *«ложь»* соответственно как 1 и 0 или TRUE и FALSE. По общей классификации предикаты свойств — это одноместные предикаты. Содержание таких предикатов $P(x)$ укладываются в форму (шаблон) *«$x$ обладает свойством $P$»*.

Приведем примеры таких предикатов.

1. $y = \text{Гр}(x) =$ «$x$ есть грек», где $D = \{$Платон, Аристотель, Пифагор, Святогор, Наполеон$\}$.
2. $y = f(x) =$ «$x < 7$», где $D$ — действительные числа.
3. $y = \text{Пр}(x) =$ «$x$ есть простое число», где $D = \{2,3,4,5,6,7\}$.
4. $y = R(x) =$ «$x$ есть река Сибири», где $D$ — множество рек.
5. $y = S(x) =$ «$x$ есть военный крейсер», где $D$ — корабли.

Различают следующие способы задания предикатов.

1. *Аналитический способ.*
   $y = f(x) =$ «$x < 7$», где $D$ — действительные числа.
2. *Табличный способ.*
   $y = \text{Пр}(x) =$ «$x$ есть простое число», где $D = \{2,3,4,5,6,7\}$:

| $x$ | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $\text{Пр}(x)$ | 1 | 1 | 0 | 1 | 0 | 1 |

3. *Логический способ* — смысловая запись предиката.
   $y = \text{Гр}(x) =$ «$x$ есть грек».
4. *Графический способ* — график функции предиката.

Замечание. Предикат свойства $y = P(x)$ разбивает все множество предметных переменных $x \in D$ на два подмножества

$D = T \cup F$ и $T \cap F = \varnothing$ таких, что $\forall x \in T\ P(x)$ — *истинно* и $\forall x \in F\ P(x)$ — *ложно*.

Для конечного множества $D$ нетрудно подсчитать общее число таких предикатов. Пусть $D = \{a, b, c\}$. Каждый предикат определяется разбиением множества $D$ на два подмножества $T$ и $F$, где $D = T \cup F$ и $T \cap F = \varnothing$. Следовательно, количество предикатов свойств равно числу подмножеств множества $D$. Для $D = \{a, b, c\}$ число подмножеств равно 8:

$$\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{b, c\}, \{a, c\}, \{a, b, c\}.$$

Множество $T$ может быть одним из них, тогда $F$ — дополнение $T$ до $D$. Итак, имеем 8 предикатов свойств. В общем случае, если $|D| = m$, то число таких предикатов будет $2^m$.

### 1.6.6. Исчисление одноместных предикатов, исчисление классов

Заголовок данного пункта можно продолжить — это исчисление категорических суждений, исчисление аристотелевых силлогизмов или узкое исчисление одноместных предикатов по общей классификации (п. 1.6.4).

Определение. Пусть $P(x)$ — одноместный предикат (свойство), где $x \in D$. *Характеристическим* множеством данного предиката называется множество таких предметных переменных $x \in D$, для которых предикат $P(x)$ является *истинным* или «$x$ обладает свойством $P$».

1. С каждым предикатом $P(x)$ связывается одноименный ему *класс* (*истинности*) $P$, под которым понимается характеристическое множество этого предиката $P(x)$.

2. Под *универсальным* классом (*универсумом*) понимается множество всех возможных предметных переменных.

3. *Пустой* класс — пустое множество переменных.

#### Категорические суждения

Рассматриваемое исчисление — это логика Аристотеля (384–322 гг. до н. э.) и его последователей. Данная логика *актуальна и востребована* в своем применении и в настоящее время. Основная роль отводится четырем видам суждений (умозаключений), называемых *категорическими*. Название *категорическое* суждение идет от греческого kategorikos — «не допускающий иных толкований».

Пусть $S(x)$ и $P(x)$ — предикаты (свойства), форма их содержания выражается шаблонами «$x$ обладает свойством $S$» и «$x$ обладает свойством $P$». Полагаем, что $S$ и $P$ — *классы* истинности предикатов $S(x)$ и $P(x)$.

Упомянутые четыре основных вида категорических суждений записываются в следующем виде.

A. Все $S$ суть $P$ — *общеутвердительное суждение.*
E. Никакое $S$ не суть $P$ — *общеотрицательное суждение.*
I. Некоторое $S$ суть $P$ — *частичноутвердительное суждение.*
O. Некоторое $S$ не суть $P$ — *частичноотрицательное суждение.*

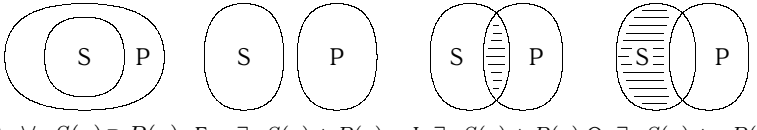
Принятое буквенное сокращенное обозначение суждений A, I, E, O восходит к гласным (**a**, **i**, **e**, **o**) в латинских словах: «**a**ff**i**rmo» (утверждаю) и «n**e**g**o**» (отрицаю). Буквы $S$ и $P$ напоминают о членах предложения, которые называют «**s**ubject» (подлежащее) и «**p**redicate» (сказуемое).

На языке логики предикатов суждения A, E, I, O представляются в следующем виде.

$$\begin{array}{lll}
\text{A.} & \forall x \;\; S(x) \supset P(x) \;\; \textit{или} & \text{Все } S \text{ суть } P. \\
& \forall x \; \neg S(x) \vee P(x). & \\
\text{E.} & \neg\exists x \;\; S(x) \wedge P(x) \;\; \textit{или} & \text{Никакое } S \text{ не суть P.} \\
& \forall x \;\; S(x) \supset \neg P(x) \;\; \textit{или} & \text{Все } S \text{ суть не } P. \\
& \forall x \; \neg S(x) \vee \neg P(x). & \\
\text{I.} & \exists x \;\; S(x) \wedge P(x). & \text{Некоторое } S \text{ суть } P. \\
\text{O.} & \exists x \;\; S(x) \wedge \neg P(x). & \text{Некоторое } S \text{ не суть } P.
\end{array} \tag{1.47}$$

### Геометрическая интерпретация категорических суждений

Геометрическая интерпретация категорических суждений на рис. 1.13 представляется взаимным расположением *классов* истинности $S$ и $P$ предикатов $S(x)$ и $P(x)$.




A. $\forall x\, S(x) \supset P(x)$ E. $\neg\exists x\, S(x) \wedge P(x)$ I. $\exists x\, S(x) \wedge P(x)$ O. $\exists x\, S(x) \wedge \neg P(x)$

Рис. 1.13. Геометрическая интерпретация категорических суждений

Категорические суждения являются обобщенными тождественно истинными высказываниями. Этому условию категори-

ческих суждений отвечает и взаимное расположение соответствующих им классов истинности $S$ и $P$.

### Примеры категорических суждений

Пример 1.

A. (*Все $S$ суть $P$.*) Все курсанты морского училища ($S$) умеют плавать ($P$).

E. (*Никакое $S$ не суть $P$.*) Ни один курсант морского училища ($S$) не умеет плавать ($P$).

I. (*Некоторое $S$ суть $P$.*) Некоторые курсанты морского училища ($S$) умеют плавать ($P$).

O. (*Некоторое $S$ не суть $P$*). Некоторые курсанты морского училища ($S$) не умеют плавать ($P$).

Пример 2.

A. (*Все $S$ суть $P$.*) Все фокусники ($S$) — обманщики ($P$).

E. (*Никакое $S$ не суть $P$.*) Ни один фокусник ($S$) не является обманщиком ($P$).

I. (*Некоторое $S$ суть $P$.*) Некоторые фокусники ($S$) — обманщики ($P$).

O. (*Некоторое $S$ не суть $P$.*) Некоторые фокусники ($S$) не являются обманщиками ($P$).

### Закон отрицания

*Категорические суждения $A$ и $O$, $E$ и $I$ являются отрицанием друг друга*:

$$\neg A = O \quad \text{и} \quad \neg E = I. \tag{1.48}$$

Доказательство $\neg A = O$. Имеем

$$\begin{array}{r} A = \forall x \;\; S(x) \supset P(x) = \forall x \;\; \neg S(x) \vee P(x). \;\; \text{Из (1.46)} \\ \neg A = \neg \forall x \;\; \neg S(x) \vee P(x) = \exists x \;\; S(x) \wedge \neg P(x) = O. \end{array}$$ □

Доказательство $\neg E = I$. Имеем

$$\begin{array}{r} E = \neg \exists x \;\; S(x) \wedge P(x). \;\; \text{Тогда} \\ \neg E = \exists x \;\; S(x) \wedge P(x) = I \end{array}$$ □

### Закон обращения

Категорические суждения $E = \neg \exists x \;\; S(x) \wedge P(x)$ и $I = \exists x \;\; S(x) \wedge P(x)$ допускают замену классов $S$ на $P$ и $P$ на $S$. Справедливость утверждения является следствием свойства коммутативности операции конъюнкции $\wedge$.

### Структура категорических силлогизмов

В классической аристотелевой логике основная роль в познании действительности (в получении новых знаний) отводится дедуктивным (силлогическим) методам (умозаключениям). Учение Аристотеля о силлогизме составляет основу современного направления логики предикатов. *Дедукция* — это частный случай умозаключений. Умозаключение — это форма мышления, посредством которой из одного или нескольких суждений (посылок) выводится новое суждение (заключение). При дедуктивном анализе связь посылок и заключения опирается на логический закон. В силу этого заключение с логической необходимостью вытекает из принятых посылок. Дедуктивная форма умозаключений из истинности посылок всегда ведет к истинности заключения.

*Категорический силлогизм* состоит из трех категорических суждений (1.47), два из которых являются посылками, а третье — заключением. Первая из посылок называется *большой* посылкой, вторая — *малой*. Разновидности силлогизма называются *модусами*. Приведем пример такого силлогизма, называемый *Barbara*.

A. Все $P$ суть $Q$.
A. Все $S$ суть $P$.
---
A. Все $S$ суть $Q$.

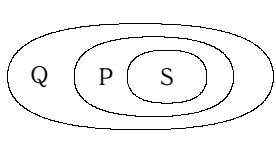



Схема силлогизма *Barbara* представляется в следующем виде:

$$\begin{array}{lll} P\ on\ Q & \text{— большая посылка} \\ S\ on\ P & \text{— малая посылка} \\ \hline S\ on\ Q & \text{— заключение} \end{array}$$

где $P, Q, S$ — классы истинности соответствующих предикатов $P(x), Q(x), S(x)$, сокращение «*on*» — символ операции. В силлогизме классы $P, Q, S$ называются *термами*: $P$ — средний терм, $Q$ — большой терм, $S$ — малый терм.

Все силлогизмы делятся на четыре группы (рис. 1.14), в зависимости от расположения термов.

Геометрический квадрат и линия внутри него — это условное обозначение силлогизма. Линия показывает расположение средних термов $P$ в посылках силлогизма.

Несложно подсчитать общее количество таких силлогизмов. Рассмотрим одну из четырех групп. На место большой по-

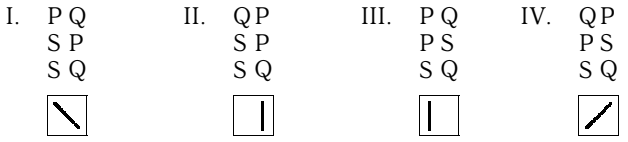



Рис. 1.14. Структурные схемы категорических силлогизмов

сылки, малой посылки и заключения можно поставить любое категорическое суждение (A, E, I, O). Значит, на основе схемы одной группы можно получить $4 \cdot 4 \cdot 4 = 64$ модусов силлогизма. А так как групп четыре, то всего различных модусов силлогизма равно $4 \cdot 64 = 256$. Из этого формально составленного множества модусов лишь 19 являются *правильными* в предположении, что классы истинности $P, Q, S$ являются непустыми. В противном случае их число уменьшается. Силлогизм является *правильно составленным*, если существуют его посылки, при которых заключение является истинным. Силлогизм *неправильный*, если при любых его посылках заключение остается ложным.

Каждый из 19 правильных аристотелевых силлогизмов имеет свое название. Выше в качестве примера рассмотрен силлогизм *Barbara*. Приведем еще два таких силлогизма, относящихся ко *второй группе* их классификации (рис. 1.14).

| **Baroco** | | | | **Festino** | | | |
|---|---|---|---|---|---|---|---|
| A. Все | $Q$ | суть | $P$ | E. Никакое | $Q$ | не суть | $P$ |
| O. Некоторые | $S$ | не суть | $P$ | I. Некоторые | $S$ | суть | $P$ |
| O. Некоторые | $S$ | не суть | $Q$ | O. Некоторые | $S$ | не суть | $Q$ |


### 1.6.7. Техника преобразования формул в исчислении предикатов

Для формул логики предикатов как высказываний выполняются все правила преобразований, установленные в исчислении высказываний. Рассмотрим ряд дополнительных операций пре-

образования формул предикатов, которые, как правило, будут относиться к квантификации предикатов.

### Квантификация многомерных предикатов

Пусть дан предикат $P(x,y)$, где $x,y \in D = \{a_1,a_2,a_3,a_4,a_5\}$, и его таблица истинности. Заполним (составим) таблицу истинности предикатов $\forall x\ P(x,y)$, $\exists x\ P(x,y)$, $\forall y\ P(x,y)$, $\exists y\ P(x,y)$.

| $x\backslash y$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
|---|---|---|---|---|---|
| $a_1$ | 0 | 0 | 1 | 0 | 1 |
| $a_2$ | 0 | 1 | 1 | 0 | 1 |
| $a_3$ | 0 | 1 | 1 | 1 | 1 |
| $a_4$ | 0 | 0 | 1 | 1 | 1 |
| $a_5$ | 0 | 1 | 1 | 0 | 1 |

| $x$ | $\forall y\ P(x,y)$ | $\exists y\ P(x,y)$ |
|---|---|---|
| $a_1$ | 0 | 1 |
| $a_2$ | 0 | 1 |
| $a_3$ | 0 | 1 |
| $a_4$ | 0 | 1 |
| $a_5$ | 0 | 1 |

| $y$ | $a_1$ | $a_2$ | $a_3$ | $a_4$ | $a_5$ |
|---|---|---|---|---|---|
| $\forall x\ P(x,y)$ | 0 | 0 | 1 | 0 | 1 |
| $\exists x\ P(x,y)$ | 0 | 1 | 1 | 1 | 1 |

### Перестановка кванторов

1. $\forall x\,\forall y\ P(x,y) = \forall y\,\forall x\ P(x,y)$.
2. $\exists x\,\exists y\ P(x,y) = \exists y\,\exists x\ P(x,y)$.
3. $\exists x\,\forall y\ P(x,y) \supset \forall y\,\exists x\ P(x,y)$.
4. $\exists y\,\forall x\ P(x,y) \supset \forall x\,\exists y\ P(x,y)$.

### Замена кванторов друг на друга, законы де Моргана

1. $\neg\forall x\ P(x) = \exists x\ \neg P(x)$;
   $\forall x\ P(x) = \neg\exists x\ \neg P(x)$.
2. $\neg\exists x\ P(x) = \forall x\ \neg P(x)$;
   $\exists x\ P(x) = \neg\forall x\ \neg P(x)$.

### Вынесение кванторов за скобки

1. $\forall x\,P(x) \wedge \forall x\,Q(x) = \forall x\,(P(x) \wedge Q(x))$.
2. $\exists x\,P(x) \vee \exists x\,Q(x) = \exists x\,(P(x) \vee Q(x))$.
3. $\forall x\,P(x) \vee \forall x\,Q(x) \neq \forall x\,(P(x) \vee Q(x))$.
4. $\exists x\,P(x) \wedge \exists x\,Q(x) \neq \exists x\,(P(x) \wedge Q(x))$.
5. $\forall x\,P(x) \vee C = \forall x\,(P(x) \vee C)$, $C$ — const.
6. $\exists x\,P(x) \wedge C = \exists x\,(P(x) \wedge C)$, $C$ — const.

Доказательство 5. $\forall x\ P(x) \vee C = \forall x\ (P(x) \vee C)$, где $C$ — const. Пусть множество предметных переменных $x \in D = \{a_1, a_2\}$.

$$\begin{aligned}
\forall x\ P(x) \vee C &= P(a_1)P(a_2) \vee C.\\
\forall x\ (P(x) \vee C) &= (P(a_1) \vee C)(P(a_2) \vee C) =\\
&= P(a_1)P(a_2) \vee P(a_1)C \vee CP(a_2) \vee C =\\
&= P(a_1)P(a_2) \vee C(P(a_1) \vee P(a_2) \vee 1) =\\
&= P(a_1)P(a_2) \vee C. \qquad \square
\end{aligned}$$

Замечание. *Применение свойства* 5:

$$\forall x\, P(x) \vee \forall x\ Q(x) = \forall x\, P(x) \vee \underbrace{\forall y\ Q(y)}_{C} = \forall x\,(P(x) \vee C) =$$

$$= \forall x\,(\underbrace{P(x)}_{C} \vee \forall y\, Q(y)) = \forall x\,(C \vee \forall y\ Q(y)) =$$

$$= \forall x\,(\forall y\ (C \vee Q(y)) = \forall x\,(\forall y\ (P(x) \vee Q(y)) =$$

$$= \forall x\ \forall y\,(P(x) \vee Q(y)).$$

### Вынесение кванторов за знак отрицания

При вынесении квантора за знак отрицания квантор меняется на двойственный. В основе данного правила лежат законы де Моргана. Приведем пример для этого свойства:

$$\begin{aligned}
&\neg\forall x\exists y\exists z\forall w\ \exists u\ \ P(x,y,z,w,u) =\\
=\ &\exists x\neg\exists y\exists z\forall w\ \exists u\ \ P(x,y,z,w,u) =\\
=\ &\exists x\forall y\neg\exists z\forall w\ \exists u\ \ P(x,y,z,w,u) = \ldots\\
=\ &\exists x\forall y\forall z\exists w\ \forall u\ \ \neg P(x,y,z,w,u).
\end{aligned}$$

### Приведение формул к предваренной нормальной форме

*Предваренной нормальной формой* (ПНФ) называется формула представления предиката, в которой все кванторы стоят вначале формулы (кванторная приставка, префикс), а бескванторная часть (ядро) приведена к некоторой стандартной форме. В качестве такой формы, как правило, принимается ДНФ или КНФ. Предваренная нормальная форма используется в *автоматических* доказательствах и выводах. Приведенное ядро предиката к стандартному виду позволяет оценивать его доказуемость посредством раздельной оценки доказуемости его составных частей — слагаемых в ДНФ или множителей в КНФ.

Задача. Привести $\exists x\forall y\ P(x,y) \supset \neg\forall x\ \exists y\ Q(x,y)$ к ПНФ.

Р е ш е н и е. Свойство импликации $A \supset B = \neg A \vee B$. Тогда

$$\begin{aligned} & \exists x \forall y \quad P(x,y) \supset \neg \forall x \ \exists y \quad Q(x,y) = \\ = \quad & \neg \exists x \forall y \quad P(x,y) \vee \neg \forall x \ \exists y \quad Q(x,y) = \\ = \quad & \forall x \exists y \neg P(x,y) \vee \quad \exists x \ \forall y \ \neg Q(x,y) = \\ = \quad & \forall x \exists y \neg P(x,y) \vee \quad \exists z \ \forall w \neg Q(z,w) = \\ = \forall x \exists y \ & \exists z \ \forall w \ \ (\neg P(x,y) \vee \neg Q(z,w)\,). \end{aligned}$$

### 1.6.8. Формализация некоторых отношений средствами узкого исчисления предикатов

Язык узкого исчисления предикатов (1.6.4):

1) многоместные двузначные предикаты;
2) предметные переменные предикатов;
3) кванторы для предметных переменных;
4) логические булевы операции.

П р и м е р. Установим связь следующих отношений:

Мать$(x,y)$ — «*$x$ есть мать $y$*».
Отец$(x,y)$ — «*$x$ есть отец $y$*».
Дед$(x,y)$ — «*$x$ есть дед $y$*».
«Дед$(x,y) = \exists z$ Отец$(x,z) \wedge (\,$Отец$(z,y)) \vee$ Мать$(z,y)\,)$».

#### Свойства бинарных отношений

Пусть $P(x,y)$ — некоторое бинарное отношение (предикат), заданное на множестве $x, y \in D$ предметных переменных.

1. *Рефлексивность*: $\forall x \ \ P(x,x)$.
   Каждый элемент множества $x \in D$ сам к себе находится в отношении $P(x,x)$. *Например*, для отношения чисел $P(x,y) =$ «*$x$ равен $y$*» условие $\forall x \ \ P(x,x)$ выполняется, так как «*$x$ равен $x$*».

2. *Симметричность*: $\forall x \ \forall y \, P(x,y) \supset P(y,x)$.
   Если отношение применимо к двум предметам, то — независимо от их порядка. *Например*, отношение $P(x,y) =$ «*Прямая $x$ параллельна прямой $y$*» — симметрично.

3. *Транзитивность*: $\forall x \forall y \forall z \, P(x,y) \wedge P(y,z) \supset P(x,z)$.
   Переносимость отношений между двумя парами элементов $(x,y)$ и $(y,z)$ на третью пару $(x,z)$.

4. *Антирефлексивность*: $\forall x \ \neg P(x,x)$. *Например*, отношение $P(x,y) =$«$x > y$» для чисел является антирефлексивным, так как условие $P(x,x) =$«$x > x$» не может быть выполнено.

5. *Антисимметричность*: $\forall x\,\forall y\,P(x,y) \wedge P(y,x) \supset r(x,y)$, где $r(x,y) =$ «$x$ равно $y$». Назначение отношения антисимметричности — обеспечить невозможность одновременного выполнения условий $P(x,y)$ и $P(y,x)$ для переменных $x \neq y$. Это возможно достичь выполнением отношения $P(x,y) \supset \neg P(y,x)$, противоположного отношению *симметричности*. Однако оно может нарушаться для равных значений переменных $x$ и $y$. Именно это потребовало модификации отношения антисимметричности. *Например*, отношение $P(x,y) =$«$x \geqslant y$» для чисел является антисимметричным.

6. *Трихотомия*: $\forall x\forall y\,P(x,y) \vee P(y,x) = \forall x\forall y P(x,y) \wedge \wedge\neg P(y,x) \vee \neg P(x,y) \wedge P(y,x) \vee P(x,y) \wedge P(y,x)$. Предикат условия трихотомии предполагает обязательное выполнение отношений $P(x,y)$ или $P(y,x)$ для каждой пары элементов $x, y \in D$. Возможно и одновременное выполнение условий $P(x,y)$ и $P(y,x)$.

**Отношение эквивалентности**

Бинарное отношение $P(x,y)$, где $x, y \in D$, называется *отношением эквивалентности*, если оно *рефлексивно*, *симметрично* и *транзитивно*. Таким образом, предикат отношения эквивалентности — это конъюнкция трех отношений:

$$\begin{aligned}\forall x\forall y\forall z\ & P(x,x)\\ & \&\ P(x,y) \supset P(y,x)\\ & \&\ P(x,y) \wedge P(y,z) \supset P(x,z).\end{aligned}$$

Примером отношения эквивалентности может служить классификация объектов. Отношение $P(x,y)$ для элементов $x, y \in D$ выполняется, если они имеют выделенные общие признаки. Объекты с общими признаками объединяются в отдельный класс. Элементы $x, y \in D$, для которых выполняется отношение $P(x,y)$, называются *эквивалентными*.

Определение. *Классом эквивалентности*, порожденным элементом $x_i \in D$, называется подмножество элементов $D_i = \{x \mid P(x_i, x) \wedge x \in D\}$, эквивалентных данному $x_i$.

Теорема 1.6.1. *Пусть $P(x,y)$ — отношение эквивалентности, где $x, y \in D$, и $D_i$ — классы эквивалентности, составленные для каждого элемента $x_i \in D$. Тогда выполняются следующие условия*:

1) *для каждого $x_i \in D$ верно, что $x_i \in D_i$ принадлежит порожденному им классу эквивалентности*;

2) *классы, порожденные эквивалентными элементами* $x_i, x_j$, *совпадают* $D_i = D_j$;

3) *классы эквивалентности совпадают* $D_i = D_j$, *если их пересечение непусто* $D_i \cap D_j \neq \varnothing$.

Доказательство.

1). Пусть класс $D_i$ порожден элементом $x_i \in D$. Тогда $D_i = \{x \,|\, P(x_i, x) \wedge x \in D\}$. Рефлексивность отношения $P(x_i, x_i)$ позволяет заключить, что $x_i \in D_i$.

2). Пусть $z \in D_j$ — произвольный элемент класса, порожденного элементом $x_j$. Тогда верно отношение $P(x_j, z)$. По условию имеем эквивалентность элементов $P(x_i, x_j)$. Из транзитивности отношения заключаем эквивалентность $P(x_i, z)$. Следовательно, $z \in D_i$ и, значит, $D_j \subseteq D_i$. Подобным образом проверяется включение в обратную сторону $D_j \supseteq D_i$. Таким образом, классы эквивалентных элементов совпадают: $D_j = D_i$.

3). Пусть $z \in D_i \cap D_j \neq \varnothing$ — непустое пересечение. Тогда выполняются $P(x_i, z)$ и $P(x_j, z)$. Из симметричности отношения имеем $P(z, x_j)$, а из транзитивности $P(x_i, x_j)$. Таким образом, элементы $x_i$ и $x_j$ — эквивалентные. По условию пункта 2), классы, порожденные такими элементами, совпадают. □

Следствие. *Пусть* $P(x, y)$ *— отношение эквивалентности, где* $x, y \in D$. *Тогда множество* $D$ *разбивается на классы эквивалентности* $D = D_{k_1} \cup D_{k_2} \cup \ldots \cup D_{k_m}$, *где* $D_{k_i} \cap D_{k_j} = \varnothing$, *если* $k_i \neq k_j$.

Вследствие попарного непересечения $D_{k_i}$, разложение называется *прямым*. Совокупность множеств-классов $D_{k_i}$, где $i = 1, 2, \ldots, m$, называется *фактормножеством* множества $D$ относительно отношения $P(x, y)$ и обозначается как $D/P$.

Пример. Пусть $D$ — множество людей. Определим на $D$ бинарное отношение $P(x, y) =$ «$x$ и $y$ родились в одной и той же стране», где $x, y \in D$. Отношение $P(x, y)$ является отношением эквивалентности. Классы эквивалентности составляют люди, родившиеся в одной стране.

Пример. Пусть $D$ — множество студентов одного города. Определим на $D$ бинарное отношение $P(x, y) =$ «$x$ и $y$ учатся в одном вузе», где $x, y \in D$. Отношение $P(x, y)$ будет отношением эквивалентности, если ни один из студентов множества $D$ не является одновременно слушателем нескольких вузов. В этом случае классы эквивалентности будут составлять студенты одного вуза.

### Отношение частичного порядка

Бинарное отношение $P(x,y)$, где $x,y \in D$, называется *отношением частичного порядка*, если для него выполняется *рефлексивность*, *антисимметричность* и *транзитивность*. Для отношения антисимметричности характерно то, что не все переменные $x,y$ должны быть охвачены выполнением отношения $P(x,y)$. Антисимметричность не допускает лишь одновременного выполнения отношений $P(x,y)$ и $P(y,x)$. Предикат отношения частичного порядка — это конъюнкция трех отношений:

$$\begin{aligned}\forall x\forall y\forall z\ & P(x,x)\\ & \&\ P(x,y)\wedge P(y,x)\supset (x=y)\\ & \&\ P(x,y)\wedge P(y,z)\supset P(x,z).\end{aligned}$$

Множество $D$, на котором определено такое отношение $P(x,y)$, называется *частично упорядоченным*. Приведем пример этого отношения. Пусть элементами множества $D$ являются подмножества из элементов множества $\{1,2,3\}$. Тогда $D=\{\varnothing\ ,\{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\},\}$. Определим $P(x,y)=$ «$x\supseteq y$» — отношение включения множеств. Данное отношение является отношением частичного порядка.

### Отношение полного порядка, линейный порядок

Бинарное отношение $P(x,y)$, где $x,y \in D$, называется *отношением полного порядка*, если для него выполняется *рефлексивность*, *антисимметричность*, *транзитивность* и *трихотомия*. Отношение полного порядка — это отношение частичного порядка, для которого выполняется *трихотомия*. Трихотомия предполагает обязательное выполнение отношения $P(x,y)$ или $P(y,x)$ для каждой пары элементов $x,y \in D$. Предикат линейного порядка — это конъюнкция четырех отношений:

$$\begin{aligned}\forall x\forall y\forall z\ & P(x,x)\\ & \&\ P(x,y)\wedge P(y,x)\supset (x=y)\\ & \&\ P(x,y)\wedge P(y,z)\supset P(x,z)\\ & \&\ P(x,y)\wedge \neg P(y,x)\ \vee\ \neg P(x,y)\wedge P(y,x)\ \vee\ P(x,y)\wedge P(y,x).\end{aligned}$$

Множество $D$, на котором определено такое отношение $P(x,y)$, называется *линейно упорядоченным*. Примером такого отношения может служить сравнение чисел с равенством: $P(x,y)=$ «$x\geqslant y$».

### Изоморфизм частично упорядоченных множеств

О п р е д е л е н и е. Пусть $P(x,y)$ и $Q(x,y)$ — отношения частичного порядка, определенные на множествах, соответственно, $D_P$ и $D_Q$. Такие множества называются *частично*

*упорядоченными*. Изоморфизмом частично упорядоченных множеств $D_P$ и $D_Q$ называется взаимно однозначное соответствие $\varphi : D_P \to D_Q$, сохраняющее отношение частичного порядка: $\forall x, y \in D_P\, P(x,y) \supset Q(\varphi(x), \varphi(y))$, и обратно. Очевидно, что в том случае, когда природа элементов не играет роли, изоморфные множества можно считать тождественными.

Теорема 1.6.2. *Всякое частично упорядоченное множество $D$ изоморфно некоторой системе подмножеств $M(D)$ множества $D$, частично упорядоченной по включению* $(\subseteq)$.

Доказательство. По условию имеем два отношения частичного порядка $P(a,b)$, где $a, b \in D$, и $Q(A,B)$, где $A, B \in M(D)$, и $Q(A,B) =$ «$A \subseteq B$» — отношение включения. Множество $A \in M(D)$ определяется как $A = \{x \in D \,|P(x,a) \wedge a \in D\}$. Говорят, что $a \in D$ соответствует $A \in M(D)$, и обратно.

Определим соответствие $\varphi : D \to M(D)$, где $\varphi(a) = A$ и $a \in D$ соответствует $A \in M(D)$. Покажем, что это — изоморфное отображение.

Докажем, что это — взаимно однозначное соответствие $\varphi$. По построению системы подмножеств $M(D)$ для каждого $A \in M(D)$ существует прообраз $a \in D$. Таким образом, соответствие $\varphi$ является отображением «на».

Покажем, что для произвольных $a \neq b \in D$ соответствующие им $A, B \in M(D)$ не совпадают, т. е. $A \neq B$. Предположим, что $A = B$, будем иметь $a \in A$ вследствие *рефлексивности* отношения $P(a,a)$, где $A = \{x \in D \,|P(x,a) \wedge a \in D\}$. Так как $A = B$, то и $a \in B$. В свою очередь $B = \{x \in D \,|P(x,b) \wedge b \in D\}$, а значит, верно $P(a,b)$. Подобным образом можно установить, что верно и $P(b,a)$. Таким образом, имеем $P(a,b)$ и $P(b,a)$. В силу *антисимметричности* отношения $P(x,y)$ можем заключить, что $a = b$. Это противоречит условию $a \neq b$ и доказывает, что соответствие $\varphi$ является взаимно однозначным.

Докажем сохранение отображением $\varphi$ отношения частичного порядка. Пусть выполняется $P(a,b)$. Тогда для каждого $x \in D$, для которого верно $P(x,a)$, верным будет и $P(x,b)$ вследствие транзитивности отношения $P(x,y)$. Имеем $A = \{x \in D \,|P(x,a) \wedge a \in D\}$ и $B = \{x \in D \,|P(x,b) \wedge b \in D\}$. Отсюда $A \subseteq B$, т. е. $Q(A,B) = Q(\varphi(a)\varphi(b))$. Итак, из $P(a,b)$ следует сохранение операций $Q(\varphi(a)\varphi(b))$. И обратно. Пусть $Q(A,B)$, т. е. $A \subseteq B$. Из рефлексивности $P(a,a)$ имеем $a \in A$, а значит, и $a \in B$. А так как $B = \{x \in D \,|P(x,b) \wedge b \in D\}$, то заключаем, что выполняется и $P(a,b)$. Сохранение отношения частичного порядка доказано и доказана теорема. $\square$

Глава 2

# КОМБИНАТОРНЫЕ СХЕМЫ

В этой главе будет сделан обзор комбинаторных формул, наиболее важных для вычислительных задач. Мы не ставим себе целью сделать этот обзор всеобъемлющим, а хотим сосредоточить внимание читателя на таких формулах, которые он мог бы недооценить или даже совсем не заметить. Заинтересованному читателю рекомендуется обратиться к специальной литературе.

Введем следующие обозначения. Множества будем обозначать заглавными буквами. Множества состоят из элементов, которые будем обозначать малыми буквами. Так, запись $a \in A$ означает, что элемент $a$ принадлежит множеству $A$. Такие множества будем изображать перечислением элементов, заключая их в фигурные скобки. Например, $\{a, b, x, y\}$. Количество элементов в множестве называется *мощностью* и записывается как $|A|$.

Пусть имеются два множества, $A$ и $B$. Рассмотрим все пары элементов при условии, что первый элемент берется из множества $A$, а второй — из множества $B$. Полученное таким образом множество называется *прямым произведением* $A \times B$ множеств $A$ и $B$. Напомним некоторые операции над множествами, которыми время от времени будем пользоваться:

$A \times B = \{(a, b) \mid a \in A,\ b \in B\}$ — прямое произведение множеств;
$A \cup B = \{x \mid x \in A \vee x \in B\}$ — объединение множеств;
$A \cap B = \{x \mid x \in A \wedge x \in B\}$ — пересечение множеств;
$A \setminus B = \{x \mid x \in A \wedge x \notin B\}$ — разность множеств;
$U$ — универсальное множество;
$\varnothing$ — пустое множество;
$\overline{A} = U \setminus A = \{x \mid x \notin A\}$ — дополнение множества.

## § 2.1. Правило суммы

Пусть $A$ и $B$ — конечные множества такие, что $A \cap B = \varnothing$, $|A| = m$ и $|B| = n$. Тогда $|A \cup B| = m + n$.

*Интерпретация*. Если элемент $a \in A$ можно выбрать $m$ способами, а элемент $b \in B$ — $n$ способами, то выбор элемента $x \in A \cup B$ можно осуществить $m + n$ способами. Пусть $X_1, X_2, \ldots, X_k$ — попарно непересекающиеся множества,

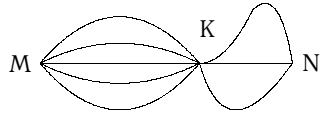
$X_i \cap X_j = \varnothing$, где $i \neq j$. Тогда, очевидно, выполняется равенство $|\bigcup\limits_{i=1}^{k} X_i| = \sum\limits_{i=1}^{k} |X_i|$.

## § 2.2. Правило прямого произведения

Пусть $A$ и $B$ — конечные множества, $|A| = m$ и $|B| = n$, тогда $|A \times B| = m \cdot n$.

*Интерпретация*. Если элемент $a \in A$ можно выбрать $m$ способами и если после каждого такого выбора элемент $b \in B$ можно выбрать $n$ способами, то выбор пары $(a, b) \in A \times B$ в указанном порядке можно осуществить $|A \times B| = m \cdot n$ способами. В этом случае говорят, что выбор элементов множества $A$ не зависит от способа выбора элементов множества $B$. Пусть теперь $X_1, X_2, \ldots, X_k$ — произвольные множества, $|X_i| = n_i$, $i = \overline{1,k}$. Тогда $|X_1 \times X_2 \times \ldots \times X_k| = |\{(x_1, x_2, \ldots, x_k) | x_i \in X_i, i = \overline{1,k}\}| = n_1 \cdot n_2 \cdot \ldots \cdot n_k$.

З а д а ч а. Найти число маршрутов из пункта $M$ в пункт $N$ через пункт $K$. Из $M$ в $K$ ведут 5 дорог, из $K$ в $N$ — 3 дороги.




Р е ш е н и е. Введем два множества: $S = \{s_1, s_2, s_3, s_4, s_5\}$ — дороги из $M$ в $K$, $T = \{t_1, t_2, t_3\}$ — дороги из $K$ в $N$. Теперь дорогу из $M$ в $N$ можно представить парой $(s_i, t_j)$, где $i = \overline{1,5}$; $j = \overline{1,3}$. Значит, $S \times T$ — это множество всех дорог из $M$ в $N$, количество которых равно $|S \times T| = 5 \cdot 3 = 15$.

## § 2.3. Размещения с повторениями

Задача формулируется следующим образом. Имеются предметы $n$ различных видов: $a_1, a_2, \ldots, a_n$. Из них составляют всевозможные расстановки длины $k$. Например, $a_1a_2a_3a_3a_4a_3a_2a_1$ — расстановка длины 8. Такие расстановки называются *размещениями с повторениями* из $n$ по $k$ (элементы одного вида могут повторяться). Найдем общее число расстановок, среди которых две расстановки считаются различными, если они отличаются друг от друга или видом входящих в них предметов, или порядком этих предметов. При составлении указанных расстановок длины $k$ на каждое место можно поставить предмет любого вида. Рассмотрим множества $X_1, X_2, \ldots, X_k$ такие, что $X_1 = X_2 = \ldots = X_k = \{a_1, a_2, \ldots, a_n\}$. Тогда все размещения с по-

вторениями составят множество $X_1 \times X_2 \times \ldots \times X_k$. По правилу прямого произведения получаем, что общее число размещений с повторениями из $n$ по $k$ равно $|X_1 \times X_2 \times \ldots \times X_k| = n^k$.

Задача. Найти количество всех пятизначных чисел.

Решение. Введем пять множеств: $A_1 = \{1, 2, \ldots, 9\}$, $A_2 = A_3 = A_4 = A_5 = \{0, 1, 2, \ldots, 9\}$. Тогда все пятизначные числа составят прямое произведение указанных множеств $A_1 \times A_2 \times A_3 \times A_4 \times A_5$. Согласно правилу прямого произведения, количество элементов в множестве $A_1 \times A_2 \times A_3 \times A_4 \times A_5$ равно $9 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 90000$.

## § 2.4. Размещения без повторений

Имеются $n$ различных предметов: $a_1, a_2, \ldots, a_n$. Сколько из них можно составить расстановок длины $k$? Две расстановки считаются различными, если они отличаются видом входящих в них элементов или порядком их в расстановке. Такие расстановки называются *размещениями без повторений*, а их число обозначают $A_n^k$. При составлении данных расстановок на первое место можно поставить любой из имеющихся $n$ предметов. На второе место теперь можно поставить только любой из $n-1$ оставшихся. И, наконец, на $k$-е место — любой из $n-k+1$ оставшихся предметов. По правилу прямого произведения получаем, что общее число размещений без повторений из $n$ по $k$ равно $A_n^k = n \cdot (n-1) \cdot \ldots \cdot (n-k+1) = n!/(n-k)!$. Напомним, что $n! = n \cdot (n-1) \cdot \ldots \cdot 1$ и $0! = 1$.

Задача. В хоккейном турнире участвуют 17 команд. Разыгрываются золотые, серебряные и бронзовые медали. Сколькими способами могут быть распределены медали?

Решение. 17 команд претендуют на 3 места. Тогда тройку призеров можно выбрать способами $A_{17}^3 = 17 \cdot 16 \cdot 15 = 4080$.

## § 2.5. Перестановки

При составлении размещений без повторений из $n$ по $k$ мы получали расстановки, отличающиеся друг от друга либо составом, либо порядком элементов. Но если брать расстановки, которые включают все $n$ элементов, то они могут отличаться друг от друга лишь порядком входящих в них элементов. Такие расстановки называются *перестановками* из $n$ элементов, а их число обозначается $P_n$. Следовательно, число перестановок

равно $P_n = A_n^n = n!$. Перестановки $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ элементов $1, 2, \ldots, n$ записывают и в матричной форме $\pi = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix}$, где верхняя строка — это порядковые номера $1, 2, \ldots, n$ позиций элементов в перестановке; нижняя строка — тот же набор чисел $1, 2, \ldots, n$, взятых в каком-либо порядке; $\pi_j$ — номер элемента на $j$-м месте перестановки. Порядок столбцов в перестановках, записанных в матричной форме, не является существенным, так как в этом случае номер позиции каждого элемента в перестановке указывается явно в нижней строке. Например, перестановка $(3, 2, 4, 1)$ из четырех элементов может быть записана как $\begin{pmatrix} 1\,2\,3\,4 \\ 3\,2\,4\,1 \end{pmatrix}$, $\begin{pmatrix} 3\,1\,4\,2 \\ 4\,3\,1\,2 \end{pmatrix}$, $\begin{pmatrix} 2\,1\,4\,3 \\ 2\,3\,1\,4 \end{pmatrix}$ и т. д.

Задача. Сколькими способами можно расположить на шахматной доске 8 ладей, чтобы они «не били» друг друга?

Решение. Условие «не могли бить» означает, что на каждой горизонтали и вертикали может стоять лишь одна ладья. Ввиду этого, каждому расположению ладей на доске соответствует перестановка $\pi = \begin{pmatrix} 1 & 2 & \ldots 8 \\ \pi_1 & \pi_2 & \ldots \pi_8 \end{pmatrix}$. Верхняя строка перестановки — это номера горизонталей, нижняя — вертикалей, пересечение которых определяет положение ладей на доске. Следовательно, число расстановок равно числу перестановок $P_8 = 8!$.

## § 2.6. Сочетания

В тех случаях, когда нас не интересует порядок элементов в расстановке, а интересует лишь ее состав, говорят о сочетаниях. *Сочетаниями* из $n$ различных элементов по $k$ называют все возможные расстановки длины $k$, образованные из этих элементов и отличающиеся друг от друга составом, но не порядком элементов. Общее число сочетаний обозначают через $C_n^k$ или $\binom{n}{k}$. Определим это число. Составим все сочетания из $n$ по $k$. Затем переставим в каждом сочетании элементы всеми возможными способами. Теперь мы получили расстановки, отличающиеся либо составом, либо порядком, т. е. это все размещения без повторений из $n$ по $k$. Их число равно $A_n^k$. Учитывая, что каждое сочетание дает $k!$ размещений, по правилу произведения можно записать $C_n^k \times k! = A_n^k$ . Тогда $C_n^k = \dfrac{n \cdot (n-1) \ldots (n-k+1)}{k!}$ или $C_n^k = \dfrac{n!}{k!(n-k)!}$ и $C_n^k = C_n^{n-k}$.

| 1 | 2 | … | n |
|---|---|---|---|
| 2 | | | |
| ⋮ | | | |
| m | | | |

Задача. Сколько различных прямоугольников можно вырезать из клеток доски, размер которой $m \times n$?

Решение. Прямоугольник однозначно определяется положением его сторон. Горизонтальные стороны могут занимать любое из $m+1$ положения. Тогда число способов их выбора равно $C^2_{m+1}$. Вертикальные стороны можно выбрать $C^2_{n+1}$ способами. По правилу прямого произведения заключаем, что количество прямоугольников равно $C^2_{m+1} \times C^2_{n+1}$.

## § 2.7. Сочетания с повторениями

Имеются предметы $n$ различных видов. Число элементов каждого вида неограниченно. Сколько существует расстановок длины $k$, если не принимать во внимание порядок элементов? Такие расстановки называют *сочетаниями с повторениями*, количество и обозначение которых следующее: $\overline{C}^k_n = C^{n-1}_{n+k-1} = C^k_{n+k-1}$. Выведем данную формулу. Пусть $a, b, c, \ldots, d$ — это исходные различные типы элементов, количество которых $n$. Рассмотрим произвольное сочетание с повторениями $cbbcaccda \ldots ddaccbbb$ из данных типов элементов. Так как порядок элементов в сочетаниях не учитывается, то расстановку можно записать и так: $aa \ldots a|bb \ldots b|cc \ldots c| \ldots |dd \ldots d$, где элементы каждого из типов упорядочены и завершаются вертикальной чертой, за исключением последней серии элементов. Длина такой расстановки с учетом вертикальных линий составляет $k + (n-1) = n + k - 1$, где $k$ — количество элементов в расстановке; $n-1$ — число вертикальных линий. Очевидно, что любую такую расстановку можно задать выбором $n-1$ места из $n+k-1$ мест для положений вертикальных линий. Это можно сделать $C^{n-1}_{n+k-1}$ способами. Промежуточные места между линиями заполняются соответствующими типами элементов.

Задача. Трое ребят собрали в саду 63 яблока. Сколькими способами они могут их разделить между собой?

Решение. Поставим в соответствие каждому делению яблок между ребятами сочетание с повторениями следующим способом. Типами элементов в нашем случае будут ребята. Таким образом, имеем три типа элементов $a, b, c$ $(n = 3)$, из которых

предстоит составить все различные расстановки длины $k = 63$. Наличие в расстановке какого-либо из элементов $a, b, c$ отвечает принадлежности данного яблока соответствующему мальчику. Порядок элементов в такой расстановке не играет роли. При делении яблок между ребятами не важно, какое из них попадет тому или иному мальчику. Тогда число способов разделить яблоки между ребятами равно $\overline{C}_3^{63} = C_{3+63-1}^{3-1} = 2080$.

З а д а ч а. Найти количество целочисленных решений системы $x_1 + x_2 + \ldots + x_n = k$, $k \geqslant 0$, $x_i \geqslant 0$, $i = 1, 2, \ldots, n$; $n \geqslant 1$.

Р е ш е н и е. Рассмотрим следующую интерпретацию решения уравнения. Каждое значение $x_i = 1_i + 1_i + \ldots + 1_i$ представим как сумму единиц, количество которых $x_i$. Индекс у $1_i$ отмечает ее принадлежность к разложению числа $x_i$. Таким образом, мы ввели $n$ типов различных элементов $\{1_1, 1_2, \ldots, 1_n\}$, значение каждого из них равно единице. Теперь любое решение исходного уравнения можно представить как сумму, составленную из $k$ произвольных единиц множества $\{1_1, 1_2, \ldots, 1_n\}$. Суммируя подобные единицы $1_i$ с одинаковыми индексами, можно составить соответствующие слагаемые $x_i$ решения исходного уравнения. Данное соответствие является взаимно однозначным, откуда и следует, что число решений системы равно числу сочетаний с повторениями: $\overline{C}_n^k = C_{n+k-1}^{n-1} = C_{n+k-1}^k$.

## § 2.8. Перестановки с повторениями, мультимножества

Задача формулируется следующим образом. Имеются предметы $k$ различных видов. Сколько существует перестановок из $n_1$ элементов первого типа, $n_2$ элементов второго типа и т. д., $n_k$ элементов $k$-го типа? Рассмотрим, например, *мультимножество* $M = \{a, a, a, b, b, c, d, d, d, d\}$, в котором содержатся 3 элемента $a$, 2 элемента $b$, 1 элемент $c$ и 4 элемента $d$. Мультимножество — это то же самое, что и множество, но в нем могут содержаться одинаковые элементы. Повторения элементов можно указать и другим способом: $M = \{3a, 2b, 1c, 4d\}$. Таким образом, искомые перестановки с повторениями — это перестановки элементов мультимножества. Если бы мы рассматривали все элементы множества $M$ как различные, обозначив их $M = \{a_1, a_2, a_3, b_1, b_2, c_1, d_1, d_2, d_3, d_4\}$, то получили бы 10! перестановок, но после отбрасывания индексов многие из них

оказались бы одинаковыми. Фактически каждая перестановка множества $M$ встретилась бы ровно $3!\,2!\,1!\,4!$ раз, поскольку в любой перестановке индексы при буквах $a$ можно расставить $3!$ способами, при $b$ — $2!$ способами, при $c$ — одним способом, а при $d$ — соответственно $4!$ способами. Поэтому число перестановок множества $M$ равно $10!/(3!\,2!\,1!\,4!)$. В применении к общему случаю те же рассуждения показывают, что число перестановок любого мультимножества (перестановки с повторениями) равно полиномиальному коэффициенту

$$P(n_1, n_2, \ldots, n_k) = \frac{n}{n_1!\,n_2! \cdot \ldots \cdot n_k!},$$

где $n = n_1 + n_2 + \ldots + n_k$ общее число элементов.

Перестановки с повторениями имеют тесную связь с сочетаниями. Определим количество этих перестановок следующим образом. Из всех $n$ мест перестановки $n_1$ место занимают элементы первого типа. Выбор мест для них можно сделать $C_n^{n_1}$ способами. Из оставшихся $n - n_1$ мест элементы второго типа занимают $n_2$ места, которые можно выбрать $C_{n-n_1}^{n_2}$ способами. Те же рассуждения показывают, что элементы $k$-го типа можно расположить в перестановке $C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k}$ способами. Согласно правилу прямого произведения, число перестановок с повторениями равно

$$P(n_1, n_2, \ldots, n_k) = C_n^{n_1} C_{n-n_1}^{n_2} \cdot \ldots \cdot C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = \frac{n}{n_1!\,n_2! \cdot \ldots \cdot n_k!}.$$

Задача. Сколько существует различных перестановок из букв слова «Уссури»?

Решение. $P(2\text{у}, 1\text{и}, 1\text{р}, 2\text{с}) = \dfrac{6!}{2!\,1!\,1!\,2!} = 180.$

## § 2.9. Упорядоченные разбиения множества

Подсчитаем число разбиений конечного множества $S$, где $|S| = n$, на $k$ различных подмножеств $S = S_1 \cup S_2 \cup \ldots \cup S_k$, попарно не пересекающихся, $|S_i| = n_i$, $i = 1, 2, \ldots, k$, и $n_1 + n_2 + \ldots + n_k = n$. Последовательность различных $S_1, S_2, \ldots, S_k$ рассматривается как упорядоченная последовательность подмножеств. При формировании упорядоченной $S_1, S_2, \ldots, S_k$ последовательности на первое место подмножество $S_1$ можно выбрать $C_n^{n_1}$ способами, на второе место подмножество $S_2$ можно выбрать

из оставшихся $n - n_1$ элементов $C_{n-n_1}^{n_2}$ способами и т. д., на последнее место множество $S_k$ можно выбрать из оставшихся $n - n_1 - n_2 - \ldots - n_{k-1}$ элементов $C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k}$ способами. По правилу прямого произведения получаем, что общее число упорядоченных разбиений множества $S$ на $k$ подмножеств равно

$$C_n^{n_1} C_{n-n_1}^{n_2} \cdot \ldots \cdot C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = \frac{n}{n_1!\, n_2! \cdot \ldots \cdot n_k!},$$

что совпадает с числом $P(n_1, n_2, \ldots, n_k)$ перестановок с повторениями.

Замечание 1. Установим взаимно однозначное соответствие между упорядоченными разбиениями множества и перестановками с повторениями. Каждой перестановке с повторениями можно поставить в соответствие упорядоченное разбиение множества номеров элементов $S = \{1, 2, \ldots, n\}$ в перестановке на подмножества $S_1, S_2, \ldots, S_k$, где $S_i$ — множество номеров элементов $i$-го типа в перестановке. Очевидно, что данное соответствие между перестановками с повторениями и разбиениями является взаимно однозначным.

Замечание 2. Упорядоченные разбиения множества $S$ на попарно не пересекающиеся подмножества $S = S_1 \cup S_2 \cup \ldots \cup \cup S_k$, допускают интерпретацию в терминах «корзин» и «шаров». Обозначим элементы исходного множества $|S| = n$ «шарами». Под разбиением исходного множества, теперь множества шаров, на различные $S_i$ упорядоченные подмножества будем понимать разложение шаров по $k$ различным корзинам (упорядоченные $S_1, S_2, \ldots, S_k$ подмножества): $n_1$ шаров положить в корзину $S_1$, $n_2$ шаров положить в корзину $S_2$ и т. д., $n_k$ шаров положить в корзину $S_k$, где $n_1 + n_2 + \ldots$ $\ldots + n_k = n$. Как установлено, число таких разложений равно $C_n^{n_1} C_{n-n_1}^{n_2} \cdot \ldots \cdot C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = n/(n_1!\, n_2! \cdot \ldots \cdot n_k!)$.

Задача. В студенческой группе, состоящей из 25 человек, при выборе старосты за выдвинутую кандидатуру проголосовали 19 человек, против — 3, воздержались — 3. Сколькими способами может быть проведено такое голосование?

Решение. Имеем три различные корзины: «за», «против», «воздержались», в которые необходимо разложить 25 шаров, соответственно 19 — в первую, 3 — во вторую, 3 — в третью. Количество таких разложений определяется выражением $C_{25}^{19} C_6^3 C_3^3 = 25!/(19!\, 3!\, 3!)$.

## § 2.10. Неупорядоченные разбиения множества

Подсчитаем, сколькими способами можно разбить множество $S$, где $|S| = n$, на подмножества, среди которых для каждого $i = 1, 2, \ldots, n$ имеется $m_i \geqslant 0$ подмножеств с $i$ элементами. Тогда верно, что $\sum_{i=1}^{n} i \cdot m_i = n$. Данное разбиение позволяет представить исходное множество следующим образом:

$$S = \bigcup_{j=1}^{m_1} S_{1_j} \cup \bigcup_{j=1}^{m_2} S_{2_j} \cup \ldots \cup \bigcup_{j=1}^{m_n} S_{n_j} = \bigcup_{i=1}^{n} \bigcup_{j=1}^{m_i} S_{i_j},$$

где $S_{i_j}$ попарно не пересекаются и $|S_{i_1}| = |S_{i_2}| = \ldots = |S_{i_{m_i}}| = i$ для каждого $i = 1, 2, \ldots, n$. Порядок подмножеств в разбиении не является существенным. Так, например, разбиения множества $S = \{1, 2, 3, 4, 5\}$ вида

$\{1,3\},\ \{4\},\ \{2,5\};$
$\{1,3\},\ \{2,5\},\ \{4\};$
$\{2,5\},\ \{1,3\},\ \{4\};$
$\{4\},\ \{1,3\},\ \{2,5\};$

считаются одинаковыми.

Обозначим число неупорядоченных разбиений множества $S$ через $N(m_1, m_2, \ldots, m_n)$. Рассмотрим схему формирования упорядоченных разбиений для представления $n = 1 \cdot m_1 + 2 \cdot m_2 + \ldots + n \cdot m_n$:

$$\underbrace{C_n^1 C_{n-1}^1 C_{n-2}^1 \cdot \ldots \cdot}_{m_1} \underbrace{C_{n-1\cdot m_1}^2 C_{n-1\cdot m_1-1\cdot 2}^2 C_{n-1\cdot m_1-2\cdot 2}^2 \times \ldots \times}_{m_2}$$

$$\underbrace{\times C_{n-1\cdot m_1-2\cdot m_2-\ldots-(n-1)\cdot m_{n-1}}^n C_{n-1\cdot m_1-2\cdot m_2-\ldots-(n-1)\cdot m_{n-1}-1\cdot n}^n \cdot \ldots}_{m_n} =$$

$$= \frac{n!}{\underbrace{1!\,1! \cdot \ldots \cdot 1!}_{m_1} \underbrace{2!\,2! \cdot \ldots \cdot 2!}_{m_2} \cdot \ldots \cdot \underbrace{n!\,n! \cdot \ldots \cdot n!}_{m_n}} =$$

$$= \frac{n!}{(1!)^{m_1} (2!)^{m_2} \cdot \ldots \cdot (n!)^{m_n}}.$$

Воспользуемся интерпретацией формирования упорядоченных разбиений как разложения $n$ различных шаров по различным $1 \cdot m_1 + 2 \cdot m_2 + \ldots + n \cdot m_n$ корзинам так, что в каждую из $m_i$ корзину кладут $i$ шаров. Теперь откажемся от упорядоченности подмножеств в разбиении. Пусть все корзины имеют различное число шаров, такие корзины можно рассматривать как различные

(они отличаются числом шаров). В этом случае упорядоченные и неупорядоченные разложения шаров совпадают. Пусть теперь в разложении существуют корзины с одинаковым количеством шаров. При упорядоченном разложении такие корзины рассматриваются как различные. Однако при неупорядоченном разложении обмен шарами таких корзин можно рассматривать как соответствующую перестановку указанных корзин, что не приводит к новым разложениям. Если количество корзин с одинаковым числом шаров равно $m_i$, то неупорядоченных разложений будет в $m_i!$ меньше, чем упорядоченных. Тогда общее число неупорядоченных разбиений будет в $m_1!\,m_2!\cdot\ldots\cdot m_n!$ раз меньше, чем упорядоченных. Следовательно,

$$N(m_1, m_2, \ldots, m_n) = \frac{n!}{(1!)^{m_1}(2!)^{m_2}\cdot\ldots\cdot(n!)^{m_n}m_1!\,m_2!\cdot\ldots\cdot m_n!}.$$

*Заметим еще раз, что если выполнено упорядоченное разбиение числа $n$ на подмножества различной длины* (*мощности*), *то они совпадают с неупорядоченными разбиениями. В этом случае все* $m_i \in \{0, 1\}$.

Задача. Сколькими способами из группы в 17 человек можно сформировать 6 коалиций по 2 человека и 1 коалицию из 5 человек?

Решение. Требуется разбить множество из 17 человек на непересекающиеся и неупорядоченные группы людей. Отсюда искомое число равно $N(0_1, 6_2, 0_3, 0_4, 1_5, 0_6, 0_7, \ldots, 0_{17}) = 17!/((2!)^6(5!)^1 6!\,1!)$.

Задача. Сколькими способами можно разделить колоду из 36 карт пополам так, чтобы в каждой пачке было по два туза?

Решение. 4 туза можно разбить на $4!/((2!)^2 2!) = 3$ различные коалиции по две карты в каждой (неупорядоченные разбиения), т. е. только 3 способами можно разделить тузы пополам. Далее, каждая половина любого из этих трех разбиений тузов выполняет роль различных двух «корзин», куда необходимо разложить пополам оставшиеся 32 карты. Разложение 32 оставшихся карт уже будет упорядоченным, так как «корзины» различные, число разложений равно $32!/(16!\,16!)$. Согласно правилу прямого произведения, общее число вариантов разделить колоду пополам равно $4!/((2!)^2 2!)\cdot 32!/(16!\,16!)$.

## § 2.11. Полиномиальная формула

Формула

$$(x_1 + x_2 + \ldots + x_k)^n = \sum_{n_1+n_2+\ldots+n_k=n} \frac{n!}{n_1!\, n_2! \cdot \ldots \cdot n_k!} x_1^{n_1} x_2^{n_2} \cdot \ldots \cdot x_k^{n_k} \quad (2.1)$$

называется *полиномиальной*, где суммирование выполняется по всем решениям уравнения $n_1 + n_2 + \ldots + n_k = n$ в целых неотрицательных числах, $n_i \geqslant 0$, $i = 1, 2, \ldots, k$. Для доказательства выполним умножение

$$\underbrace{(x_1 + x_2 + \ldots + x_k) \cdot \ldots \cdot (x_1 + x_2 + \ldots + x_k)}_{n} = (x_1 + x_2 + \ldots + x_k)^n. \quad (2.2)$$

Чтобы привести подобные в полученном выражении, необходимо подсчитать количество слагаемых вида $x_1^{n_1} x_2^{n_2} \cdot \ldots \cdot x_k^{n_k}$ каждого разбиения $n_1 + n_2 + \ldots + n_k = n$. Для получения же слагаемого $x_1^{n_1} x_2^{n_2} \cdot \ldots \cdot x_k^{n_k}$ необходимо выбрать $x_1$ в качестве множителя в $n_1$ скобках при раскрытии выражения $(x_1 + x_2 + \ldots + x_k)^n$. Это можно сделать $C_n^{n_1}$ способами. Из оставшихся $n - n_1$ не раскрытых скобок необходимо выбрать $x_2$ в качестве множителя в $n_2$ скобках. Это можно сделать $C_{n-n_1}^{n_2}$ способами и т. д. Тогда количество слагаемых $x_1^{n_1} x_2^{n_2} \cdot \ldots \cdot x_k^{n_k}$ при раскрытии выражения (2.2) будет равно числу $C_n^{n_1} C_{n-n_1}^{n_2} \cdot \ldots \cdot C_{n-n_1-n_2-\ldots-n_{k-1}}^{n_k} = \dfrac{n}{n_1!\, n_2! \cdot \ldots \cdot n_k!}$ упорядоченных разбиений.

## § 2.12. Бином Ньютона

Частный вид полиномиальной формулы (2.2):

$$(a + b)^n = \sum_{k=0}^{n} C_n^k a^k b^{n-k} \quad (2.3)$$

называется *биномом Ньютона*. Рассмотрим несколько задач, в основе решения которых лежит бином Ньютона.

З а д а ч а. Доказать тождество $\sum_{k=0}^{n} C_n^k = 2^n$.

Р е ш е н и е. Воспользуемся формулой (2.3) бинома Ньютона, в которой положим $a = 1$ и $b = 1$, тогда $(1 + 1)^n = \sum_{k=0}^{n} C_n^k 1^k 1^{n-k}$.

З а д а ч а. Доказать тождество $\sum_{k=0}^{n} C_n^k (m-1)^{n-k} = m^n$.

Р е ш е н и е. Воспользуемся формулой (2.3), где положим $a = 1$ и $b = m - 1$.

З а д а ч а. Доказать, что $\sum_{k=0}^{\lfloor n/2 \rfloor} C_n^{2k} = \sum_{k=1}^{\lceil n/2 \rceil} C_n^{2k-1} = 2^{n-1}$.

Р е ш е н и е. Воспользуемся формулой (2.3), в которой положим $a = -1$ и $b = 1$, тогда $(-1 + 1)^n = \sum_{k=0}^{n} (-1)^k C_n^k = 0$. Группируя положительные и отрицательные члены равенства, установим $\sum_{k=0}^{\lfloor n/2 \rfloor} C_n^{2k} = \sum_{k=1}^{\lceil n/2 \rceil} C_n^{2k-1}$. Так как $\sum_{k=0}^{\lfloor n/2 \rfloor} C_n^{2k} + \sum_{k=1}^{\lceil n/2 \rceil} C_n^{2k-1} = \sum_{k=0}^{n} C_n^k = 2^n$, то каждая из сумм — это половина числа $2^n$.

З а д а ч а. Доказать тождество $\sum_{k=0}^{n} k\, C_n^k = n\, 2^{n-1}$.

Р е ш е н и е. Воспользуемся формулой (2.3), из которой, полагая $a = 1$ и $b = x$, получим $(1 + x)^n = \sum_{k=0}^{n} C_n^k x^k$. Дифференцирование последнего равенства дает $n(1 + x)^{n-1} = \sum_{k=0}^{n} k\, C_n^k x^{k-1}$. Пусть $x = 1$, тогда $n(1 + 1)^{n-1} = \sum_{k=0}^{n} k\, C_n^k 1^{k-1}$, что доказывает искомое тождество.

## § 2.13. Инверсии

Перестановки особенно важны при изучении алгоритмов сортировки, так как они служат для представления неупорядоченных исходных данных. Чтобы исследовать эффективность различных методов сортировки, нужно уметь подсчитывать число перестановок, которые вынуждают повторять некоторый шаг алгоритма определенное число раз.

Пусть $(a_1, a_2, \ldots, a_n)$ — перестановка элементов множества $\{1, 2, \ldots, n\}$. Если $i < j$, а $a_i > a_j$, то пара $(a_i, a_j)$ называется *инверсией перестановки*. Например, перестановка $(3, 1, 4, 2)$ имеет три инверсии $(3, 1), (3, 2), (4, 2)$. Каждая инверсия — это пара элементов, «нарушающих порядок»; следовательно, единственная перестановка, не содержащая инверсий, — это отсортированная перестановка $(1, 2, \ldots, n)$.

*Таблицей инверсий перестановки* $(a_1, a_2, \ldots, a_n)$ называется последовательность $(d_1, d_2, \ldots, d_n)$, где $d_j$ — число элементов,

больших $j$ и расположенных левее $j$. Другими словами, $d_j$ — число инверсий, у которых второй элемент равен $j$. Например, таблица инверсий перестановки $(5,9,1,8,2,6,4,7,3)$ будет $(2,3,6,4,0,2,2,1,0)$, поскольку 5 и 9 расположены левее 1; 5, 9, 8 — левее 2 и т. д. Всего 20 инверсий. По определению $0 \leqslant d_1 \leqslant n-1,\ 0 \leqslant d_2 \leqslant n-2, \ldots, 0 \leqslant d_{n-1} \leqslant 1,\ d_n = 0$.

*М. Холл установил, что таблица инверсий единственным образом определяет соответствующую перестановку.* Из любой таблицы инверсий $(d_1, d_2, \ldots, d_n)$ можно однозначно восстановить перестановку, которая порождает данную таблицу, путем последовательного определения относительного расположения элементов $n, n-1, \ldots, 1$ (в этом порядке). Например, перестановку, соответствующую таблице инверсий $(2,3,6,4,0,2,2,1,0) = (d_1, d_2, \ldots, d_9)$, можно построить следующим образом: выпишем число 9; так как $d_8 = 1$, то 8 стоит правее 9. Поскольку $d_7 = 2$, то 7 стоит правее 8 и 9. Так как $d_6 = 2$, то 6 стоит правее двух уже выписанных чисел; таким образом, получили расположение $(9,8,6,7)$. Припишем теперь 5 слева, потому что $d_5 = 0$; помещаем 4 вслед за четырьмя из уже записанных чисел, 3 — после шести выписанных чисел (т. е. в правый конец) и получаем $(5,9,8,6,4,7,3)$. Вставив аналогичным образом 2 и 1, придем к перестановке $(5,9,1,8,2,6,4,7,3)$.

Такое соответствие между перестановками и таблицами инверсий важно, потому как часто можно заменить задачу, сформулированную в терминах перестановок, эквивалентной ей задачей, сформулированной в терминах таблиц инверсий. Рассмотрим, например, еще раз вопрос: сколько существует перестановок множества $\{1, 2, \ldots, n\}$? Ответ должен быть равен числу всевозможных таблиц инверсий, а их легко пересчитать, так как $d_1$ можно выбрать $n$ различными способами, $d_2$ можно не зависимо от $d_1$ выбрать $n-1$ способами и т. д., $d_n$ — одним способом. Тогда различных таблиц инверсий $n(n-1) \cdot \ldots \cdot 1 = n!$. Таблицы инверсий пересчитать легко, потому что все $d_j$ независимые, в то время как элементы $a_j$ перестановки должны все быть различными.

## § 2.14. Обратные перестановки

Не следует путать «инверсии» перестановок с обратными перестановками. Пусть $(a_1, a_2, \ldots, a_n)$ — различные шары, индексы которых свяжем с номерами шаров. Тогда исходное расположение шаров однозначно определяется тождественной перестановкой $e = (1, 2, \ldots, n)$. Пусть $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — произ-

вольная перестановка номеров $1, 2, \ldots, n$, где $\pi_k$ — номер шара на $k$-м месте. Такая перестановка отвечает расположению шаров $(a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_n})$. Вспомним (см. п. 2.5), что перестановка $\pi = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix}$ записана в матричном виде. Данная форма записи позволяет рассматривать перестановку в качестве оператора, который заменяет старые номера шаров (верхняя строка матрицы) на новые номера (нижняя строка матрицы). Тогда результат двух последовательных изменений $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ и $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ исходной последовательности $1, 2, \ldots, n$ номеров шаров можно рассматривать как операцию умножения перестановок $\rho = \pi\sigma = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & \ldots n \\ \sigma_1 & \sigma_2 & \ldots \sigma_n \end{pmatrix}$. Упорядочим столбцы перестановки $\sigma$ в соответствии с перестановкой $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, т. е. $\begin{pmatrix} 1 & 2 & \ldots n \\ \sigma_1 & \sigma_2 & \ldots \sigma_n \end{pmatrix} = \begin{pmatrix} \pi_1 & \pi_2 & \ldots \pi_n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \ldots \sigma_{\pi_n} \end{pmatrix}$. Тогда можно записать, что $\rho = \pi\sigma = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix} \cdot \begin{pmatrix} \pi_1 & \pi_2 & \ldots \pi_n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \ldots \sigma_{\pi_n} \end{pmatrix} = \begin{pmatrix} 1 & 2 & \ldots n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \ldots \sigma_{\pi_n} \end{pmatrix} = \begin{pmatrix} 1 & 2 & \ldots n \\ \rho_1 & \rho_2 & \ldots \rho_n \end{pmatrix}$. Этой перестановке отвечает расположение шаров $(a_{\rho_1}, a_{\rho_2}, \ldots, a_{\rho_n})$, где значение $\rho_k = \sigma_{\pi_k}$ — это номер шара на $k$-м месте.

*Обратной* к перестановке $\pi = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix}$ называется перестановка $\pi^{-1} = \begin{pmatrix} \pi_1 & \pi_2 & \ldots \pi_n \\ 1 & 2 & \ldots n \end{pmatrix} = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1^{-1} & \pi_2^{-1} & \ldots \pi_n^{-1} \end{pmatrix}$, которая получается, если в исходной перестановке поменять местами строки, а затем упорядочить столбцы в возрастающем порядке по верхним элементам, т. е. $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \ldots, \pi_n^{-1})$. Ясно, что последовательное изменение порядка шаров согласно перестановкам $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ и обратной $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \ldots, \pi_n^{-1})$ приводит к исходному их расположению, т. е. к тождественной перестановке $e = \begin{pmatrix} 1 & 2 & \ldots n \\ \pi_1 & \pi_2 & \ldots \pi_n \end{pmatrix} \cdot \begin{pmatrix} \pi_1 & \pi_2 & \ldots \pi_n \\ 1 & 2 & \ldots n \end{pmatrix} = \begin{pmatrix} 1\,2 \ldots n \\ 1\,2 \ldots n \end{pmatrix}$. Например, обратной к перестановке $(5, 9, 1, 8, 2, 6, 4, 7, 3)$ будет перестановка $(3, 5, 9, 7, 1, 6, 8, 4, 2)$, так как $\begin{pmatrix} 5\,9\,1\,8\,2\,6\,4\,7\,3 \\ 1\,2\,3\,4\,5\,6\,7\,8\,9 \end{pmatrix} = \begin{pmatrix} 1\,2\,3\,4\,5\,6\,7\,8\,9 \\ 3\,5\,9\,7\,1\,6\,8\,4\,2 \end{pmatrix}$.

Сортированную последовательность элементов перестановки $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ можно получить, заполнив в цикле вектор $e_1, e_2, \ldots, e_n$:

$$\textbf{for}\ \ k = 1\ \ \textbf{to}\ \ n\ \ \textbf{do}\ \ e_{\pi_k} = \pi_k\ \ \text{или}\ \ e[\pi_k] = \pi_k.$$

Ясно, что результирующие значения $e_1, e_2, \ldots, e_n$ будут соответственно $1, 2, \ldots, n$. В цикле каждый элемент $\pi_k$ вста-

ет на свое упорядоченное место $e_{\pi_k}$ (см. п. 6.7). Подобным образом выполним заполнение элементов перестановки $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \ldots, \pi_n^{-1})$, однако в упорядоченное место $\pi_{\pi_k}^{-1}$ элемента $\pi_k$ будем размещать его номер в исходной перестановке $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$:

$$\textsf{for}\ \ k = 1\ \ \textsf{to}\ \ n\ \ \textsf{do}\ \ \pi_{\pi_k}^{-1} = k\ \ \textsf{или}\ \ \pi^{-1}[\pi_k] = k.$$

Результирующий вектор $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \ldots, \pi_n^{-1})$ будет обратной перестановкой к $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$.

Х. А. Роте впервые установил связь между обратными перестановками и инверсиями: *обратная перестановка содержит ровно столько же инверсий, сколько исходная*.

Глава 3

# ПРЕДСТАВЛЕНИЕ АБСТРАКТНЫХ ОБЪЕКТОВ

Рассматривая решение задачи с абстрактной точки зрения, как правило, избегают каких бы то ни было предположений относительно того, как мы намерены автоматизировать их решение. В идеале структура вычислительной машины должна соответствовать естественной структуре задачи, однако это требование часто не выполняется, хотя приспособляемость современной вычислительной техники такова, что она позволяет обойти эти ограничения без труда. Языки высокого уровня при условии, что они должным образом сконструированы, предоставляют в распоряжение программиста, реализующего алгоритм, более удобную «машину», смягчая несоответствие основной машины требованиям алгоритма. Будем считать, что читатель знаком с элементарными понятиями математики и основными типами данных: целыми и вещественными числами, массивами, строками и т. д.

## § 3.1. Представление последовательностей

Любой заданный класс абстрактных объектов может иметь несколько возможных представлений, и выбор наилучшего из них решающим образом зависит от того, каким образом объект будет использован, а также от типа производимых над ним операций.

### 3.1.1. Смежное представление

В алгоритмах на дискретных структурах часто приходится встречаться с представлением конечных последовательностей и операциями с ними. С вычислительной точки зрения простейшим представлением конечной последовательности $s_1, s_2, \ldots, s_n$ является точный список ее членов, расположенных по порядку в смежных ячейках памяти. В языках высокого уровня — это одномерные, двухмерные и т. д. массивы данных. Очевидные преимущества последовательного представления имеют и некоторые значительные недостатки. Смежное представление становится неудобным, если требуется изменить последовательность

путем включения новых и исключения имеющихся там элементов. Включение между $s_i$ и $s_{i+1}$ нового элемента требует сдвига $s_{i+1}, s_{i+2}, \ldots, s_n$ вправо на одну позицию; аналогично исключение требует сдвига тех же элементов на одну позицию влево, как показано в алгоритме 3.1.

Алгоритм 3.1.

```
{ z  i }
  n = n + 1;
  for  j = n − 1 to  i  by  −1  do  s_{j+1} = s_j;
  s_i = z.
{  i }
  for  j = i to  n − 1  do  s_j = s_{j+1};
  n = n − 1.
```

В обоих случаях включение или удаление элементов при смежном представлении требует перемещения многих элементов. С точки зрения времени обработки такое перемещение элементов может оказаться дорогостоящим из-за сложности $O(n)$ операций включения и удаления.

### 3.1.2. Характеристические векторы

Важной разновидностью смежного размещения является случай, когда такому представлению подвергается подпоследовательность $s_{k_1}, s_{k_2}, \ldots, s_{k_m}$ некоторой основной последовательности $s_1, s_2, \ldots, s_n$. В этом случае подпоследовательность можно представить более удобно, используя характеристический вектор — последовательность из нулей и единиц, где $i$-й разряд равен единице, если $s_i$ принадлежит рассматриваемой подпоследовательности.

Например, для последовательности $(1, 2, 3, 4, 5, 6, 7, 8, 9)$ характеристический вектор подпоследовательности чисел, кратных 3, имеет вид $(0, 0, 1, 0, 0, 1, 0, 0, 1)$.

Характеристические векторы полезны в том случае, когда формирование нужной подпоследовательности выполняется путем последовательного удаления из основной последовательности элементов, которые не входят в подпоследовательность. Главное неудобство характеристических векторов состоит в том, что они не экономичны.

### 3.1.3. Связанное размещение

Неудобство включения и исключения элементов при смежном представлении происходит из-за того, что порядок следования элементов задается неявно требованием, чтобы смежные элементы последовательности находились в смежных ячейках памяти. В результате многие элементы последовательности во время включения или исключения должны передвигаться.

Если требование последовательного размещения элементов опущено, то операции включения и исключения можно выполнить без того, чтобы передвигать элементы. *При любом размещении элементов необходимо сохранять информацию о способе их упорядочения*. При связанном размещении последовательности $s_1, s_2, \ldots, s_n$ каждому $s_i$ ставится в соответствие указатель $l_i$, который указывает на следующую подобную пару элементов $s_{i+1}, l_{i+1}$ по списку. Вводится начальный указатель $l_0$, который указывает на первый элемент последовательности. Последний указатель $l_n$ в списке является пустым или нулевым, признак конца списка. Графическое представление связанного списка можно изобразить следующим образом:

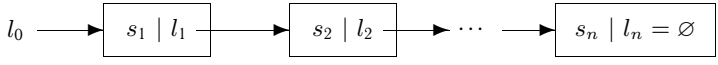



Рис. 3.1. Однонаправленный связанный список

Здесь каждый элемент связанного списка состоит из двух полей. В поле *DATA* размещен сам элемент последовательности, а в поле *NEXT* — указатель на следующий за ним элемент. Связанное представление последовательностей облегчает операции включения и удаления элементов из списка. Например, для исключение второго элемента достаточно переустановить указатели $NEXT(l_1) = NEXT(l_2)$. Графически это изображается следующим образом:

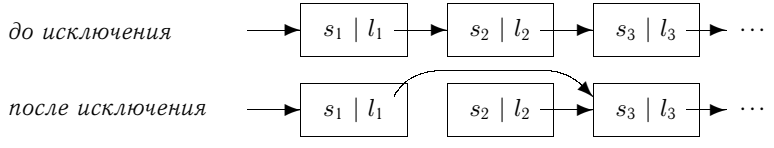



Рис. 3.2. Исключение элемента из списка

Чтобы в последовательность включить новый элемент $s_r$ после $s_1$, необходимо установить указатели: $NEXT(l_r) = NEXT(l_1)$ и $NEXT(l_1) = l_r$, начальное значение указателя $l_r$ установлено

на новый включаемый элемент. Графически включение нового элемента изображается так:

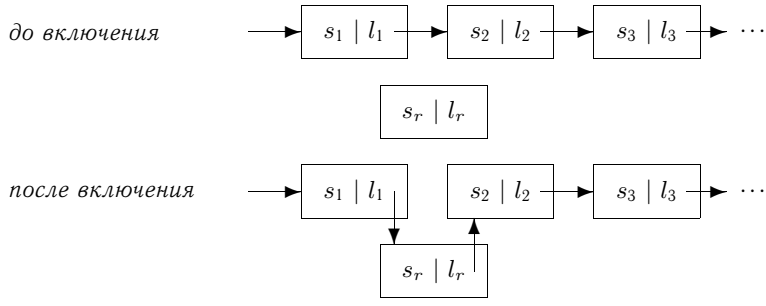



Рис. 3.3. Включение элемента в список

С помощью связанных распределений мы добились большей гибкости, но потеряли возможность работать с элементами последовательности как с массивами, когда по номеру $i$ можно непосредственно обратиться к элементу $s_i$. В связанном размещении такой возможности не существует, и доступ к элементам последовательности не является прямым и эффективным. Например, при поиске среднего элемента последовательности, даже при известной ее длине, требуется просмотреть по связанному списку половину последовательности. В алгоритмах 3.2 и 3.3 приводятся программы, реализованные на языках Pascal и C, связанного формирования списка элементов последовательности. В программу включены операции работы со списком: печать элементов списка, включение новых элементов в список и удаление элементов из списка.

Существуют различные модификации представления последовательностей в виде связанных списков. Следующие два примера позволят, при желании, читателю самостоятельно продолжить получение и других модификаций связанных распределений, отличных от приводимых ниже.

Циклическая форма представления позволяет эффективно возвращаться с последнего элемента списка к первому.

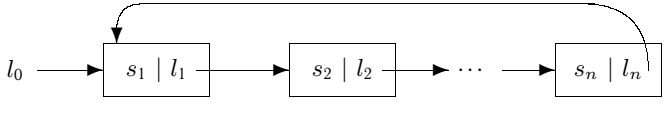



Рис. 3.4. Циклический список

Еще большая гибкость достигается, если использовать дважды связанный список, когда каждый элемент последовательности вместо одного имеет два связанных с ним указателя. В таком списке для любого элемента имеется мгновенный прямой доступ к предыдущему и последующему элементам. Следует помнить, что выбор того или иного представления последовательности в значительной степени зависит от типа операций, выполняемых с элементами последовательности.

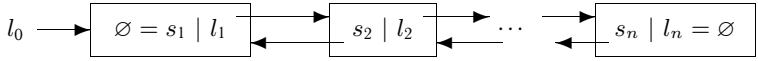



Рис. 3.5. Дважды связанный список

Алгоритм 3.2. Pascal

```
Program Number_List; { }
uses CRT;
Type
  NodePointer=^Node;
  Node= RECORD { }
    s  :Integer; {}
    next :NodePointer; {  }
  END;
Const
  first :NodePointer=NIL; { }

{ }
function InitNode: NodePointer;
var
  newNode :NodePointer;
begin
  New(newNode); {  }
  newNode^.s:=Random(99)+1; { }
  newNode^.next:=NIL;
  InitNode:=newNode;
end;
```

```
{    }
procedure IncludeNode(newNode: NodePointer);
begin
  newNode^.next:=first;
  first:=newNode;
end;
{      }
procedure DeleteNode( k: Integer);
var
  previos,current :NodePointer;
  i :Integer;
begin
  i:=0;
  current:=first;
  while current<>NIL do begin
    i:=i+1;
    if i=k then begin {      }
      if first=current then first:=current^.next
      else previos^.next:=current^.next; { }
      dispose(current);
      break;
    end;
    previos:=current;
    current:=current^.next;
  end;
end;
{ }
procedure PrintNodeList;
var
  p :NodePointer;
begin
  WriteLn;
  p:=first;
  while p<>nil do begin
    Write(p^.s:3,' ');
    p:=p^.next
  end;
end;
```

```
Var{       }
  i,m,n :Integer;
begin {       }
  ClrScr;
  Randomize;
  n:=17; {     }
  for i:=1 to n do IncludeNode(InitNode);
  PrintNodeList;
  WriteLn;
  m:=17; {       }
  DeleteNode(m);
  PrintNodeList;
  ReadKey;
end. {       }
```

Алгоритм 3.3. Си

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
typedef struct tagNode{
  int s;
  tagNode *next;
} Node;
typedef Node *NodePointer;
Node *first=NULL;


NodePointer InitNode( void ){
  NodePointer newNode;
  newNode=new Node;
  newNode ->s=random(99)+1;
  newNode ->next=NULL;
  return newNode;
}

void IncludeNode( NodePointer newNode ){
```

```
  newNode ->next=first;
  first=newNode;
}

void DeleteNode( int k ){
  NodePointer previos,current;
  int i;
  i=0;
  current=first;
  while( current!=NULL ){
    i++;
    if( i==k ){
      if( first==current ) first=current ->next;
      else previos ->next=current ->next;
      delete current;
      break;
    }
    previos=current;
    current=current ->next;
  }
}

void PrintNodeList( void ){
  NodePointer p;
  p=first;
  while( p!=NULL ){
    printf("%3d ",p ->s);
    p=p ->next;
  }
}
void main( void ){
  int i,m,n;
  clrscr();
  randomize();
  n=17;
  for( i=0; i<n; i++ ) IncludeNode( InitNode() );
  PrintNodeList();
  printf("\n");
```

```
  m=17;
  DeleteNode(m);
  PrintNodeList();
  getch();
}
```

Связанное представление предпочтительнее лишь в том случае, если в значительной степени используются операции включения и исключения элементов.

## § 3.2. Представление деревьев

Конечное *корневое дерево* $T$ формально определяется как непустое конечное множество упорядоченных узлов таких, что существует один выделенный узел, называемый *корнем* дерева, а оставшиеся узлы разбиты на $m \geqslant 0$ поддеревьев $T_1, T_2, \ldots, T_m$.

Корневое дерево на рис. 3.6 содержит 9 узлов, помеченных буквами от $a$ до $r$. Узлы с метками $e, f, c, g, h, r$ являются листьями, остальные узлы — внутренние. Узел с меткой $a$ — корень. Понятие дерева используется в различных аспектах. Деревья — наиболее важные нелинейные объекты, используемые для представления данных в алгоритмах на дискретных структурах.

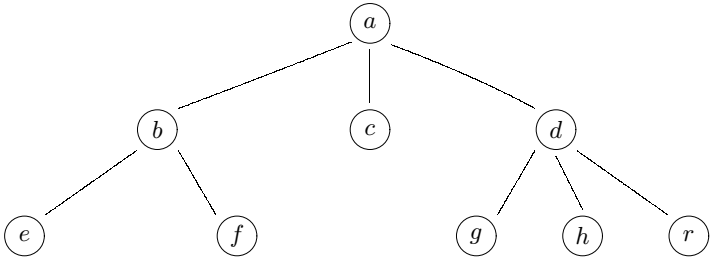



Рис. 3.6. Корневое дерево с тремя поддеревьями

Важной разновидностью корневых деревьев являются бинарные деревья. Бинарное дерево $T$ либо пусто, либо состоит из выделенного узла, называемого корнем, и двух бинарных поддеревьев: левого $T_1$ и правого $T_2$. *Лесом* называют упорядоченное множество деревьев. Тогда корневое дерево можно определить как непустое множество узлов, такое, что существует один выделенный узел, называемый корнем дерева, а оставшиеся узлы образуют лес с поддеревьями корня.

### 3.2.1. Представление деревьев на связанной памяти

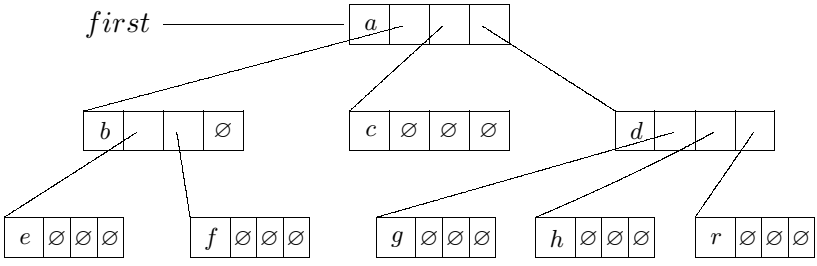



Рис. 3.7. Регулярная связанная структура представления дерева

Почти все машинные представления деревьев основаны на связанных распределениях. Каждый узел состоит из поля данных и некоторых полей для указателей. В следующем примере на рис. 3.7 представления дерева каждый узел имеет по три поля указателей.

Произвольное дерево с переменным числом поддеревьев всегда можно представить с помощью односторонних списков с использованием двухкомпонентных звеньев (3.8), в которых в первом поле находится либо указатель, либо данные, а во втором — всегда указатель.

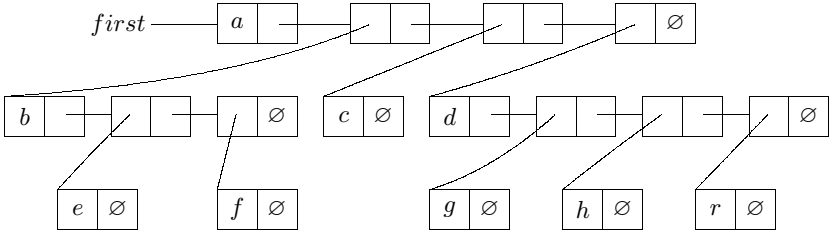



Рис. 3.8. Универсальная связанная структура представления дерева

Применение указателей и связанных списков придает памяти гибкость, необходимую для представления различных структур. Но при этом легко и перестараться; поэтому следует избегать слишком большого количества указателей; сложность программной поддержки таких структур возрастает «экспоненциально», теряется четкость основной структуры, которую пытаются представить в памяти (последний пример представления дерева это наглядно подтверждает).

### 3.2.2. Представление деревьев на смежной памяти

Представление деревьев на смежной памяти (одномерный массив) предполагает неявное присутствие ребер, переход по которым выполняется посредством арифметических операций над индексами элементов массива — смежной памяти. Формирование таких деревьев с помощью адресной арифметики можно осуществлять двумя способами. Идея первого способа применима при любом постоянном количестве ребер, выходящих из вершин (регулярное дерево). Рассмотрим данный способ формирования на примере двоичного (бинарного) дерева.

Пусть имеется одномерный массив смежных элементов $a_1, a_2, \ldots, a_n$. Неявная структура двоичного дерева определяется как на рис. 3.9. По дереву на рис. 3.9 легко перемещаться в обоих направлениях. Переход вниз на один уровень из вершины $a_k$ можно выполнить, удвоив индекс $k$ (индекс левого поддерева) или удвоив и прибавив 1 (индекс правого поддерева). Переход вверх на один уровень из вершины $a_m$ можно выполнить, разделив $m$ пополам и отбросив дробную часть. Рассмотренная структура применима к любому дереву с постоянным количеством ребер, выходящих из вершин.

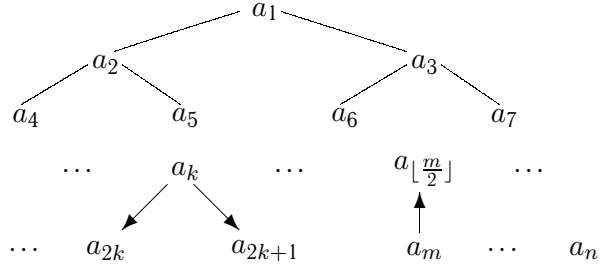



Рис. 3.9. Двоичное дерево на смежной памяти с последовательной нумерацией вершин

Другой способ, основанный на индексной арифметике, применим только для *двоичных деревьев*. Пусть для представления дерева используется одномерный массив $a_i, a_{i+1}, \ldots, a_j$. Корнем дерева полагают элемент $a_m$, где индекс элемента корня рассчитывается по формуле $m = \lfloor (i + j)/2 \rfloor$, т. е. середина массива. Левое поддерево располагается в массиве $a_i, a_{i+1}, \ldots, a_{m-1}$, а правое поддерево — в массиве $a_{m+1}, a_{m+2}, \ldots, a_j$. Корни поддеревьев рассчитываются подобным же образом, как и корень основного дерева. Второй способ формирования двоичных деревьев

на смежной памяти имеет довольно ограниченное применение. Основное его использование — поиск данных в *сортированных* массивах, таблицах и т. д.

В качестве примера использования представления регулярных деревьев *на смежной памяти* рассмотрим решение следующей задачи.

З а д а ч а. Написать программу поиска всех замкнутых маршрутов длины $n < 15$ по ребрам треугольника **abc**. Длину ребра принять равной 1. Начальная и конечная точка искомых маршрутов — вершина **a**. Длина маршрута $n$ задается в текстовом файле исходных данных. Результаты расчетов всех маршрутов сохранить в выходном текстовом файле. Каждый маршрут представить как последовательную комбинацию меток **a,b,c** посещаемых вершин треугольника при движении по нему. Каждый маршрут должен включать $n + 1$ метку, где первой и последней меткой должна быть вершина **a**.

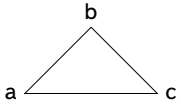


Рис. 3.10

Пример файла исходных данных:

```
4
```

Выходной файл для данного примера:

```
abcba   1
ababa   2
acaba   3
abaca   4
acaca   5
acbca   6
```

В алгоритме 3.4 представлена программа расчета всех искомых маршрутов длины $n$. Алгоритм делится на две части. В первой части (процедура **CreateTreeAbc**) выполняется формирование двоичного регулярного дерева на смежной памяти рис. 3.11.

А л г о р и т м 3.4. **Pascal**

```
Program Tree_Abc; {        }
uses CRT,DOS;
const n_max=$fc00; {   }
Type Vector=array[1..n_max] of Char;
Var
  f :Text; { }
  z :Vector; {    }
```

```
Procedure CreateTreeAbc( n:Integer ); { }
var
  k,level,m,m1,m2 :LongInt;
begin
  z[1]:='a'; {} a
  level:=1; { }
  m1:=1; {  }
  m2:=1; {  }
  while level<=n do begin
    for k:=m1 to m2 do begin {  }
      m:=2*k;
      case z[k] of
        'a': begin z[m]:='b'; z[m+1]:='c'; end;
        'b': begin z[m]:='c'; z[m+1]:='a'; end;
        'c': begin z[m]:='a'; z[m+1]:='b'; end;
      end;
    end;
    level:=level+1;
    m1:=2*m1;
    m2:=2*m2+1;
  end;
end;

Procedure RouteTreeAbc( n:Integer ); { }
var
  i,k,m1,m2,r :LongInt;
begin
  r:=0; {}
  k:=1;
  for i:=1 to n do k:=2*k;
  m1:=k; {    }
  m2:=2*k-1; {    }
  for i:=m1 to m2 do begin {    }
    k:=i;
    if z[k]='a' then begin
      r:=r+1;
      WriteLn(f);
      repeat
        Write(f,z[k]);
```

```
          k:=k div 2;
        until k=0;
        Write(f,' ',r);
      end;
  end;
end;
Var {      }
  n :Integer; { }
begin {      }
  Assign(f,'treeabc.in');
  Reset(f); {   }
  Read(f,n); { }
  Close(f);
  Assign(f,'treeabc.out');
  Rewrite(f); {   }
  CreateTreeAbc(n); {          }
  RouteTreeAbc(n); {     }
  Close(f);
end. {      }
```

При проходе вниз вершины дерева заполняются метками **a, b, c** соответствующими вершинам треугольника при перемещении по нему. Два ребра, выходящих из каждой вершины, показывают возможные варианты выбора дальнейшего маршрута продвижения по треугольнику. В каждом случае из вершин **a, b, c** можно попасть в любые две другие вершины. Индексы меток дерева прохода на рис. 3.11 показывают соответствующее им место в массиве данных (смежной памяти).

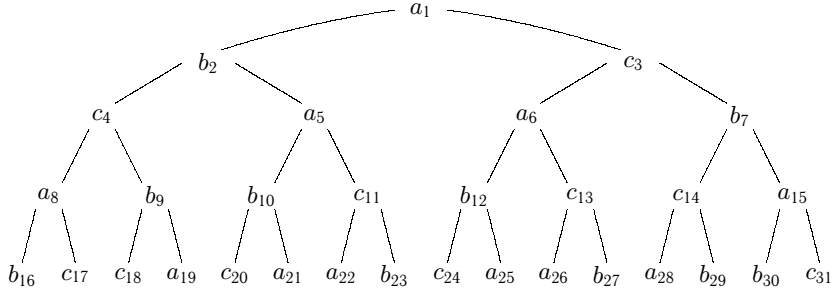



Рис. 3.11. Двоичное дерево маршрутов по треугольнику на смежной памяти

Во второй части алгоритма выполняется формирование искомых маршрутов (процедура RouteTreeAbc), основой для построения которых служит дерево прохода на рис. 3.11. Для формирования всех маршрутов теперь достаточно подняться по нему от вершин треугольника с метками a к корню, запоминая пройденные метки. Ясно, что число маршрутов будет равно числу вершин (листьев) на последнем уровне с меткой a.

## § 3.3. Представление множеств

Существуют два основных подхода к представлению множеств в памяти.

1. При первом подходе хранят описание каждого элемента, действительно присутствующего в множестве, как это делается, когда выписываются все элементы множества и заключаются в фигурные скобки.
2. При втором подходе изначально определяются все потенциально возможные элементы множества, а затем для любого подмножества этого универсального множества для каждого возможного члена указывается, принадлежит ли он на самом деле данному подмножеству или нет.

При первом подходе представления множеств используют как смежное, так и связанное размещение его элементов в памяти (рис. 3.12). Данные методы размещения подробно рассмотрены в п. 3.1 представления последовательностей.

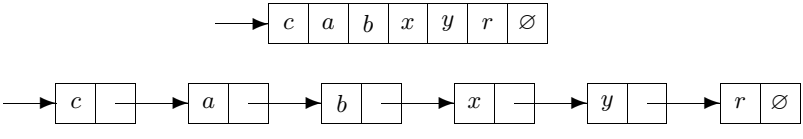



Рис. 3.12. Смежное и связанное представление множества в памяти

Как и для последовательностей, наилучший метод представления множеств существенно зависит от операций, которые мы собираемся выполнять над ними. Типичные операции над множествами: выяснить, имеется ли конкретный элемент в данном множестве; добавить в множество новые элементы; удалить элементы из множества; выполнить обычные теоретико-множественные операции, такие, как объединение или пересечение двух множеств. Как правило, для представления множеств применяют связанную память.

При втором методе множество представляется в виде вектора на смежной памяти. Пусть $U$ — универсальное множество (т. е.

все рассматриваемые множества являются его подмножествами), состоящее из $n$ элементов. Любое подмножество $S \subseteq U$ представляется в виде *характеристического вектора* из $n$ элементов. Элемент $i$ в этом векторе равен 1 тогда и только тогда, когда $i$-й элемент множества $U$ принадлежит $S$, в противном случае устанавливается равным 0 (рис. 3.13). Представление в виде

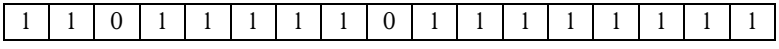



Рис. 3.13. Представление множества характеристическим вектором

характеристического вектора удобнее тем, что можно определять принадлежность $i$-го элемента множеству за время, не зависящее от его размера. Основные операции над множествами, такие, как объединение и пересечение, можно осуществлять как операции $\vee$ и $\wedge$ над двоичными векторами. Недостаток этого представления заключается в том, что операции объединение и пересечение занимают время, пропорциональное мощности универсального множества $U$, а не рассматриваемого множества $S$. Данное представление требует дополнительной памяти для хранения характеристического вектора, что для больших $n$ (размер универсального множества $U$) бывает практически невыполнимо.

Глава 4

# МЕТОДЫ ПОДСЧЕТА И ОЦЕНИВАНИЯ

Рассмотренные в предыдущих разделах комбинаторные формулы подсчета означают вычисление или определение свойств некоторой последовательности чисел, соответствующие той или иной задаче. В этом разделе предлагается полезный инструмент для работы с последовательностями. Идея состоит в том, чтобы каждой числовой последовательности сопоставить функцию действительного или комплексного переменного, с тем, чтобы обычные операции над последовательностями соответствовали простым операциям над соответствующими функциями. Аналитические методы работы с функциями оказываются проще и эффективнее, чем непосредственные комбинаторные методы работы с последовательностями.

## § 4.1. Производящие функции

Пусть $a_0, a_1, a_2, \ldots$ — произвольная последовательность. Сопоставим последовательности функцию действительного или комплексного переменного:

$$A(x) = \sum_{k=0}^{\infty} a_k x^k. \tag{4.1}$$

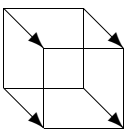
Функция $A(x)$ называется *производящей функцией* последовательности $a_0, a_1, a_2, \ldots$. Как правило, поиск функции $A(x)$ по формуле (4.1) прямыми методами является сложной задачей. Заметим, что последовательность $\{a_k\}$ может быть восстановлена по $A(x)$. Выражение (4.1) является разложением в ряд Тейлора в окрестности точки $x = 0$. Воспользуемся этим замечанием и приведем некоторые наиболее распространенные производящие функции и соответствующие им последовательности.

Задача. Найти число $k$-мерных граней в $n$-мерном кубе.

Решение. Обозначим через $a_k$ число $k$-мерных граней в $n$-мерном кубе, где $0 \leqslant k \leqslant n$. Тогда производящую функцию

| Производящие функции $A(x)$ | Последовательности $\{a_k\}$ | |
|---|---|---|
| $\frac{1}{1-x} = \sum\limits_{k=0}^{\infty} 1 \cdot x^k$ | $a_k = 1,\ k \geqslant 0$ | (4.2) |
| $\frac{1}{(1-x)^2} = \sum\limits_{k=0}^{\infty} (k+1)x^k$ | $a_k = k+1,\ k \geqslant 0$ | (4.3) |
| $\ln \frac{1}{1-x} = \sum\limits_{k=1}^{\infty} \frac{1}{k} x^k$ | $a_0 = 0, a_k = \frac{1}{k},\ k \geqslant 1$ | (4.4) |
| $\ln(1+x) = \sum\limits_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} x^k$ | $a_0 = 0, a_k = \frac{(-1)^{k+1}}{k},\ k \geqslant 1$ | (4.5) |
| $(1+x)^r = \sum\limits_{k=0}^{\infty} \binom{r}{k} x^k$ | $a_k = \binom{r}{k},\ k \geqslant 0,\ r$ — любое | (4.6) |
| $e^x = \sum\limits_{k=0}^{\infty} \frac{1}{k!} x^k$ | $a_k = \frac{1}{k!},\ k \geqslant 0$ | (4.7) |
| $e^{rx} = \sum\limits_{k=0}^{\infty} \frac{r^k}{k!} x^k$ | $a_k = \frac{r^k}{k!},\ k \geqslant 0,\ r$ — любое | (4.8) |
| $\frac{1}{(1-x)^r} = \sum\limits_{k=0}^{\infty} \binom{r+k-1}{k}$ | $a_k = \binom{r+k-1}{k},\ k \geqslant 0,\ r$ — любое | (4.9) |

последовательности $\{a_k\}$ можно записать как $A_n(x) = \sum\limits_{k=0}^{n} a_k x^k$. Индекс $n$ для $A_n(x)$ показывает размерность куба. Например, $A_0(x) = 1,\ A_1(x) = 2 + x,\ A_2(x) = 4 + 4x + x^2$. Рассмотрим производящую функцию $A_{n+1}(x)$ последовательности $\{a_k\}$ для $(n+1)$-мерного куба. Заметим, что $(n+1)$-мерный куб можно получить из $n$-мерного куба сдвигом последнего по $(n+1)$-му измерению. На рисунке показан пример получения трехмерного куба сдвигом по третьему измерению квадрата (двухмерного куба). Отсюда видно, что $(n+1)$-мерный куб включает два старых $n$-мерных куба и каждая $k$-мерная грань при сдвиге переходит в $(k+1)$-мерную грань. Из приведенных рассуждений следует, что $A_{n+1}(x) = 2A_n(x) + x \cdot A_n(x)$, где $A_0(x) = 1$. Отсюда $A_{n+1}(x) = (2+x)A_n(x) = (2+x)^{n+1}$. Воспользуемся разложением бинома Ньютона: $A_n(x) = (2+x)^n = \sum\limits_{k=0}^{n} C_n^k x^k 2^{n-k} = \sum\limits_{k=0}^{n} a_k x^k$. Сравнивая коэффициенты при степенях $x^k$, получим, что число $k$-мерных граней в $n$-мерном кубе равно $a_k = 2^{n-k} C_n^k$.



Например, $A_3(x) = 2^{3-0}C_3^0 + 2^{3-1}C_3^1 + 2^{3-2}C_3^2 + 2^{3-3}C_3^3 = 8 + 12x + 6x^2 + x^3$.

Простейшие производящие функции (4.2–4.9) будем использовать как «строительные кирпичики» для получения производящих функций более сложных последовательностей. С этой целью рассмотрим наиболее важные из операций над производящими функциями, т. е. способы получения новых производящих функций и соответствующих им последовательностей. Обозначим через $\{a_k\}$, $\{b_k\}$, $\{c_k\}$ последовательности, а соответствующие им производящие функции — как $A(x), B(x), C(x)$.

### 4.1.1. Линейные операции

Если $\alpha$ и $\beta$ — константы, то последовательность $c_k = \alpha a_k + \beta b_k$ имеет производящую функцию $C(x) = \alpha A(x) + \beta B(x)$. Например, последовательность $\{1\}$ соответствует производящей функции $\dfrac{1}{1-x}$, а последовательность $\left\{\dfrac{1}{k!}\right\}$ соответствует производящей функции $e^x$, тогда последовательность $\left\{100 + \dfrac{5}{k!}\right\}$ соответствует производящей функции $\dfrac{100}{1-x} + 5e^x$.

### 4.1.2. Сдвиг начала вправо

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = \begin{cases} 0, & \text{для } k = 0, 1, \ldots, i-1; \\ a_{k-i}, & \text{для } k = i, i+1, \ldots, \end{cases}$$

тогда $B(x) = x^i A(x)$. Действительно,

$$B(x) = \sum_{k=0}^{\infty} b_k x^k = \sum_{k=i}^{\infty} a_{k-i} x^k = x^i \sum_{k=0}^{\infty} a_k x^k = x^i A(x).$$

### 4.1.3. Сдвиг начала влево

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом: $b_k = a_{k+i}$, $k = 0, 1, \ldots$, тогда

$$B(x) = \Bigl(A(x) - \sum_{k=0}^{i-1} a_k x^k\Bigr) x^{-i}.$$

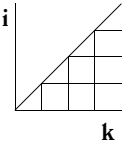
Действительно,

$$B(x)=\sum_{k=0}^{\infty}b_kx^k=\sum_{k=0}^{\infty}a_{k+i}x^k=x^{-i}\sum_{k=0}^{\infty}a_{k+i}x^{k+i}=x^{-i}\sum_{k=i}^{\infty}a_kx^k=$$

$$=x^{-i}\Big(\sum_{k=0}^{\infty}a_kx^k-\sum_{k=0}^{i-1}a_kx^k\Big)=\Big(A(x)-\sum_{k=0}^{i-1}a_kx^k\Big)x^{-i}.$$

### 4.1.4. Частичные суммы

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:




$$b_k=\sum_{i=0}^{k}a_i,\ k=0,1,\ldots,\ \text{тогда}\ B(x)=\frac{A(x)}{1-x}.$$

Действительно,

$$B(x)=\sum_{k=0}^{\infty}b_kx^k=\sum_{k=0}^{\infty}\Big(\sum_{i=0}^{k}a_i\Big)x^k.$$

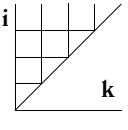
Множество пар точек $(k,i)$, по которым ведется суммирование, представлено на рисунке. Изменим порядок суммирования (сначала по $i$, затем по $k$). Выражение для $B(x)$ примет вид

$$B(x)=\sum_{i=0}^{\infty}a_i\Big(\sum_{k=i}^{\infty}x^i\Big)=\Big(\sum_{i=0}^{\infty}a_ix^i\Big)\Big(\sum_{k=0}^{\infty}x^k\Big)=A(x)\frac{1}{1-x}.$$

### 4.1.5. Дополнительные частичные суммы

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k=\sum_{i=k}^{\infty}a_i,\ k=0,1,\ldots,\ \text{тогда}\ B(x)=\frac{A(1)-xA(x)}{1-x}.$$




Действительно,

$$B(x)=\sum_{k=0}^{\infty}b_kx^k=\sum_{k=0}^{\infty}\Big(\sum_{i=k}^{\infty}a_i\Big)x^k.$$

Множество пар точек $(k,i)$, по которым ведется суммирование, представлено на рисунке. Изменим порядок суммирования

(сначала по $i$, затем по $k$). Выражение для B(x) примет вид

$$B(x) = \sum_{i=0}^{\infty}\sum_{k=0}^{i} a_i x^k = \sum_{i=0}^{\infty} a_i\Big(\sum_{k=0}^{i} x^k\Big) = \sum_{i=0}^{\infty} a_i \frac{1-x^{i+1}}{1-x} =$$

$$= \frac{1}{1-x}\Big(\sum_{i=0}^{\infty} a_i - x\sum_{i=0}^{\infty} a_i x^i\Big) = \frac{A(1)-xA(x)}{1-x}.$$

### 4.1.6. Изменение масштаба

1. Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = ka_k, \text{ тогда } B(x) = xA'(x).$$

Действительно,

$$A(x) = \sum_{k=0}^{\infty} a_k x^k \text{ и } A'(x) = \sum_{k=0}^{\infty} ka_k x^{k-1}$$

или

$$xA'(x) = \sum_{k=0}^{\infty}(ka_k)x^k = B(x).$$

2. Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = \frac{a_k}{k+1}, \text{ тогда } B(x) = \frac{1}{x}\int_0^x A(x)\,dx.$$

Поскольку $A(x) = \sum\limits_{k=0}^{\infty} a_k x^k$, то

$$\int_0^x A(x)\,dx = \sum_{k=0}^{\infty}\int_0^x a_k x^k dx = \sum_{k=0}^{\infty}\frac{a_k}{k+1}x^{k+1} = x\sum_{k=0}^{\infty} b_k x^k = xB(x).$$

### 4.1.7. Свертка

Последовательность

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}, \ k = 0, 1, \ldots, \text{ тогда } C(x) = A(x)B(x).$$

Действительно, $A(x) = \sum\limits_{k=0}^{\infty} a_k x^k$ и $B(x) = \sum\limits_{k=0}^{\infty} b_k x^k$, тогда

$$C(x) = \sum_{k=0}^{\infty} c_k x^k = \sum_{k=0}^{\infty} \Big( \sum_{i=0}^{k} a_i b_{k-i} \Big) x^k = \sum_{i=0}^{\infty} \sum_{k=i}^{\infty} a_i b_{k-i} x^k =$$

$$= \Big( \sum_{i=0}^{\infty} a_i x^i \Big) \Big( \sum_{k=i}^{\infty} b_{k-i} x^{k-i} \Big) = \Big( \sum_{i=0}^{\infty} a_i x^i \Big) \Big( \sum_{k=0}^{\infty} b_k x^k \Big) = A(x) B(x).$$

Далее обсудим наиболее общие приемы использования производящих функций на примере решения следующих задач.

З а д а ч а. Рассмотрим обобщенное биномиальное правило раскрытия выражений

$$\frac{1}{(1-x)^r} = (1-x)^{-r} = \sum_{k=0}^{\infty} \binom{-r}{k} (-x)^k,$$

где обобщенный биномиальный коэффициент

$$\binom{-r}{k} = \frac{(-r)(-r-1) \cdot \ldots \cdot (-r-k+1)}{k!} =$$

$$= (-1)^k \frac{r(r+1) \cdot \ldots \cdot (r+k-1)}{k!} = (-1)^k \binom{r+k-1}{k}.$$

Тогда

$$\frac{1}{(1-x)^r} = \sum_{k=0}^{\infty} \binom{-r}{k} (-x)^k = \sum_{k=0}^{\infty} \binom{r+k-1}{k} x^k = \sum_{k=0}^{\infty} C_{r+k-1}^k x^k.$$

Рассмотрим полученное выражение при $r = -\dfrac{1}{2}$:

$$\sqrt{1-x} = (1-x)^{1/2} = \sum_{k=0}^{\infty} \binom{1/2}{k} (-x)^k =$$

$$= 1 + \sum_{k=1}^{\infty} \frac{\frac{1}{2} \left( \frac{1}{2} - 1 \right) \left( \frac{1}{2} - k + 1 \right)}{k!} (-x)^k =$$

$$= 1 + \sum_{k=1}^{\infty} \frac{1(-1)(-3) \cdot \ldots \cdot (-2k+3)}{2^k k!} (-x)^k =$$

$$= 1 + \sum_{k=1}^{\infty} (-1)^{2k-1} \frac{1 \cdot 1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2k-3)}{2^k k!} x^k =$$

$$= 1 - \sum_{k=1}^{\infty} \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots \cdot (2k-3)}{2^k k!} x^k =$$

$$= 1 - \sum_{k=1}^{\infty} \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot \ldots \cdot (2k-3) \cdot (2k-2)}{2^k k! \cdot 2 \cdot 4 \cdot 6 \cdot \ldots \cdot (2k-2)} x^k =$$

$$= 1 - \sum_{k=1}^{\infty} \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot \ldots \cdot (2k-3) \cdot (2k-2)}{2^k k! 2^{k-1} (k-1)!} x^k =$$

$$= 1 - \sum_{k=1}^{\infty} \frac{2 \cdot (2k-2)!}{4^k k (k-1)! (k-1)!} x^k = 1 - \sum_{k=1}^{\infty} \frac{2 C_{2k-2}^{k-1}}{4^k k} x^k.$$

Таким образом,

$$\sqrt{1-x} = (1-x)^{1/2} = 1 - \sum_{k=1}^{\infty} \frac{2\, C_{2k-2}^{k-1}}{4^k k} x^k.$$

Рассмотрим выражение $\dfrac{1}{(1-x)^r} = (1-x)^r$ при $r = 1$:

$$\frac{1}{1-x} = (1-x)^{-1} = \sum_{k=0}^{\infty} \binom{-1}{k} (-x)^k,$$

где

$$\binom{-1}{k} = \frac{(-1)(-1-1) \cdot \ldots \cdot (-1-k+1)}{k!} =$$

$$= (-1)^k \frac{1 \cdot 2 \cdot \ldots \cdot k}{k!} = (-1)^k.$$

Следовательно, $\dfrac{1}{1-x} = (1-x)^{-1} = \sum\limits_{k=0}^{\infty} (-1)^k (-x)^k = \sum\limits_{k=0}^{\infty} x^k.$

Задача. Сколькими способами можно разбить выпуклый $(n+3)$-угольник, $n \geqslant 0$, на треугольники диагоналями, не пересекающимися внутри многоугольника?

Решение. Пусть $u_{n+3}$ — искомое число способов разбить $(n+3)$-угольник на треугольники. Перенумеруем вершины исходного многоугольника числами от 1 до $n+3$. Заметим, что при любом разбиении найдется треугольник, содержащий ребро

многоугольника с вершинами $n+2$ и $n+3$. Третья вершина этого треугольника может быть любой из остальных $1, 2, \ldots, n+1$. Пусть это будет вершина $k$. Если удалить треугольник с вершинами $n+2$, $n+3$, $k$, то получим два многоугольника с числом вершин $k+1$ и $n+3-k$, которые можно разбить на треугольники $u_{k+1}$ и $u_{n+3-k}$ способами. Суммируя по $k = 1, 2, \ldots, n+1$, получим (*согласно правилам прямого произведения и суммы*) искомое число разбиений исходного $(n+3)$-угольника на треугольники:

$$u_{n+3} = u_2 u_{n+2} + u_3 u_{n+1} + \ldots + u_{n+2} u_2,$$

где $n \geqslant 0$, и положили $u_2 = 1$.

Получили нелинейное рекуррентное соотношение для последовательности $\{u_{n+2}\}$, $n \geqslant 0$, для поиска которой удобно ввести новую последовательность $v_n$, $n \geqslant 0$, такую, что $v_n = u_{n+2}$, $n \geqslant 0$.

Тогда рекуррентное соотношение перепишется в виде

$$v_{n+1} = v_0 v_n + v_1 v_{n-1} + \ldots + v_n v_0$$

или

$$v_{n+1} = \sum_{k=0}^{n} v_k v_{n-k}.$$

Заметим, что правая часть является сверткой двух одинаковых последовательностей $\{v_n\}$ и $\{v_n\}$ (см. п. 4.1.7 операции с производящими функциями). Имея это ввиду, составим производящую функцию правой части:

$$\sum_{n=0}^{\infty} v_{n+1} x^n = \sum_{n=0}^{\infty} \Big( \sum_{k=0}^{n} v_k v_{n-k} \Big) x^n.$$

Пусть $V(x)$ — производящая функция последовательности $\{v_n\}$, $n \geqslant 0$, тогда запишем последнее соотношение:

$$\frac{V(x) - v_0}{x} = V(x) V(x) \quad \text{или} \quad x V^2(x) - V(x) + 1 = 0.$$

Отсюда $V(x) = \dfrac{1 \pm \sqrt{1-4x}}{2x}$.

Ранее рассмотренное разложение обобщенного бинома

$$\sqrt{1-x} = 1 - \sum_{k=1}^{\infty} \frac{2\, C_{2k-2}^{k-1}}{4^k k}\, x^k$$

запишем для случая

$$\sqrt{1-4x} = 1 - \sum_{k=1}^{\infty} \frac{2\,C_{2k-2}^{k-1}}{k}\,x^k.$$

Поскольку результат $V(x)$ должен быть рядом по неотрицательным степеням $x^k$, то решение $V(x) = (1/2x)(1+\sqrt{1-4x}\,)$ является посторонним. Окончательно

$$V(x) = \frac{1-\sqrt{1-4x}}{2x} = \sum_{k=1}^{\infty} \frac{C_{2k-2}^{k-1}}{k}\,x^{k-1} = \sum_{k=0}^{\infty} \frac{C_{2k}^{k}}{k+1}\,x^k.$$

Отсюда $u_{n+2} = v_n = C_{2n}^n/(n+1),\ n \geqslant 0$. Значит, число способов разбить выпуклый $(n+2)$-угольник на треугольники непересекающимися диагоналями равно $C_{2n}^n/(n+1)$, $n \geqslant 0$.

З а д а ч а. Найти сумму $1^2 + 2^2 + \ldots + k^2 = \sum\limits_{i=1}^{k} i^2$.

Р е ш е н и е. Определим четыре последовательности и их производящие функции:

$$a_k = 1,\ b_k = ka_k,\ c_k = kb_k,\ d_k = \sum_{i=0}^{k} i^2$$

и

$$A(x),\ B(x),\ C(x),\ D(x).$$

Для решения задачи необходимо найти $d_k$. С этой целью определим $D(x)$, что позволит нам установить значения $d_k$, $D(x) = \sum\limits_{k=0}^{\infty} d_k x^k$. Последовательности $b_k$ и $a_k$ связаны «изменением масштаба», значит, $B(x) = xA'(x)$. Последовательности $c_k$ и $b_k$ также связаны «изменением масштаба», следовательно,

$$C(x) = xB'(x) = x(xA'(x))' = xA'(x) + x^2A''(x).$$

Последовательности $d_k$ и $c_k$ связаны «частичной суммой» тогда

$$D(x) = \frac{C(x)}{1-x} = \frac{xA'(x) + x^2A''(x)}{1-x},$$

$$A(x) = \sum_{k=0}^{\infty} 1 \cdot x^k = \frac{1}{1-x}, \quad A'(x) = \frac{1}{(1-x)^2}, \quad A''(x) = \frac{2}{(1-x)^3}.$$

Окончательно: $D(x) = \dfrac{x(1+x)}{(1-x)^4}$. Для получения коэффициентов $d_k$ воспользуемся разложением (4.9):

$$\frac{1}{(1-x)^4} = \sum_{k=0}^{\infty} \binom{-4}{k} (-x)^k = \sum_{k=0}^{\infty} C_{k+3}^3 x^k.$$

Теперь можно записать, что

$$D(x) = (x+x^2) \sum_{k=0}^{\infty} C_{k+3}^3 x^k =$$

$$= \sum_{k=0}^{\infty} C_{k+3}^3 x^{k+1} + \sum_{k=0}^{\infty} C_{k+3}^3 x^{k+2} = C_3^3 x + \sum_{k=0}^{\infty} (C_{k+4}^3 + C_{k+3}^3) x^{k+2} =$$

$$= C_3^3 x + \sum_{k=2}^{\infty} (C_{k+2}^3 + C_{k+1}^3) x^k = \sum_{k=0}^{\infty} d_k x^k.$$

Сравнивая коэффициенты при одинаковых степенях $x$, получим

$$d_0 = 0, \quad d_1 = C_3^3, \quad d_k = C_{k+1}^3 + C_{k+2}^3, \quad k = 2, 3, \ldots.$$

Таким образом,

$$1^2 + 2^2 + \ldots + k^2 = C_{k+1}^3 + C_{k+2}^3 = \frac{(2k+1)(k+1)k}{6}.$$

З а д а ч а. Показать, что $\sum\limits_{i=0}^{k} C_{r+i}^i = C_{r+k+1}^k$ или

$$C_{r+0}^0 + C_{r+1}^1 + \ldots + C_{r+k}^k = C_{r+k+1}^k.$$

Р е ш е н и е. Заметим, что $\dfrac{1}{(1+x)^{r+1}} = \sum\limits_{k=0}^{\infty} C_{r+k}^k x^k$ является производящей функцией последовательности $a_k = C_{r+k}^k$. Следовательно, искомая сумма равна $\sum\limits_{i=0}^{k} C_{r+1}^i = \sum\limits_{i=0}^{k} a_i$. Рассмотрим последовательность $b_k = \sum\limits_{i=0}^{k} a_i$. Так как $b_k$ и $a_k$ связаны частичной суммой (п. 4.1.4), то $B(x) = \dfrac{A(x)}{1-x} = \dfrac{1}{(1-x)^{r+2}}$. Разложение (4.9) позволяет записать последнее выражение в виде:

$$B(x) = \frac{1}{(1-x)^{r+2}} = \sum_{k=0}^{\infty} C_{r+k+1}^{k} x^k,$$

отсюда

$$b_k = C_{r+k+1}^{k} = \sum_{i=0}^{k} a_i = \sum_{i=0}^{k} C_{k+i}^{i}.$$

Определение. Пусть $X$ — целочисленная случайная величина и определен ее ряд распределения. *Характеристической функцией* распределения случайной величины (с. в.) $X$ называется функция

$$g_X(s) = \sum_{k=0}^{\infty} \mathsf{P}(X = k) s^k.$$

Таким образом, $g_X(s)$ — это производящая функция последовательности чисел $\mathsf{P}(X = k)$, где $k = 0, 1, \ldots$ .

Задача. Пусть $X$ и $Y$ — независимые целочисленные случайные величины и определены их ряды распределений и пусть $g_X(s) = \sum_{k=0}^{\infty} \mathsf{P}(X = k) s^k$ и $g_Y(s) = \sum_{k=0}^{\infty} \mathsf{P}(Y = k) s^k$ — их характеристические функции. Рассмотрим с. в. $Z = X + Y$. Очевидно, что

$$\mathsf{P}(Z = k) = \mathsf{P}(X + Y = k) = \sum_{i=0}^{k} \mathsf{P}(X = i)\, \mathsf{P}(Y = k - i).$$

Согласно свойству свертки производящих функций (4.1.7) характеристическая функция с. в. $Z$ может быть записана таким образом:

$$g_Z(s) = \sum_{k=0}^{\infty} \mathsf{P}(Z = k) s^k = g_X(s) g_Y(s).$$

## § 4.2. Линейные рекуррентные соотношения

Рассмотрим последовательность $\{u_n\}$, $n = 0, 1, 2, \ldots$ . Будем говорить, что задано однородное линейное рекуррентное соотношение с постоянными коэффициентами порядка $r$, если для членов последовательности $\{u_n\}$ выполняется равенство

$$u_{n+r} = c_1 u_{n+r-1} + c_2 u_{n+r-2} + \ldots + c_r u_n, \qquad (4.10)$$

где $c_1, c_2, \ldots, c_r$ — постоянные величины. Выражение (4.10) позволяет вычислить очередной член последовательности по

предыдущим $r$ членам. Ясно, что, задав начальные значения $u_0, u_1, \ldots, u_{r-1}$, можно последовательно определить все члены последовательности. Мы рассмотрим общий метод решения (т. е. поиска $u_n$ как функции от $n$ ) рекуррентного соотношения. Для решения задачи достаточно найти производящую функцию

$$U(x) = \sum_{k=0}^{\infty} u_k x^k \tag{4.11}$$

последовательности $\{u_n\}$. Введем обозначение для полинома

$$K(x) = 1 - c_1 x - c_2 x^2 - \ldots - c_r x^r$$

и рассмотрим произведение

$$U(x)K(x) = C(x).$$

Непосредственным умножением можно убедиться, что $C(x)$ — это полином, степень которого не превышает $r-1$, так как коэффициенты при $x^{n+r}$ $(n = 0, 1, \ldots)$ в $U(x)K(x)$, согласно уравнению (4.10), удовлетворяют соотношению $u_{n+r} - (c_1 u_{n+r-1} + c_2 u_{n+r-2} + \ldots + c_r u_n) = 0$.

Характеристическим полиномом соотношения (4.10) называется

$$F(x) = x^r - c_1 x^{r-1} - c_2 x^{r-2} - \ldots - c_{r-1} x - c_r. \tag{4.12}$$

Выполним разложение $F(x)$ на линейные множители

$$F(x) = (x - \alpha_1)^{e_1} (x - \alpha_2)^{e_2} \ldots (x - \alpha_r)^{e_r},$$

где $e_1 + e_2 + \ldots + e_r = r$. Сравнивая $K(x)$ и $F(x)$, запишем $K(x) = x^r F(1/x)$. Отсюда

$$K(x) = x^r \left(\frac{1}{x} - \alpha_1\right)^{e_1} \left(\frac{1}{x} - \alpha_2\right)^{e2} \cdot \ldots \cdot \left(\frac{1}{x} - \alpha_r\right)^{er} =$$
$$= (1 - \alpha_1 x)^{e_1} (1 - \alpha_2 x)^{e_2} \cdot \ldots \cdot (1 - \alpha_r x)^{e_r}, \quad e_1 + e_2 + \ldots + e_r = r.$$

Разложение на множители используем для представления

$$U(x) = \frac{C(x)}{K(x)}$$

в виде суммы простых дробей:

$$U(x) = \frac{C(x)}{K(x)} = \sum_{i=1}^{r} \sum_{n=1}^{e_i} \frac{\beta_{in}}{(1 - \alpha_i x)^n}. \tag{4.13}$$

Таким образом, $U(x)$ является суммой функций вида

$$\frac{\beta}{(1-\alpha x)^n} = \beta \sum_{k=0}^{\infty} \binom{n+k-1}{k} \alpha^k x^k.$$

Тогда выражение (4.13) примет вид

$$U(x) = \frac{C(x)}{K(x)} = \sum_{i=1}^{r} \sum_{n=1}^{e_i} \beta_{in} \sum_{k=0}^{\infty} \binom{n+k-1}{k} \alpha_i^k x^k. \qquad (4.14)$$

Данное разложение является производящей функцией $U(x) = \sum\limits_{n=0}^{\infty} u_n x^n$ последовательности $\{u_n\}$. Для определения $u_n$ необходимо найти коэффициент при $x^n$ в разложении (4.14).

З а д а ч а. Найти последовательность $\{u_n\}$, удовлетворяющую рекуррентному соотношению $u_{n+2} = 5u_{n+1} - 6u_n$, $u_0 = u_1 = 1$.

Р е ш е н и е. Решение. $K(x) = 1 - 5x + 6x^2$, $K(x)U(x) = C(x)$. Выполним данное умножение: $(1 - 5x + 6x^2)(u_0 + u_1 x + u_2 x^2 + \ldots) = u_0 + (u_1 - 5u_0)x + (u_2 - 5u_1 + 6u_0)x^2 + \ldots = u_0 + (u_1 - 5u_0)x = 1 - 4x$. Таким образом, $C(x) = 1 - 4x$. Характеристический полином $f(x) = x^2 - 5x + 6 = (x-2)(x-3)$. Отсюда $U(x) = \dfrac{C(x)}{K(x)} = \dfrac{1-4x}{(1-2x)(1-3x)} = \dfrac{A}{1-2x} + \dfrac{B}{1-3x}$. Значения величин $A$ и $B$ находим методом неопределенных коэффициентов: $A = 2$, $B = -1$. Наконец, принимая во внимание (4.2),

$$U(x) = \frac{2}{1-2x} + \frac{-1}{1-3x} = 2\sum_{k=0}^{\infty} 2^k x^k - \sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} (2^{k+1} - 3^k) x^k.$$

С другой стороны, $U(x) = \sum\limits_{k=0}^{\infty} u_k x^k$ . Сравнивая коэффициенты при одинаковых степенях $x^k$, заключаем, что $u_k = 2^{k+1} - 3^k$.

## § 4.3. Неоднородные линейные рекуррентные соотношения

Неоднородное линейное рекуррентное соотношение имеет вид

$$u_{n+r} = c_1 u_{n+r-1} + c_2 u_{n+r-2} + \ldots + c_r u_n + b_n, \qquad (4.15)$$

где величина $b_n$ в общем случае является функцией от $n$. Общее решение соотношения (4.15) представляет собой сумму частного

решения неоднородного уравнения (4.15) (т. е. любого решения, которое удовлетворяет (4.15)) и общего решения соответствующего ему однородного соотношения (4.10), которое находится рассмотренным способом. Общих способов определения частного решения нет, однако для специальных значений $b_n$ существуют стандартные приемы определения $u_n$. Рассмотрим на примере в некотором роде универсальную процедуру, которая позволяет сразу находить общее решение неоднородного уравнения (4.15).

Задача. Найти $\{u_n\}$, если известно, что

$$u_{n+1} = u_n + (n+1) \text{ и } u_0 = 1.$$

Решение. Умножив левую и правую части рекуррентного соотношения на $x_n$, получим

$$u_{n+1}x^n = u_n x^n + (n+1)x^n.$$

Суммирование последнего уравнения для всех $n$ дает

$$\sum_{n=0}^{\infty} u_{n+1}x^n = \sum_{n=0}^{\infty} u_n x^n + \sum_{n=0}^{\infty}(n+1)x^n.$$

Свойства операций с производящими функциями позволяют данное выражение привести к виду

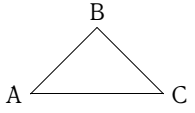
$$\frac{U(x) - u_0}{x} = U(x) + \frac{1}{(1-x)^2}.$$

Учитывая, что $u_0 = 1$, запишем

$$U(x) = \frac{1 - x + x^2}{(1-x)^3} = (1 - x + x^2)\sum_{n=0}^{\infty} C_{n+2}^2 x^n = \sum_{n=0}^{\infty} u_n x^n.$$

Сравнивая коэффициенты при $x^n$, заключаем, что

$$u_n = C_{n+2}^2 - C_{n+1}^2 + C_n^2 = \frac{n^2 + n + 2}{2}.$$




Задача. Найти число замкнутых маршрутов длины $n$ по ребрам треугольника $ABC$. Длину ребра принять равной единице. Начальная и конечная точка маршрутов — вершина $A$.

Решение. Введем обозначения: $a_n$ — число замкнутых маршрутов длины $n$ из вершины $A$ в вершину $A$; $b_n$ — число

маршрутов длины $n$ из вершины $A$ в вершину $B$; $c_n$ — число маршрутов длины $n$ из вершины $A$ в вершину $C$.

Очевидно, что из условия симметрии задачи $b_n = c_n$. Величины же $a_n, b_n, c_n$ связаны системой рекуррентных соотношений:

$$\begin{cases} a_{n+1} = b_n + c_n, \\ b_{n+1} = a_n + b_n, \\ c_{n+1} = a_n + c_n, \\ b_n = c_n. \end{cases}$$

В виду того, что $b_n = c_n$, система приводится к виду

$$\begin{cases} a_{n+1} = 2b_n, \\ b_{n+1} = a_n + b_n, \end{cases}$$

Выражая $b_n$ из первого уравнения и подставляя во второе, получим однородное рекуррентное соотношение относительно последовательности $\{a_n\}$. Запишем его в принятых обозначениях, полагая $u_n = a_n$:

$$u_{n+2} = u_{n+1} + 2u_n, \quad u_0 = 1, u_1 = 0, \quad \text{где} \quad n = 0, 1, 2, \dots .$$

Решение данного соотношения найдем согласно изложенной выше теории:

$$K(x) = 1 - x - 2x^2, \qquad U(x)K(x) = C(x) = 1 - x.$$

Характеристический полином

$$F(x) = x^2 - x - 2 = (x - 2)(x + 1).$$

Отсюда

$$U(x) = \frac{1 - x}{(1 - 2x)(1 + x)} = \frac{1/3}{1 - 2x} + \frac{2/3}{1 + x}.$$

Перепишем $U(x)$ в развернутом виде по степеням $x^n$:

$$U(x) = \frac{1}{3}\sum_{n=0}^{\infty} 2^n x^n + \frac{2}{3}\sum_{n=0}^{\infty}(-1)^n x^n = \\ = \sum_{n=0}^{\infty} \frac{2^n + (-1)^n 2}{3} x^n = \sum_{n=0}^{\infty} u_n x^n.$$

Сравнивая коэффициенты при $x^n$, заключаем, что число замкнутых маршрутов длины $n$ равно $u_n = \dfrac{2^n + (-1)^n 2}{3}$.

## § 4.4. Обобщенное правило произведения

Пусть $S_1, S_2, \ldots, S_n$ — произвольные множества и пусть $\Omega = \{\omega_1, \omega_2, \ldots\}$ — множество весов, где под $\omega_i$, будем понимать любой из символов $1, x, y, z, x^{-1}, y^{-1}, z^{-1}$ и их произведения. В отличие от элементов, вес — это есть число либо переменная, которая может принимать любые числовые значения. Назначим каждому элементу $s \in S_i$ $(i = 1, 2, \ldots, n)$ вес $\omega(s) \in \Omega$. Во многих задачах требуется определить количество элементов с определенными свойствами, а не их вес. В этом случае полагают $\omega(s) = 1$.

Пусть $C_{i\omega}$ — количество элементов множества $S_i$ с весом $\omega$, тогда $A(S_i) = \sum\limits_{\omega\in\Omega} C_{i\omega}\omega$ — сумма весов элементов множества $S_i$. Рассмотрим прямое произведение множеств

$$S = S_1 \times S_2 \times \ldots \times S_n = \{\varepsilon = (s_1, s_2, \ldots, s_n) \mid s_i \in S_i,\ i = \overline{1, n}\}.$$

Положим вес элемента множества равным

$$\omega(\varepsilon) = \omega(s_1, s_2, \ldots, s_n) = \omega(s_1)\omega(s_2) \cdot \ldots \cdot \omega(s_) = \prod_{i=1}^{n} \omega(s_i) \in \Omega.$$

Пусть $E_\omega$ — количество элементов $\varepsilon \in S$ с весом $\omega$, тогда $A(S) = \sum\limits_{\omega\in\Omega} E_\omega\omega$ — сумма весов элементов множества $S$.

Теорема 4.4.1. $A(S) = A(S_1)A(S_2) \cdot \ldots \cdot A(S_n)$.

Доказательство.

$$A(S_i) = \sum_{\omega\in\Omega} C_{i\omega}\omega = \sum_{s\in S_i} \omega(s_i).$$

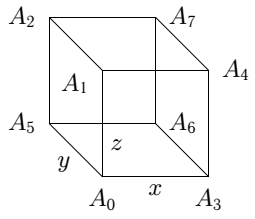
$$A(S) = \sum_{\omega\in\Omega} E_\omega\omega = \sum_{s_1\in S_1} \sum_{s_2\in S_2} \cdot \ldots \cdot \sum_{s_n\in S_n} \omega(s_1, s_2, \ldots, s_n) =$$

$$= \sum_{s_1\in S_1} \sum_{s_2\in S_2} \cdot \ldots \cdot \sum_{s_n\in S_n} \omega(s_1)\omega(s_2) \cdot \ldots \cdot \omega(s_n) =$$

$$= \left(\sum_{s_1\in S_1} \omega(s_1)\right)\left(\sum_{s_2\in S_2} \omega(s_2)\right) \cdot \ldots \cdot \left(\sum_{s_n\in S_n} \omega(s_n)\right) =$$

$$= A(S_1)A(S_2) \cdot \ldots \cdot A(S_n). \qquad \square$$

Задача. Найти количество замкнутых маршрутов длины $2n$ по ребрам трехмерного куба. Начальное и конечное положение — вершина $A_0$ куба.




Р е ш е н и е. Исходное положение — вершина $A_0$. Каждый шаг перемещения по кубу — это выбор одного из трех ребер. Пусть 1 означает выбор ребра вдоль оси $OX$, 2 — вдоль оси $OY$, 3 — вдоль оси $OZ$. В соответствии с этим введем множества $S_1, S_2, \ldots, S_{2n}$ такие, что $S_1 = S_2 = \ldots = S_{2n} = \{1, 2, 3\}$. Тогда все маршруты длины $2n$ составят множество $S = S_1 \times S_2 \times \ldots \times S_{2n}$. Например, маршрут 2113 означает, что из $A_0$ пошли в $A_5$, затем в $A_6$, вернулись в $A_5$ и поднялись в $A_2$.

Назначим веса элементам множества: $\omega(1) = x$, $\omega(2) = y$, $\omega(3) = z$. По правилу обобщенного произведения сумма весов всех маршрутов длины $2n$ равна $A(S) = A(S_1)A(S_2) \cdot \ldots \cdot A(S_{2n}) = (x + y + z)^{2n}$, так как $A(S_i) = x + y + z$.

После возведения в степень получим, что

$$A(S) = (x + y + z)^{2n} = \sum_i \sum_j \sum_k a_{ijk} x^i y^j z^k, \tag{4.16}$$

где $x^i y^j z^k$ — вес маршрута, включающего $i$ шагов вдоль оси $OX$, $j$ — вдоль $OY$ и $k$ — вдоль $OZ$ $(i + j + k = 2n)$, $a_{ijk}$ — количество маршрутов с весом $x^i y^j z^k$.

Заметим, что только маршрут, заканчивающийся в $A_0$, имеет вес $x^i y^j z^k$ с четными степенями $i, j, k$, так как в замкнутом маршруте, сделав шаг вдоль оси, необходимо вернуться по этой оси. Для выделения таких маршрутов положим $x^2 = y^2 = z^2 = 1$ ($x, y, z$ — произвольные). Тогда выражение (4.16) примет вид

$$(x + y + z)^{2n} = C_0 + C_2 yz + C_4 xz + C_6 xy, \tag{4.17}$$

где $C_0$ — число маршрутов, заканчивающихся в $A_0$; $C_2$ — в $A_2$; $C_4$ — в $A_4$; $C_6$ — в $A_6$. Учитывая симметрию точек $A_2, A_4, A_6$ относительно $A_0$, можно заключить, что $C_2 = C_4 = C_6 = C$. Тогда из системы (4.17) следует, что

$$(x + y + z)^{2n} = C_0 + C(yz + xz + xy). \tag{4.18}$$

Уравнение (4.18) содержит два неизвестных, $C_0$ и $C$, при $x^2 = y^2 = z^2 = 1$. Для их определения составим систему уравнений из выражения (4.18), полагая $x = y = z = 1$ и $x = y = 1, z = -1$. Это возможно, так как выполняется условие $x^2 = y^2 = z^2 = 1$.

Получим систему

$$\begin{cases} (1+1+1)^{2n} = C_0 + C(1+1+1) \\ (1+1-1)^{2n} = C_0 + C(-1-1+1) \end{cases} \text{ или } \begin{cases} 3^{2n} = C_0 + 3C, \\ 1 = C_0 - C. \end{cases}$$

Отсюда $C_0 = \dfrac{3^{2n}+3}{4}$ — число искомых маршрутов.

З а д а ч а. Найти число решений $p, q$ уравнения

$$2p + 3q = n, \text{ где } n, p, q \in \{0, 1, 2, \ldots\}.$$

Р е ш е н и е. Введем множества: $S_1 = \{2p \,|\, p = 0, 1, 2, \ldots \ldots\}$, $S_2 = \{3q \,|\, q = 0, 1, 2, \ldots\}$ и $S = S_1 \times S_2 = \{(2p, 3q) \,|\, p, q = = 0, 1, 2, \ldots)\}$. Назначим веса элементам множеств следующим образом:

$$\omega(2p) = x^{2p}, \quad \omega(3q) = x^{3q}$$

и

$$\omega(\varepsilon) = \omega((2p, 3q)) = x^{2p+3q=n} = x^{2p}x^{3q} = \omega(2p)\omega(3q).$$

Веса элементов определены таким образом, что выполняется правило обобщенного произведения $A(S) = A(S_1)A(S_2)$. Легко заметить, что степень веса $x^{2p+3q=n}$ любого элемента $\varepsilon = (2p, 3q) \in S$ дает одно из решений исходного уравнения $2p + 3q = n$. Раскроем выражение $A(S) = A(S_1)A(S_2)$:

$$u_0 + u_1 x + u_2 x^2 + \ldots + u_n x^n + \ldots =$$
$$= (x^{2\cdot 0} + x^{2\cdot 1} + \ldots + x^{2\cdot p} + \ldots)(x^{3\cdot 0} + x^{32\cdot 1} + \ldots + x^{3\cdot q} + \ldots) =$$
$$= (1 + x^2 + x^4 + \ldots)(1 + x^3 + x^6 + \ldots) = \frac{1}{1-x^2} \cdot \frac{1}{1-x^3},$$

где $u_n$ — число решений уравнения $2p + 3q = n$. Фактически, сумма весов $A(S)$ элементов множества $S$ является производящей функцией числа решений уравнения $2p + 3q = n$. Таким образом, $A(S) = \dfrac{1}{1-x^2} \cdot \dfrac{1}{1-x^3}$. Разложим данное выражение на множители:

$$A(S) = \frac{1/4}{1+x} + \frac{1/6}{(1-x)^2} + \frac{7/12 - x/12 + x^2/4}{1-x^3} =$$
$$= \frac{1}{12}\left(\frac{3}{1+x} + \frac{2}{(1-x)^2} + \frac{7 - x + 3x^2}{1-x^3}\right) =$$

$$= \frac{1}{12}\left(3\sum_{k=0}^{\infty}(-1)^k x^k + 2\sum_{k=0}^{\infty}(k+1)^k x^k + (7-x+3x^2)\sum_{k=0}^{\infty}x^{3k}\right) =$$

$$= \sum_{k=0}^{\infty} u_k x^k,$$

где $u_k$ — число решений уравнения $2p + 3q = n$.

Сравнивая коэффициенты при одинаковых степенях, получим:

$$u_{3n} = \frac{3(-1)^{3n} + 2(3n+1) + 7}{12} = \frac{2n + 3 + (-1)^{3n}}{4},$$

$$u_{3n+1} = \frac{3(-1)^{3n+1} + 2(3n+2) - 1}{12} = \frac{2n + 1 - (-1)^{3n}}{4},$$

$$u_{3n+2} = \frac{3(-1)^{3n+2} + 2(3n+3) + 3}{12} = \frac{2n + 3 + (-1)^{3n}}{4},$$

где $n \in \{0, 1, 2, \ldots\}$.

## § 4.5. Принцип включения и исключения

Введем обозначения:

$S = \{s_1, s_2, s_3, \ldots\}$ — произвольное множество элементов.

$\Omega = \{\omega_1, \omega_2, \omega_3, \ldots\}$ — множество весов, $\omega(s_k)$ — вес элемента $s_k \in S$.

$P_1, P_2, P_3, \ldots, P_n$ — свойства элементов или индикаторы свойств элементов.

$P_i(s_k) = \begin{cases} 1, & \text{если элемент } s_k \text{ обладет свойством } P_i; \\ 0 & \text{— в противном случае.} \end{cases}$

$W(P_i) = \sum\limits_{s_k \in S} P_i(s_k)\omega(s_k)$ — веса элементов со свойством $P_i$.

$W(P_{i_1}P_{i_2} \cdot \ldots \cdot P_{i_m})$ — сумма весов элементов множества $S$, которые обладают каждым из свойств $P_{i_1}, P_{i_2}, \ldots, P_{i_m}$.

$W(m) = \sum\limits_{(i_1 i_2 \ldots i_m)} W(P_{i_1}P_{i_2} \cdot \ldots \cdot P_{i_m})$, суммирование выполняется по всем сочетаниям $(i_1 i_2 \ldots i_m)$ длины $m$ из $n$ свойств, количество сочетаний равно $C_n^m$.

Таким образом, в $W(m)$ суммируются веса только тех элементов, которые имеют как минимум $m$ свойств. Пусть элемент

$s$ обладает $k$ свойствами и $k \geqslant m$, тогда его вес $\omega(s)$ в $W(m)$ войдет $C_k^m$ раз. Так,

$$W(1) = \sum_{(i_1)} W(P_{i_1}) = W(P_1) + W(P_2) + \ldots + W(P_n)$$

содержит $C_n^1 = n$ слагаемых и

$$W(2) = \sum_{(i_1 i_2)} W(P_{i_1} P_{i_2}) = W(P_1P_2) + W(P_1P_3) + \ldots + W(P_{n-1}P_n)$$

содержит $C_n^2 = n(n-1)/2$ слагаемых.

Распространим определение $W(m)$ на $m = 0$, положив $W(0) = \sum_{s \in S} \omega(s)$ — суммы весов элементов исходного множества $S$. Данное определение $W(0)$ выполнено корректно, так как сумма должна включать элементы, обладающие нуль свойствами и более. Действительно, любой из элементов множества $S$ удовлетворяет этим условиям.

Положим:

$E(m)$ — веса элементов, обладающих ровно $m$ свойствами;

$E(0)$ — веса элементов, которые не имеют ни одного из указанных свойств.

Теорема 4.5.1. *Сумма весов элементов, обладающих точно $m$ свойствами из $n$ свойств $P_1, P_2, \ldots, P_n$, равна*

$$E(m) = W(m) - C_{m+1}^m W(m+1) + \ldots + (-1)^{n-m} C_n^m W(n))$$

*или*

$$E(m) = \sum_{i=0}^{n-m} (-1)^i C_{m+i}^m W(m+i). \tag{4.19}$$

Доказательство. Достаточно показать, что вклад веса $\omega(s)$ произвольного элемента $s \in S$ в правую и левую части (4.19) одинаковый. Пусть $s \in S$ обладает точно $t$ свойствами. Возможны следующие соотношения между $t$ и $m$:

1) $t<m$, тогда $\omega(s)$ не входит в $E(m)$ и не входит в $W(m+i)$ для $i = 0, 1, \ldots, n-m$. Равенство (4.19) примет вид 0=0;

2) $t = m$, тогда $\omega(s)$ входит один раз в $E(m)$ и в $W(m)$;

3) $t > m$, тогда $\omega(s)$ не входит в $E(m)$ и левая часть 4.19 равна 0. Покажем, что в этом случае и правая часть рассматриваемого выражения равна нулю.

Вес $\omega(s)$ в $W(m+i)$ входит $C_t^{m+i}$ раз, $m + i \leqslant t$. Правая часть (4.19) для веса $\omega(s)$ одного элемента примет вид

$$C_m^m\omega(s)C_t^m - C_{m+1}^m\omega(s)C_t^{m+1} + \ldots + (-1)^{t-m}C_t^m\omega(s)C_t^t = \\ = \omega(s)\sum_{i=0}^{t-m}(-1)^i C_{m+i}^m C_t^{m+i}.$$

Заметим, что

$$C_{m+i}^m C_t^{m+i} = \frac{(m+i)!}{m!\,i!}\frac{t!}{(m+i)!\,(t-(m+i))!} = \\ = \frac{t!}{m!\,(t-m)!}\cdot\frac{(t-m)!}{i!\,((t-m)+i)!}.$$

Значит, $C_{m+i}^m C_t^{m+i} = C_t^m C_{t-m}^i$, а исходная сумма составит

$$\omega(s)\sum_{i=0}^{t-m}(-1)^i C_{m+i}^m C_t^{m+i} = \omega(s)\sum_{i=0}^{t-m}(-1)^i C_t^m C_{t-m}^i = \\ = \omega(s)C_t^m\sum_{i=0}^{t-m}(-1)^i C_{t-m}^i = \omega(s)C_t^m(1-1)^{t-m} = 0.$$

И в третьем случае выражение (4.19) справедливо. □

Следствие. $E(0)=W(0)-W(1)+W(2)-\ldots+(-1)^n W(n)$.

Задача. В группе 23 студента. Из них 18 знают английский язык, 9 — немецкий и 6 — оба языка. Сколько студентов в группе не знают ни одного языка? Сколько студентов знают один язык?

Решение. Пусть $S$ множество всех студентов $|S| = 23$. Назначим вес $\omega(s) = 1$ всем элементам $s \in S$. Теперь под суммой весов будем понимать количество студентов.

Назначим свойства элементам $s \in S$: $P_1$ — знание английского языка, $P_2$ — знание немецкого языка. $E(0)$ — студенты, не знающие языков (не обладают указанными свойствами):

$$E(0) = W(0) - W(1) + W(2),$$
$$W(0) = \sum_{s\in S}\omega(s) = 23,$$
$$W(1) = W(P_1) + W(P_2) = 18 + 9 = 27,$$
$$W(2) = W(P_1P_2) = 6.$$

Тогда $E(0) = 23 - 27 + 6 = 2$.

$E(1)$ — студенты со знанием ровно (только) одного языка:

$$E(1) = C_1^1 W(1) - C_2^1 W(2) = 1 \cdot (18 + 9) - 2 \cdot 6 = 15.$$

Задача. Найти число перестановок $m$ шаров, в которых ровно $r$ шаров остаются на месте.

Решение. Введем обозначения. $P_i$ — свойство, состоящее в том, что при перестановке шаров $i$-й шар остается на месте, $i = 1, 2, \ldots, m$. $E(r)$ — количество перестановок, обладающих ровно $r$ свойствами, т. е. при перестановке шаров ровно $r$ из них остаются на своих места. Тогда по формуле включений и исключений запишем: $E(r) = \sum\limits_{i=0}^{m-r} (-1)^i C_{r+i}^i W(r+i)$. Рассмотрим $W(r) = \sum\limits_{(i_1 i_2 \ldots i_r)} W(P_{i_1} P_{i_2} \cdot \ldots \cdot P_{i_r})$, где суммирование выполняется по всем сочетаниям $(i_1 i_2 \ldots i_r)$ длины $r$ из $m$ свойств, количество сочетаний равно $C_m^r$. $W(P_{i_1} P_{i_2} \cdot \ldots \cdot P_{i_r}) = (m-r)!$ — количество перестановок из $m - r$ шаров, так как $r$ шаров должны оставаться на месте. Тогда значение $W(r) = C_m^r (m - r)!$. Следовательно, и $E(r) = \sum\limits_{i=0}^{m-r} (-1)^r C_{r+i}^r C_m^{r+i} (m-r-i)!$ или $E(r) = \dfrac{m!}{r!} \sum\limits_{r=0}^{m-r} \dfrac{(-1)^i}{i!} \simeq \dfrac{m!\, e^{-1}}{r!}$.

## § 4.6. Ладейные многочлены и многочлены попаданий

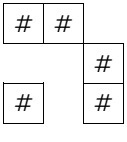


Рис. 4.1

Основная цель данного подраздела — показать возможности применения методов подсчета и оценивания при решении конкретных задач. В то же время обсуждаемые здесь задачи сами по себе не лишены интереса и в большей степени для тех, кто впервые знакомится ними.

### 4.6.1. Ладейные многочлены

Определение. *Доской запрещенных позиций* называется произвольный набор выделенных клеток шахматной доски, сохраняющих свое расположение относительно других клеток доски. Пример доски запрещенных позиций показан на рис. 4.1.

Определение. Пусть $C$ — доска запрещенных позиций. Введем обозначения: $r_k$ — количество способов расставить $k$ ладей на доске запрещенных позиций так, чтобы они не били

друг друга; $r_0 = 1$ — количество способов расстановки 0 ладей на доске запрещенных позиций, т. е. способов не ставить ладьи на доску. Ладьи считаются неразличимыми. Производящую функцию последовательности $\{r_k\}$ будем называть *ладейным многочленом* доски $C$ и обозначать

$$R(x, C) = \sum_{k=0}^{\infty} r_k x^k. \tag{4.20}$$

Задача. Найти ладейный многочлен для доски $C_1$ с одной клеткой (рис. 4.2).


Рис. 4.2

Решение. Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 1$ и $r_i = 0$, $i \geqslant 2$. Тогда $R(x, C_1) = 1 + x$.

Задача. Найти ладейный многочлен доски $C_2$ на рис. 4.3.


Рис. 4.3

Решение. Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 2$, $r_2 = 1$ и $r_i = 0$, $i \geqslant 3$. Тогда $R(x, C_2) = 1 + 2x + x^2$.

Задача. Найти ладейный многочлен доски $C_3$ на рис. 4.4.


Рис. 4.4

Решение. Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 3$, $r_2 = 1$ и $r_i = 0$, $i \geqslant 3$. Тогда $R(x, C_3) = 1 + 3x + x^2$.

Определение. Доски $C_1$ и $C_2$ называются *независимыми*, если их клетки располагаются на различных горизонталях и вертикалях. Пример независимых досок показан на рис. 4.5.

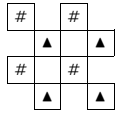



Рис. 4.5

### Свойства ладейных многочленов

Свойство 1. *Правило произведения. Пусть $R(x, C_1)$ и $R(x, C_2)$ — ладейные многочлены независимых досок $C_1$ и $C_2$. Если доска $C = C_1 \cup C_2$, то $R(x, C) = R(x, C_1)R(x, C_2)$.*

Доказательство. Пусть $\pi_1$— расстановка ладей на доске $C_1$, $\pi_2$ — расстановка ладей на доске $C_2$. Тогда $\pi = (\pi_1, \pi_2)$ — перестановка ладей на доске $C = C_1 \cup C_2$. Верно и обратное. Пусть $S_{\pi_1}$ и $S_{\pi_2}$ — множество расстановок ладей на доске, соответственно $C_1$ и $C_2$. Тогда прямое произведение $S_\pi = S_{\pi_1} \times S_{\pi_1}$ — множество расстановок ладей на доске $C$.

Обозначим веса расстановок: $\omega(\pi_1) = x^{k_1}$, где $k_1$ — число ладей в перестановке $\pi_1$; $\omega(\pi_2) = x^{k_2}$, где $k_2$ — число ладей в перестановке $\pi_2$. Тогда вес перестановки $\pi$ равен $\omega(\pi) = x^{k_1+k_2} = x^{k_1}x^{k_2} = \omega(\pi_1)\omega(\pi_2)$.

В соответствии с тем, как введены веса перестановок, ладейные многочлены можно записать как сумму весов элементов множеств:

$$R(C_1) = \sum_{\pi_1 \in S_{\pi_1}} \omega(\pi_1),\ R(C_2) = \sum_{\pi_2 \in S_{\pi_2}} \omega(\pi_2),\ R(C) = \sum_{\pi \in S_\pi} \omega(\pi).$$

Многочлены $R(x, C_1)$, $R(x, C_2)$ и $R(x, C)$ удовлетворяют всем условиям правила обобщенного произведения (см. п. 4.4), в соответствии с которым $R(x, C) = R(x, C_1)R(x, C_2)$. □

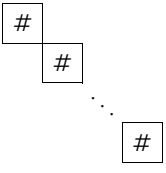


Рис. 4.6

Задача. Найти ладейный многочлен доски $C$ запрещенных позиций из $n$ клеток (рис. 4.6).

Решение. Пусть $C_1$ — доска из одной клетки, $R(x, C_1) = 1 + x$. Ясно, что доска $C$ состоит из $n$ независимых досок $C_1$. Отсюда следует, что $R(x, C) = (R(x, C_1))^n = (1 + x)^n$.

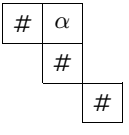


Рис. 4.7

Свойство 2. Правило суммы. *Пусть $C$ — доска запрещенных позиций. Введем обозначения*: $C_{i_\alpha}$ — *доска с ладьей в клетке $\alpha$* ($i$ — *index*); $C_{e_\alpha}$ — *доска с удаленной клеткой $\alpha$* ($e$ — *erase*). *Тогда* $R(x, C) = xR(x, C_{i_\alpha}) + R(x, C_{e_\alpha})$ *или* $R(xi_\alpha + r_\alpha) = xR(i_\alpha) + R(e_\alpha)$ (рис. 4.7).

Доказательство. Для доказательства данного свойства вновь рассмотрим ладейный многочлен как сумму весов перестановок. Все расстановки множества разобьем на два подмножества $S_{\pi_{i_\alpha}}$ и $S_{\pi_{e_\alpha}}$, где $S_{\pi_{i_\alpha}}$ — перестановки с ладьей на клетке $\alpha$; $S_{\pi_{e_\alpha}}$ — перестановки, которые не занимают клетку $\alpha$. Тогда

$$R(x, C) = \sum_{\pi \in S_\pi} \omega(\pi) = \sum_{\pi_{i_\alpha} \in S_{\pi_{i_\alpha}}} x\omega(\pi_{i_\alpha}) + \sum_{\pi_{e_\alpha} \in S_{\pi_{e_\alpha}}} \omega(\pi_{e_\alpha}) =$$

$$= x\Big(\sum_{\pi_{i_\alpha} \in S_{\pi_{i_\alpha}}} \omega(\pi_{i_\alpha})\Big) + \sum_{\pi_{e_\alpha} \in S_{\pi_{e_\alpha}}} \omega(\pi_{e_\alpha}) =$$

$$= xR(x, C_{i_\alpha}) + R(x, C_{e_\alpha}).$$

Множитель $x$ перед скобкой — это вес ладьи в перестановке $\pi_{i_\alpha}$, которая поставлена на выделенную клетку $\alpha$. □

Задача. Найти $R(x, C)$ для доски на рис. 4.7.

Решение. Воспользуемся правилом суммы, по которому $R(x, C) = xR(i_\alpha) + R(e_\alpha) = x(1+x) + (1+x)^3 = 1 + 4x + 4x^2 + x^3$. Отметим, что по виду полученного ладейного многочлена (производящей функции) можно сказать, что число способов расставить две ладьи на доску $C$ равно 4.

Свойство 3 для прямоугольных досок. *Пусть $C$ — прямоугольная доска запрещенных позиций размером $m \times n$, $m$ — число горизонталей; $n$ — число вертикалей (рис. 4.8). На квадратной доске размером $k \times k$ можно $k!$ способами расставить $k$ ладей. Различных досок $k \times k$ на доске $m \times n$ можно выбрать способами $C_m^k C_n^k$. Отсюда $r_k = C_m^k C_n^k k!$ — число способов расставить $k$ ладей на исходной прямоугольной доске. Тогда* $R(x, C) = \sum\limits_{k=0}^{\min(m,n)} C_m^k C_n^k k!\, x^k$.

| | | |
|---|---|---|
| # | # | # |
| # | # | # |

Рис. 4.8

Задача. Найти $R(x, C)$ для прямоугольной доски $C$ размером $2 \times 3$ (рис. 4.8).

Решение. $R(x, C) = \sum\limits_{k=0}^{2} C_2^k C_3^k k!\, x^k = C_2^0 C_3^0 0!\, x^0 + C_1 2^1 C_3^1 1!\, x^1 + C_2^2 C_3^2 2!\, x^2 = 1 + 6x + 6x^2$. Коэффициент при $x^2$ показывает, что число способов поставить две ладьи на такую доску равно 6.

### 4.6.2. Многочлены попаданий

Определение. Пусть дана квадратная доска размером $n \times n$ с запрещенными позициями (рис. 4.9). Обозначим через $E_n(k)$ количество перестановок на квадратной доске $n$ ладей, $k$ из которых занимают запрещенные позиции. *Многочленом попадания* называется производящая функция $E(t) = \sum\limits_{k=0}^{n} E_n(k) t^k$ последовательности $\{E_n(k)\}$.

| | | |
|---|---|---|
| | # | # |
| | | # |
| # | | |

Рис. 4.9

Установим связь между многочленом попаданий и ладейным многочленом. Введем обозначения:

$S_n = \{\pi_1, \pi_2, \ldots, \pi_{n!}\}$ — множество всех перестановок $n$ ладей на доске $n \times n$;

$\Omega = \{\omega_1, \omega_2, \ldots\}$ — множество весов;

$\omega : S_n \to \Omega$ — весовая функция, $\omega(\pi) \in \Omega$) — вес перестановки $\pi \in S_n$;

$\{1, 2, \ldots, m\}$ — номера запрещенных клеток на доске $n \times n$;

$P_1, P_2, \ldots, P_m$ — свойства перестановок; $\pi \in S_n$ обладает свойством $P_j$, если ее ладья занимает запрещенную позицию $j$;

$$P_j(\pi) = \begin{cases} 1, & \text{если } \pi \text{ обладает войством} P_j, \\ 0, & \text{в противном случае;} \end{cases}$$

$W(P_j) = \sum\limits_{\pi \in S_n} P_j(\pi)\omega(\pi)$ — веса перестановок со свойством $P_j$;

$W(P_{j_1} P_{j_2} \cdot \ldots \cdot P_{j_k}) = \sum\limits_{\pi \in S_n} P_{j_1}(\pi) P_{j_2}(\pi) \cdot \ldots \cdot P_{j_k}(\pi)\omega(\pi)$ — веса перестановок со свойствами $P_{j_1} P_{j_2} \ldots P_{j_k}$;

$W(k) = \sum\limits_{(j_1 j_2 \cdot \ldots \cdot j_k)} W(P_{j_1} P_{j_2} \ldots P_{j_k})$ — суммирование выполняется по всем сочетаниям $(j_1 j_2 \ldots j_k)$ длины $k$ из $m$ свойств, число сочетаний равно $C_m^k$.

Принцип включения и исключения позволяет определить

$$E_n(k) = C_k^k W(k) - C_{k+1}^k W(k+1) + \ldots + \\ + (-1)^{m-k} C_{k+(m-k)}^k W(k + (m-k)).$$

Так как нас интересует лишь количество перестановок, то будем полагать $\omega(\pi) = 1$. Не трудно установить, что $W(k) = r_k(n-k)!$, где $r_k$ — количество способов расставить $k$ ладей на запрещенных позициях доски $n \times n$. Тогда

$$E_n(k) = C_k^k r_k (n-k)! - C_{k+1}^k r_{k+1}(n-(k+1))! + \ldots + \\ + (-1)^{m-k} C_m^k r_m (n-m)!.$$

Заметим, что $r_k = 0$ для всех $k > n$. С другой стороны, если $m < n$, то $r_k = 0$ для всех $k > m$. Поэтому последнее выражение примет вид

$$E_n(k) = C_k^k r_k (n-k)! - C_{k+1}^k r_{k+1}(n-(k+1))! + \ldots + \\ + (-1)^{n-k} C_n^k r_n (n-n)!$$

или

$$E_n(k) = \sum_{i=0}^{n-k} (-1)^i C_{k+i}^i r_{k+i} (n - (k+i))!.$$

Теперь многочлен попаданий можно записать в виде

$$E(t) = \sum_{k=0}^{n} E_n(k) t^k = \sum_{k=0}^{n} \sum_{i=0}^{n-k} (-1)^i C_{k+i}^i r_{k+i} (n - (k+i))! t^k.$$

Суммирование выполняется по координатам узлов сетки на рис. 4.10. Замена переменных суммирования $k' = k + i$ и $i' = i$ в последнем выражении позволяет упростить многочлен попаданий и приводит его к виду

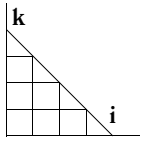


Рис. 4.10

$$E(t) = \sum_{k=0}^{n} \sum_{i=0}^{k} (-1)^i C_k^i r_k (n-k)!\, t^{k-i},$$

где суммирование уже выполняется по узлам сетки на рис. 4.11.

Таким образом,

$$E(t) = \sum_{k=0}^{n} r_k \Big( \sum_{i=0}^{k} (-1)^i C_k^i t^{k-i} \Big) (n-k)!$$

или

$$E(t) = \sum_{k=0}^{n} r_k (t-1)^k (n-k)!.$$

Для более компактной записи последнего выражения введем оператор $\varepsilon^{-1}$, действие которого распространим на функции целочисленного аргумента: $\varepsilon^{-1}(f(n)) = f(n-1)$ и $\varepsilon^{-k}(f(n)) = f(n-k)$. Например,

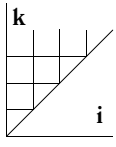


Рис. 4.11

$$\varepsilon^{-1}(n!) = (n-1)! \text{ и } \varepsilon^{-k}(n!) = (n-k)!.$$

Заметим, что $((t-1)\varepsilon^{-1})^k n! = (t-1)^k \varepsilon^{-k} n! = (t-1)^k (n-k)!$. Тогда многочлен попаданий перепишем в виде

$$E(t) = \sum_{k=0}^{n} r_k (t-1)^k (n-k)! = \sum_{k=0}^{n} r_k ((t-1)\varepsilon^{-1})^k n! =$$

$$= \Big( \sum_{k=0}^{n} r_k ((t-1)\varepsilon^{-1})^k \Big) n! = \sum_{k=0}^{n} r_k x^k = R(x) n!,$$

где $x = (t-1)\varepsilon^{-1}$. Таким образом, $E(t) = R((t-1)\varepsilon^{-1})n!$.

З а д а ч а. Найти многочлен попаданий для доски $3 \times 3$ с запрещенными позициями на рис. 4.12. Запрещенные позиции отмечены темным цветом.

Р е ш е н и е. Найдем ладейный многочлен доски запрещенных позиций, которая состоит из двух независимых досок $C_1$ и $C_3$. Тогда

$$R(x) = R(C_1)R(C_3) = (1+x)(1+3x+x^2) = 1+4x+4x^2+x^3.$$

$$\begin{aligned} E(t) &= R((t-1)\varepsilon^{-1})n! = \\ &= (1+4(t-1)\varepsilon^{-1}+4((t-1)\varepsilon^{-1})^2+((t-1)\varepsilon^{-1})^3)3! = \\ &= 1\cdot 3! + 4(t-1)\varepsilon^{-1}3! + 4(t-1)^2\varepsilon^{-2}3! + (t-1)^3\varepsilon^{-3}3! = \\ &= 3! + 4(t-1)2! + 4(t-1)^2 1! + (t-1)^3 0!. \end{aligned}$$

Итак, $E(t) = 1+3t+t^2+t^3$. Анализ коэффициентов $E(t)$ при $t^k$ показывает, что число перестановок, в которых ладьи не занимают запрещенных клеток, равно 1 (коэффициент при $t^0$); перестановок с одной ладьей на запрещенных позициях — 3 (коэффициент при $t^1$); перестановок с двумя ладьями на запрещенных позициях — 1 (коэффициент при $t^2$); перестановок с тремя ладьями на запрещенных позициях — 1 (коэффициент при $t^3$).

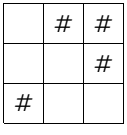


Рис. 4.12

Глава 5

# ГЕНЕРАЦИЯ КОМБИНАТОРНЫХ ОБЪЕКТОВ

В комбинаторных алгоритмах часто необходимо порождать и исследовать все элементы некоторого класса комбинаторных объектов. Наиболее общие методы решения таких задач основываются на поиске с возвращением, однако во многих случаях объекты настолько просты, что целесообразнее применять специализированные методы. Задачи, требующие генерации комбинаторных объектов, возникают при вычислении комбинаторных формул. Например, часто приходится вычислять суммы, имеющие вид

$$\sum f(x_1, x_2, \ldots, x_n),$$

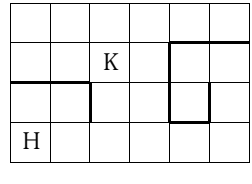
где суммирование выполняется по всем последовательностям $x_1, x_2, \ldots$ $\ldots, x_n$, удовлетворяющим некоторым ограничениям.

В алгоритмах порождения комбинаторных объектов нас прежде всего будет интересовать сложность алгоритмов, т. е. общее количество времени, требующегося для порождения всего множества объектов.

## § 5.1. Поиск с возвращением

Использование компьютера для ответа на такие вопросы, как «сколько существует способов...», «перечислите все возможные...» или «есть ли способ...», обычно требует исчерпывающего поиска множества решений. Метод поиска с возвращением постоянно пытается расширить частичное решение. Если расширение текущего частичного решения невозможно, то возвращаются к более короткому частичному решению и пытаются снова его продолжить.

Идею поиска с возвращением легче всего понять в связи с задачей прохода через лабиринт: цель попасть из некоторого заданного квадрата в другой заданный квадрат путем последовательного перемещения по квадратам. Трудность со-



стоит в том, что существующие преграды запрещают некоторые перемещения. Один из способов прохода через лабиринт — это двигаться из начального квадрата в соответствии с двумя правилами:

- в каждом квадрате выбирать еще неисследованный путь;
- если из исследуемого в данный момент квадрата не ведут неисследованные пути, то нужно вернуться на один квадрат назад по последнему пройденному пути, по которому пришли в данный квадрат.

Первое правило говорит о том, как расширить исследуемый путь, если это возможно, а второе правило — о том, как выходить из тупика. В этом и состоит сущность поиска с возвращением: продолжать расширение исследуемого решения до тех пор, пока это возможно, и когда решение нельзя расширить, возвращаться по нему и пытаться сделать другой выбор на самом близком шаге, где имеется такая возможность.

### Общий алгоритм

В самом общем случае полагаем, что решение задачи состоит из вектора $(a_1, a_2, a_3, \ldots)$ конечной, но неопределенной длины, удовлетворяющего некоторым ограничениям. Каждое $a_i \in A_i$, где $A_i$ — конечное линейно упорядоченное множество. Таким образом, при исчерпывающем поиске должны рассматриваться элементы множества $(a_1, a_2, a_3, \ldots, a_i) \in A_1 \times A_2 \times \ldots \times A_i$ для $i = 0, 1, 2, \ldots$ в качестве возможных решений. В качестве исходного частичного решения принимается пустой вектор $()$ и на основании имеющихся ограничений выясняем, какие элементы из $A_1$ являются кандидатами в $a_1$. Обозначим это подмножество кандидатов через $S_1 \subseteq A_1$. В результате имеем частичное решение $(a_1)$. В общем случае для расширения частичного решения от $(a_1, a_2, a_3, \ldots, a_{k-1})$ до $(a_1, a_2, a_3, \ldots, a_k)$ кандидаты на роль $a_k$ выбираются из $S_k \subseteq A_k$. Если частичное решение $(a_1, a_2, a_3, \ldots, a_{k-1})$ не представляет возможности для выбора элемента $a_k$, то $S_k = \varnothing$; возвращаемся и выбираем новый элемент $a_{k-1}$. Если новый элемент $a_{k-1}$ выбрать нельзя, возвращаемся еще дальше и выбираем новый элемент $a_{k-2}$ и т. д. Этот процесс удобно представлять в терминах прохождения дерева поиска в глубину (см. п. 7.3). Процедура поиска с возвращением для нахождения всех решений формально представлена в алгоритме 5.1.

Алгоритм 5.1.

$S_1 = A_1$; { }
$k = 1$; { }
**while** $k > 0$ **do begin**
  **while** $S_k \neq \varnothing$ **do begin**
    $a_k \in S_k$; { }
    $S_k = S_k - \{a_k\}$; { }
    **if** $(a_1, a_2, \ldots, a_k)$ **then** { }
    $k = k + 1$; { }
    $S_k \subseteq A_k$; { }
  **end;**
  $k = k - 1$; { }
**end.**

Поиск с возвращением приводит к алгоритмам, экспоненциальной сложности, так как из предположения, что все решения имеют длину не более $n$, исследованию подлежат приблизительно $\prod_{k=1}^{n} |A_k|$ элементов. В предположении, что все $|A_k| = C$ — константы, получаем экспоненциальную сложность $\prod_{k=1}^{n} |A_k| = C^n$. Нужно помнить, что поиск с возвращением представляет собой только общий метод. Непосредственное его применение обычно ведет к алгоритмам, время работы которых недопустимо велико. Поэтому, чтобы метод был полезен, к нему нужно относиться как к схеме, с которой следует подходить к задаче. Схема должна быть хорошо приспособлена (часто это требует большой изобретательности) к конкретной задаче, так, чтобы в результате алгоритм годился для практического использования.

## § 5.2. Перестановки различных элементов

Перестановки множества $\{a_1, a_2, \ldots, a_n\}$ различных элементов относятся к часто порождаемым комбинаторным объектам. Без ограничения общности можно полагать, что элементами множества являются целые числа от 1 до $n$, т. е. рассматриваются перестановки целых чисел $\{1, 2, \ldots, n\}$. Порождение перестановок на основе метода поиска с возвращениями выполняется в алгоритме 5.2.

Алгоритм 5.2.

```
s_1 = 1; { }
k = 1; {  }
while k > 1 do begin
  while s_k ⩽ n do begin
    a_k = s_k; {   }
    s_k = s_k + 1; {  }
    while s_k ⩽ n and not flag(s_k) do s_k = s_k + 1;
    if k = n then  (a_1, a_2, ..., a_n)
    else begin
      k = k + 1; {  }
      s_k = 1;
      while s_k ⩽ n and not flag(s_k) do s_k = s_k + 1;
    end;
  end;
  k = k - 1; {    }
end;

function flag(s_k)    s_k    (a_1, a_2, ..., a_{k-1})
  flag = TRUE;
  i = 1;
  while i < k and flag do begin
    if a_i = s_k then flag = FALSE;
    i = i + 1;
  end.
```

Заметим, что в процессе приспосабливания общего алгоритма 5.1 поиска с возвращениями к задаче порождения перестановок мы не вычисляли и не хранили явно множества $S_k$. В этом случае легче и достаточно хранить только наименьшее значение из $S_k$, т. е. $s_k$, и следующее значение вычислять по мере необходимости. Проверка условия $S_k \neq \varnothing$ соответствует условию $s_k \leqslant n$; поскольку алгоритм устроен так, что перебор значений элемента $s_k$ выполняется в порядке их возрастания. Поэтому неравенство $s_k > n$ соответствует пустому множеству кандидатов $S_k = \varnothing$.

Программа на языке **Pascal** реализации рассмотренного метода генерации перестановок приводится в алгоритме 5.3. Следует отметить, что в программе размерность (длина) $n$ перестановок

читается из файла и порождаемые перестановки также сохраняются в файле. Попутно программа вычисляет время порождения всех перестановок с точностью до сотых долей секунд, которое сохраняется в конце файла сгенерированных перестановок.

Алгоритм 5.3. Pascal

```
Program Start_BackTrack; { }
uses CRT,DOS;
Const max_n=20;
Type Vector=array[1..max_n] of LongInt;

Function Flag( Var a:Vector; sk:LongInt; k:Integer ): Boolean;
{                          }
Var
  i :Integer;
  yes :Boolean;
begin;
  yes:=TRUE;
  i:=1;
  while (i<k) and yes do begin
    if a[i]=sk then yes:=FALSE;
    i:=i+1;
  end;
  Flag:=yes;
end;{     }

Procedure BackTrack( var f:Text; Var a:Vector; n:Integer );
{                    }
Const
  m :LongInt=0;{ }
Var
  s :Vector;
  i,k :Integer;
begin;
  for i:=1 to n do s[i]:=0;
  s[1]:=1;
```

```
  k:=1;
  while k>0 do begin
    while s[k]<=n do begin
      a[k]:=s[k];
      repeat {          }
        s[k]:=s[k]+1;
      until (s[k]>n) or Flag(a,s[k],k);
      if k=n then begin { }
        m:=m+1;
        Write(f,m,') '); for i:=1 to n do Write(f,a[i],' '); WriteLn(f);
      end
      else begin{              }
        k:=k+1;
        s[k]:=1;
        while (s[k]<=n) and Not Flag(a,s[k],k) do s[k]:=s[k]+1;
      end;
    end;
    k:=k-1;
  end;
end;{              }


Var {       }
  a :Vector;
  n :Integer;
  f :Text; { }
  Hour,Minute,Second,Sec100,rHour,rMinute,rSecond,rSec100 :Word;
  delta :LongInt;
begin{       }
  Assign(f,'BkTrack.in');
  Reset(f);{   }
  ReadLn(f,n);{  }
  Close(f);
  Assign(f,'BkTrack.out');
  Rewrite(f); {   }
  GetTime(Hour,Minute,Second,Sec100);
  BackTrack (f,a,n);
  GetTime(rHour,rMinute,rSecond,rSec100);
  delta:=rHour-Hour; delta:=delta*60+rMinute-Minute;
```

```
  delta:=delta*60+rSecond-Second; delta:=delta*100+rSec100-Sec100;
  WriteLn(f,'Время счета=',delta div 100,'.',delta mod 100,' сек');
  Close(f);
end.{      }
```

## § 5.3. Эффективное порождение перестановок

Последовательность $n!$ перестановок на множестве $\{1, 2, \ldots$ $\ldots, n\}$, в которой соседние перестановки различаются так мало, как только возможно, — лучшее, на что можно надеяться с точки зрения минимизации объема работы, необходимого для порождения перестановок. Для того чтобы такое различие было минимально возможным, любая перестановка в нашей последовательности должна отличаться от предшествующей ей транспозицией двух соседних элементов. Такую последовательность перестановок легко построить рекурсивно. Для $n = 1$ единственная перестановка $\{1\}$ удовлетворяет нашим требованиям. Предположим, мы имеем последовательность перестановок $\pi_1, \pi_2, \pi_3, \ldots$ на множестве $\{1, 2, \ldots, n\}$, в которой последовательные перестановки различаются только транспозицией смежных элементов. Расширим каждую из этих $(n-1)!$ перестановок, вставляя элемент $n$ на каждое из $n$ возможных мест. Порядок порождаемых таким образом перестановок будет следующим:

$$
1\left\{\begin{array}{l}
12\left\{\begin{array}{l}
123\left\{\begin{array}{llll} 1&2&3&4\\ 1&2&4&3\\ 1&4&2&3\\ 4&1&2&3 \end{array}\right.\\
132\left\{\begin{array}{llll} 4&1&3&2\\ 1&4&3&2\\ 1&3&4&2\\ 1&3&2&4 \end{array}\right.\\
312\left\{\begin{array}{llll} 3&1&2&4\\ 3&1&4&2\\ 3&4&1&2\\ 4&3&1&2 \end{array}\right.
\end{array}\right.\\
21\left\{\begin{array}{l}
321\left\{\begin{array}{llll} 4&3&2&1\\ 3&4&2&1\\ 3&2&4&1\\ 3&2&1&4 \end{array}\right.\\
231\left\{\begin{array}{llll} 2&3&1&4\\ 2&3&4&1\\ 2&4&3&1\\ 4&2&3&1 \end{array}\right.\\
213\left\{\begin{array}{llll} 4&2&1&3\\ 2&4&1&3\\ 2&1&4&3\\ 2&1&3&4 \end{array}\right.
\end{array}\right.
\end{array}\right.
$$

Данную последовательность перестановок можно порождать итеративно, получая каждую перестановку из предшествующей ей и небольшого количества добавочной информации. Это делается с помощью трех векторов: текущей перестановки $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, обратной к ней перестановки $\rho = (\rho_1, \rho_2, \ldots, \rho_n)$ и записи направления $d_i$, в котором сдвигается каждый элемент $i$ ($-1$, если он сдвигается влево; $+1$, если вправо; и 0, если не сдвигается). Элемент сдвигается до тех пор, пока не достигнет элемента, большего, чем он сам; в этом случае сдвиг прекращается. В этот момент направление сдвига данного элемента изменяется на противоположное и передвигается следующий меньший его элемент, который можно сдвинуть. Поскольку хранится перестановка, обратная к $\pi$, то в $\pi$ легко найти место следующего меньшего элемента. Алгоритм 5.4 представляет собой реализацию рассмотренного метода.

Корректность алгоритма доказывается индукцией по $n$. Алгоритм порождения всех $n!$ перестановок линеен, сложность его определяется как $n! + o(n!)$. Алгоритм 5.4 — один из наиболее эффективных алгоритмов для порождения перестановок.

Алгоритм 5.4.

for $i = 1$ to $n$ do begin $\pi_i = \rho_i = i$; $d_i = -1$; end;
$d_1 = 0$;
$\pi_0 = \pi_{n+1} = m = n + 1$; { }
while $m \neq 1$ do begin
        $(\pi_1, \pi_2, \ldots, \pi_n)$;
  $m = n$;
  while $\pi_{\rho_m + d_m} > m$ do begin
    $d_m = -d_m$;
    $m = m - 1$;
  end;
  $\pi_{\rho_m} \leftrightarrow \pi_{\rho_m + d_m}$;  $\pi$
  $\rho_{\pi_{\rho_m}} \leftrightarrow \rho_m$;  $\rho = \pi^{-1}, \pi_{\rho_m + d_m} = m$
end.

Рабочая программа на языке Pascal реализации эффективного метода генерации перестановок приводится в алгоритме 5.5.

В качестве примера в алгоритме 5.6 приводится программа реализации эффективного метода генерации перестановок, написанная на языке C.

Алгоритм 5.5. Pascal

```
Program Start_Effect; { }
uses CRT,DOS;
Const
  max_n=20; {              }
Type
  Vector=array[0..max_n+1] of Integer;
Var
  f :Text;

{                    }
Procedure Effect( Var z:Vector; n:Integer );
Const
  k :LongInt=0; {}
Var
  p,d :Vector;
  pm,dm,zpm :Integer;
  i,m,w :Integer;
begin;
  for i:=1 to n do begin z[i]:=i; p[i]:=i; d[i]:=-1; end;
  d[1]:=0;
  m:=n+1;
  z[0]:=m; z[n+1]:=m;
  while m<>1 do begin
    k:=k+1; Write(f,k,') '); {}
    for i:=1 to n do Write(f,z[i],' '); WriteLn(f);
    m:=n;
    while z[p[m]+d[m]]>m do begin d[m]:=-d[m]; m:=m-1; end;
    pm:=p[m]; dm:=pm+d[m]; w:=z[pm]; z[pm]:=z[dm]; z[dm]:=w;
    zpm:=z[pm]; w:=p[zpm]; p[zpm]:=pm; p[m]:=w;
  end;
end;{       }

Var {      }
  z :Vector;
  n :Integer; {}
  Hour,Minute,Second,Sec100 :Word;
  rHour,rMinute,rSecond,rSec100 :Word;
```

```
  delta :LongInt;
begin
  Assign(f,'Effect.in');
  Reset(f); {  }
  ReadLn(f,n); { }
  Close(f);
  Assign(f,'Effect.out');
  Rewrite(f); {  }
  GetTime(Hour,Minute,Second,Sec100);
  Effect(z,n);
  GetTime(rHour,rMinute,rSecond,rSec100);
  delta:=rHour-Hour;
  delta:=delta*60+rMinute-Minute;
  delta:=delta*60+rSecond-Second;
  delta:=delta*100+rSec100-Sec100;
  WriteLn(f,'Время счета=',delta div 100,'.',delta mod 100,' сек');
  Close(f);
end.
```

Алгоритм 5.6. C

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

void main(){      [1]  [2]      [n]
  int n;
  int *z,*p,*d;
  FILE *f;
  struct dostime_t t,tnew;
  long delta;
  unsigned long k;
  int pm,dm,zpm;
  int i,m,w;

  _dos_gettime(&t);
  f=fopen("primer.in","rt"); fscanf(f,"%d",&n);
```

```
  z=(int*)malloc((n+2)*sizeof(int));
  p=(int*)malloc((n+2)*sizeof(int));
  d=(int*)malloc((n+2)*sizeof(int));
  fclose(f);
  f=fopen("primer.out","wt");

  for( i=1; i<=n; i++ ){ z[i]=p[i]=i; d[i]=-1; }
  d[1]=0; z[0]=z[n+1]=m=n+1; k=0;
  while( m!=1 ){
    k++; fprintf(f,"\n%ld) ",k);
    for( i=1; i<=n; i++ )fprintf(f,"%d",z[i]);
    m=n;
    while( z[p[m]+d[m]]>m ) d[m]=-d[m]; m-;
    pm=p[m]; dm=pm+d[m]; w=z[pm]; z[pm]=z[dm]; z[dm]=w;
    zpm=z[pm]; w=p[zpm]; p[zpm]=pm; p[m]=w;
  }

  free(z);  free(p);  free(d);

  _dos_gettime(&tnew);
  delta=tnew.hour; delta-=t.hour; delta*=60;
  delta+=tnew.minute; delta-=t.minute; delta*=60;
  delta+=tnew.second; delta-=t.second; delta*=100;
  delta+=tnew.hsecond; delta-=t.hsecond;
  fprintf(f,"\nВремя счета %ld.%ld сек",(long)(delta/100),
  (long)(delta%100));
  fclose(f);
}
```

## § 5.4. Порождение подмножеств множества

Порождение подмножеств множества $\{a_1, a_2, \ldots, a_n\}$ — эквивалентно порождению $n$-разрядных двоичных наборов $a_i$, принадлежащих подмножеству, если и только если $i$-й разряд равен единице. Таким образом, задача порождения всех подмножеств множества сводится к задаче порождения всех возможных двоичных последовательностей длины $n$. Очевидно, что наиболее прямым способом порождения всех двоичных наборов длины $n$ является счет в системе счисления с основанием 2, как

показано в алгоритме 5.7. Перевод этого алгоритма на язык подмножеств множества $\{a_1, a_2, \ldots, a_n\}$ осуществляется согласно алгоритму 5.8, где добавлен фиктивный элемент $a_{n+1}$.

Алгоритм 5.7. $\binom{n}{2}$

```
for i = 0 to n do b_i = 0;
while b_n ≠ 1 do begin
        (b_{n-1}, b_{n-2}, ..., b_0);
  i = 0;
  while b_i = 1 do begin
    b_i = 0;
    i = i + 1;
  end;
  b_i = 1;
end.
```

Алгоритм 5.8.

```
S = ∅;
while a_{n+1} ∉ S do begin
        (S);
  i = 0;
  while a_i ∈ S do begin
    S = S − {a_i};
    i = i + 1;
  end;
  S = S ∪ {a_i};
end.
```

Задача. *Счастливая сумма*. Дано $n\,(n \geqslant 2)$ произвольных цифр: $a_1, a_2, \ldots, a_n$, где $a_i \in \{1, 2, \ldots, 9\}$, и произвольное целое число $m$. Написать программу, которая расставляла бы между каждой парой цифр: $a_1, a_2, \ldots, a_n$, записанными именно в таком порядке, знаки «+», «−» так, чтобы значением получившегося выражения было число $m$. Например, если $a_1, a_2, \ldots, a_n$ соответственно равны $1, 2, \ldots, 9$ и $m = 5$, то подойдет следующая расстановка знаков: $1 - 2 + 3 - 4 + 5 - 6 + 7 - 8 + 9$. Результаты расчетов сохранить в текстовом файле. Исходные дан-

ные представляются в текстовом файле со следующей структурой.

Первая строка — целое число $n$.
Вторая строка — целое число $m$.
Третья строка — цифры $a_1, a_2, \ldots, a_n$ через пробелы.

Пример файла исходных данных
14
71
6 5 8 8 4 7 5 2 3 4 5 7 8 9

Выходной файл для данного примера
$6+5+8+8+4+7+5+2+3+4-5+7+8+9=71$
$6+5+8+8+4+7+5-2-3+4+5+7+8+9=71$
$6+5+8+8+4+7-5+2+3+4+5+7+8+9=71$
$6-5+8+8+4+7+5+2+3+4+5+7+8+9=71$
Время счета = 0.60 с.

Ясно, что для решения данной задачи достаточно выполнить полный перебор всех возможных вариантов расстановки знаков между каждой парой цифр $a_1, a_2, \ldots, a_n$ и выбрать те расстановки знаков «$\pm$», которые удовлетворяют условию равенства суммы величине $m$. Всего позиций для расстановки «$\pm$» равно $n-1$, а значит, для полного перебора необходимо проверить $2^{n-1}$ двоичных наборов, если «+» будет соответствовать 1, а «−» будет соответствовать 0.

Программа решения задачи «счастливый билет» представлена алгоритмом 5.9. Процедура SubSet (подмножество) этой программы реализует рассмотренный выше алгоритм 5.7 счета в системе счисления с основанием 2 для порождения всех $(n-1)$-разрядных наборов. Порожденные двоичные наборы используются в процедуре Summa (сумма) для формирования суммы, соответствующей данному набору знаков «$\pm$», где «+» соответствует 1, а «−» соответствует 0.

Алгоритм 5.9. Pascal

```
Program Lucky_Ticket; { }
uses CRT,DOS;
Const
  max_n=20;
Type
```

```
  Vector=array[1..max_n] of LongInt;

Procedure Summa( var f:Text; var a,b:Vector; n,m:Integer );
{ }
Var
  S :LongInt;
  i :Integer;
begin
  S:=a[1];
  for i:=1 to n-1 do S:=S+(2*b[i]-1)*a[i+1];
  if S=m then begin
    Write(f,a[1]);
    for i:=1 to n-1 do begin
      if b[i]=1 then Write(f,'+') else Write(f,'-');
      Write(f,a[i+1]);
    end;
    WriteLn(f,' = ',S:4,' - число найдено!');
  end;
end;

Procedure SubSet( var f:Text; var a:Vector; n,m:Integer );
{     2}
Var
  b: Vector;
  i :Integer;
begin
  for i:=1 to n do b[i]:=0;
  while b[n]<>1 do begin
    Summa(f,a,b,n,m);
    i:=1;
    while b[i]=1 do begin
      b[i]:=0;
      i:=i+1;
    end;
    b[i]:=1;
  end;
end;

Var {      }
```

```
  a  :Vector;
  n,m,i :Integer;
  f  :Text; { }
  Hour,Minute,Second,Sec100 :Word;
  rHour,rMinute,rSecond,rSec100 :Word;
  delta :LongInt;
begin {      }
  Assign(f,'Number.in');
  Reset(f);{   }
  Read(f,n); { }
  Read(f,m); { }
  for i:=1 to n do Read(f,a[i]);
  Close(f);
  Assign(f,'Number.out');
  Rewrite(f); {   }
  GetTime(Hour,Minute,Second,Sec100);
  SubSet(f,a,n,m);
  GetTime(rHour,rMinute,rSecond,rSec100);
  delta:=rHour-Hour; delta:=delta*60+rMinute-Minute;
  delta:=delta*60+rSecond-Second;delta:=delta*100+rSec100-Sec100;
  WriteLn(f); WriteLn(f,'Время счета=',delta div 100,'.',delta
  mod 100,' c');
  Close(f);
end. {      }
```

## § 5.5. Генерация размещений с повторениями

Порождение множества всех размещений с повторениями длины $k$ из элементов $\{a_0, a_1, \ldots, a_{n-1}\}$ эквивалентно генерации множества $k$-разрядных чисел в системе счисления с основанием $n$: на $r$-м месте в размещении будет располагаться элемент $a_i$, если цифра в $r$-м разряде соответствующего числа равна $i$. Всего размещений с повторениями $n^k$. Например, для $k = 2$ и $n = 3$ все наборы длины два в системе счисления с основанием три можно записать: 00, 01, 02, 10, 11, 12, 20, 21, 22. Тогда эквивалентные размещения примут вид $(a_0a_0), (a_0a_1), (a_0a_2), (a_1a_0), (a_1a_1), (a_1a_2), (a_2a_0), (a_2a_1), (a_2a_2)$.

Алгоритм 5.10 использует фиктивный элемент $b_k$ при порождении наборов длины $k$ в системе счисления с основанием

$n$, где $b_i \in \{0, 1, \ldots, n-1\}$, $i = 0, 1, \ldots, k$, т. е. $b_i$ — это цифры генерируемого числа в системе счисления с основание $n$.

Алгоритм 5.10. $n$
$k$

```
for i = 0 to k do b_i = 0;
while b_k ≠ 1 do begin
        (b_{k-1}, b_{k-2}, ..., b_0);
    i = 0;
    while b_i = n - 1 do begin
        b_i = 0;
        i = i + 1;
    end;
    b_i = b_i + 1;
end.
```

## § 5.6. Порождение сочетаний

Обычно требуются не все подмножества множества $\{a_1, a_2, \ldots$ $\ldots, a_n\}$, а только те, которые, удовлетворяют некоторым ограничениям. Особый интерес представляют подмножества фиксированной длины $k$, $C_n^k$ сочетаний из $n$ предметов по $k$ штук. Как обычно, предполагаем, что основным множеством является множество натуральных чисел $\{1, 2, \ldots, n\}$; таким образом, будем порождать все сочетания длины $k$ из целых чисел $\{1, 2, \ldots, n\}$. Так, например, $C_6^3 = 20$ сочетаний из шести предметов по три (т. е. трехэлементные подмножества множества $\{1, 2, 3, 4, 5, 6\}$) записываются в лексикографическом порядке следующим образом:

| | | | |
|---|---|---|---|
| 123 | 135 | 234 | 256 |
| 124 | 136 | 235 | 345 |
| 125 | 145 | 236 | 346 |
| 126 | 146 | 245 | 356 |
| 134 | 156 | 246 | 456 |

Сочетания в лексикографическом порядке можно порождать последовательно простым способом. Начиная с сочетания $\{1, 2, \ldots, k\}$ следующее сочетание находится просмотром текущего сочетания справа налево для того, чтобы определить место самого правого элемента, который еще не достиг своего мак-

симального значения. Этот элемент увеличивается на единицу, и всем элементам справа от него присваиваются новые возможные наименьшие значения, как показано в алгоритме 5.11. Алгоритм порождения всех $C_n^k$ сочетаний линеен, его сложность $C_n^k + o(C_n^k)$.

Алгоритм 5.11.

$c_0 = -1$;
**for** $i = 1$ **to** $k$ **do** $c_i = i$;
$j = 1$;
**while** $j \neq 0$ **do begin**
    $(c_1, c_2, \ldots, c_k)$;
    $j = k$;
    **while** $c_j = n - k + j$ **do** $j = j - 1$;
    $c_j = c_j + 1$;
    **for** $i = j + 1$ **to** $k$ **do** $c_i = c_{i-1} + 1$;
**end.**

Задача. *Выпуклый многоугольник.* Дано множество пар целых чисел $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ — координаты точек на плоскости. Написать программу выделения тех точек из заданного множества, которые являются вершинами выпуклого многоугольника, содержащего все остальные точки. Исходные данные представлены в текстовом файле, имеющем следующую структуру. Первым числом в файле является целое $n$ — количество точек. Последующие числа определяют $n$ пар целых $(x_i, y_i)$ — координаты точек. Результаты расчетов, признаки принадлежности исходных точек выпуклому многоугольнику: 0 — точка не принадлежит, 1 — точка принадлежит, сохранить в текстовом файле.

Пример файла исходных данных

15
0 0 9 9 5 1 2 3 2 7 1 1 5 4 6 7 1 5 6 7 3 4 5 7 7 8 8 7 6 4

Выходной файл для данного примера

1 1 1 0 1 0 0 0 1 0 0 0 0 1 0

Решение. Если не применять специальных методов, то в качестве решения можно использовать следующий алгоритм. Ясно, что точка $(x_i, y_i)$ является вершиной выпуклого многоугольника, если она не лежит ни в одном треугольнике, вер-

шинами которых являются исходные точки $(x_1, y_1), (x_2, y_2), \ldots$ $\ldots, (x_n, y_n)$ без рассматриваемой точки $(x_i, y_i)$. Всего треугольников из $n$ точек можно составить $C_n^3 + o(C_n^3)$. Тогда сложность задачи полного перебора треугольников для каждой точки $(x_i, y_i)$ составит $O(n^4)$. Реализация данного подхода представлена программой на языке Pascal в алгоритме 5.12.

Алгоритм 5.12.

```
Program Start_Evelope; { }
uses CRT,DOS;
Const
  n_max=100;
Type
  Vector=array[1..n_max] of Integer;
  Combine=array[0..3] of Integer;


Var f  :Text; { }
  x,y :Vector; { }
  z  :Vector; {  }


Procedure Envelope( var c:Combine; n:Integer );
{ }
Const
  k :Integer=3;
Var
  i,a :Integer;
  k1,k2,k3 :Integer;
  sign1,sign2,sign3 :Integer;
begin
  k1:=c[1]; k2:=c[2]; k3:=c[3];
  for i:=1 to n do begin
    if (i=k1) or (i=k2) or (i=k3) then continue;
    a:=(x[k1]-x[i])*(y[k2]-y[i])-(x[k2]-x[i])*(y[k1]-y[i]);
    if a=0 then continue; sign1:=a div abs(a);
    a:=(x[k2]-x[i])*(y[k3]-y[i])-(x[k3]-x[i])*(y[k2]-y[i]);
    if a=0 then continue; sign2:=a div abs(a);
    a:=(x[k3]-x[i])*(y[k1]-y[i])-(x[k1]-x[i])*(y[k3]-y[i]);
    if a=0 then continue; sign3:=a div abs(a);
```

```
    if (sign1=sign2) and (sign2=sign3) then z[i]:=0;
  end;
end;

Procedure Print( var c:Combine; n:Integer );
Var
  i :Integer;
begin
  WriteLn(f);
  for i:=1 to n do Write(f,c[i],' ');
end;

Procedure Combination( n:Integer );
{      3}
Const
  k :Integer=3; { }
Var c  :Combine; {}
  i,j :Integer;
begin
  c[0]:=-1;
  for i:=1 to k do c[i]:=i;
  j:=1;
  while j<>0 do begin
    {              }
    Envelope(c,n);
    j:=k;
    while c[j]=n-k+j do j:=j-1;
    c[j]:=c[j]+1;
    for i:=j+1 to k do c[i]:=c[i-1]+1;
  end;
end;

Var {      }
  i,n :Integer; {  }
begin {      }
  Assign(f,'Envelope.in');
  Reset(f); {  }
  Read(f,n); { }
```

```
  for i:=1 to n do begin Read(f,x[i],y[i]); z[i]:=1;end;
  Close(f);
  Assign(f,'Envelope.out');
  Rewrite(f); {  }
  Combination(n);
  WriteLn(f); {    }
  for i:=1 to n do Write(f,z[i],' ');
  Close(f);
end. {      }
```

## § 5.7. Порождение композиций и разбиений

Рассмотрим задачу порождения разбиений положительного числа $n$ в последовательность неотрицательных целых чисел $\{z_1, z_2 \ldots, z_k\}$, так что $z_1 + z_2 + \ldots + z_k = n$.

Разбиение $\{z_1, z_2 \ldots, z_k\}$ называется *композицией* числа $n$, если учитывается порядок чисел $z_i$. Как правило, интересуют композиции, в которых либо все $z_i \geqslant 0$, либо все $z_i > 0$.

Разбиение $\{z_1, z_2 \ldots, z_k\}$ называется *разбиением* числа $n$, если все $z_i > 0$ и порядок чисел $z_i$ не важен. По сути разбиение $\{z_1, z_2 \ldots, z_k\}$ числа $n$ является мультимножеством (см. п. 2.8).

Рассмотрим примеры разбиения числа $n = 3$:

$(1,2), (2,1)$ — все композиции числа три из двух частей, $z_i > 0$;

$(1,1,1)$ — все композиции числа три из трех частей, $z_i > 0$;

$(0,3), (1,2), (2,1), (3,0)$ — все композиции числа три из двух частей $z_i \geqslant 0$;

$(3), (1,2), (1,1,1)$ — все разбиения числа три.

### Композиции $z_i \geqslant 0$

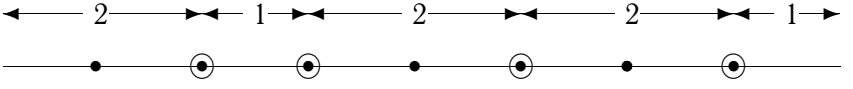
Композицию $\{z_1, z_2, \ldots, z_k\}$, где $z_i \geqslant 0$, числа $z_1 + z_2 + \ldots + z_k = n$ можно интерпретировать следующим образом. Каждое значение $z_i = 1_i + 1_i + \ldots + 1_i$ представим как сумму единиц, количество которых $z_i$. Индекс у элемента $1_i$ показывает его принадлежность разложению числа $z_i$. Таким образом, мы ввели $k$ типов различных элементов $\{1_1, 1_2, \ldots, 1_k\}$, значение каждого из них равно единице. Теперь любую композицию можно представить как сумму, составленную из $n$ произвольных единиц множества $\{1_1, 1_2, \ldots, 1_k\}$. Суммируя подобные единицы $1_i$ с одинаковыми индексами, получим соответствующие значения $z_i$ композиции. Данное соответствие является взаимно одно-

значным, откуда и следует, что число композиций равно числу сочетаний с повторениями: $\overline{C}_n^k = C_{n+k-1}^{k-1} = C_{n+k-1}^n$. Каждое из сочетаний $C_{n+k-1}^{k-1}$ можно интерпретировать как расстановку 1 и 0 длины $n+k-1$, в которой $n$ единиц и $k-1$ нуль, т. е. каждому сочетанию ставим в соответствие $(n+k-1)$-разрядное число из единиц и нулей; верно и обратно. Суммируя в таком числе слева направо единицы между нулями (всего $k-1$), будем получать соответствующие значения членов композиции $z_i$ (всего $k$). Например, одному из $C_{n+k-1}^{k-1} = C_{17+7-1}^{7-1}$ сочетаний 11011100111101111111010 соответствует композиция $(2,3,0,4,7,1,0)$ числа 17.

Ясно, что методы предыдущего раздела генерации подмножеств множества легко применить к последовательному порождению рассмотренных композиций $z_i \geqslant 0$.

### Композиции $z_i > 0$

Удобное представление композиций получается из рассмотрения целого числа $n$ как отрезка прямой, состоящего из отрезков единичной длины. Линия разделена $n-1$ точками, и композиция получается пометкой некоторых из них. Элементами композиции являются просто расстояния между смежными точками.




На рисунке показано графическое представление композиции $(2,1,2,2,1)$ для числа $n=8$. Очевидно, что каждая композиция числа $n$ соответствует способу выбора подмножества из $n-1$ точек. Каждой точке можно сопоставить двоичную цифру, и, таким образом, композиции $n$ будет соответствовать $(n-1)$-разрядное число. В этой интерпретации композиция из $k$ частей соответствует $(n-1)$-разрядному числу ровно с $k-1$ единицами, и поэтому существует $C_{n-1}^{k-1}$ таких композиций.

Воспользуемся методами предыдущего раздела для генерации композиций $z_i > 0$. Учитывая, что в композиции $z_1 + z_2 + \ldots + z_k = n$ каждый из $z_i > 0$, тогда для новых переменных $r_i = z_i - 1$ $(r_i \geqslant 0)$ получаем соответствующую композицию $\{r_1, r_2, \ldots, r_k\}$ для числа $n - k = r_1 + r_2 + \ldots + r_k$, $r_i \geqslant 0$. Генерация слагаемых $r_i \geqslant 0$ композиции $\{r_1, r_2, \ldots, r_k\}$ подробно разобрана

в предыдущем разделе добавляя к каждому из $r_i$ по единице, получим слагаемые $z_i > 0$ композиции $\{z_1, z_2, \ldots, z_k\}$.

### Разбиения

Разбиения $n$ отличаются от композиций $n$ тем, что порядок компонент не важен, так что, например, не делается различия между, скажем, $1+1+2$, $1+2+1$, $2+1+1$. Таким образом, разбиение $n$ можно рассматривать как мультимножество, которое записывается следующим способом:

$$\{m_1 \bullet z_1, m_2 \bullet z_2, \ldots, m_k \bullet z_k\},$$

где имеется $m_1$ вхождений $z_1$, $m_2$ вхождений $z_2$, $m_3$ вхождений $z_3$ и т. д. и $n = \sum_{i=1}^{k} m_i z_i$. Каждое разбиение удовлетворяет $z_1 > z_2 > \ldots > z_k$. Рассмотрим пример генерации разбиений для $n = 7$. Последовательность генерации разбиений данного примера далее будет положена в основу алгоритма порождения полного списка разбиений.

Идея приведенного списка разбиений состоит в том, чтобы переходить от одного разбиения к следующему, рассматривая самый правый элемент $m_k \bullet z_k$ разбиения. Если $m_k z_k$ достаточно велико ($m_k > 1$), можно исключить два $z_k$ для того, чтобы добавить еще одно $z_k + 1$ (или включить одно $z_k + 1$, если в текущий момент его нет). Если $m_k = 1$, то $m_{k-1} z_{k-1} + m_k z_k$ достаточно велико для того, чтобы добавить $z_{k-1} + 1$. Все, что остается, превращается в соответствующее число единиц и формируется новое разбиение.

$$\begin{array}{ll}
(7 \bullet 1) & = (1,1,1,1,1,1,1) \\
(1 \bullet 2,\ 5 \bullet 1) & = (2,1,1,1,1,1) \\
(2 \bullet 2,\ 3 \bullet 1) & = (2,2,1,1,1) \\
(3 \bullet 2,\ 1 \bullet 1) & = (2,2,2,1) \\
(1 \bullet 3,\ 4 \bullet 1) & = (3,1,1,1,1) \\
(1 \bullet 3,\ 1 \bullet 2, 2 \bullet 1) & = (3,2,1,1) \\
(1 \bullet 3,\ 2 \bullet 2) & = (3,2,2) \\
(2 \bullet 3,\ 1 \bullet 1) & = (3,3,1) \\
(1 \bullet 4,\ 3 \bullet 1) & = (4,1,1,1) \\
(1 \bullet 4,\ 1 \bullet 2, 1 \bullet 1) & = (4,2,1) \\
(1 \bullet 4,\ 1 \bullet 3) & = (4,3) \\
(1 \bullet 5,\ 2 \bullet 1) & = (5,1,1) \\
(1 \bullet 5,\ 1 \bullet 2) & = (5,2) \\
(1 \bullet 6,\ 1 \bullet 1) & = (6,1) \\
(1 \bullet 7) & = (7)
\end{array}$$

Алгоритм 5.13 использует рассмотренный процесс для порождения всех разбиений числа $n$.

Алгоритм 5.13. $n$

$k = 1;$
$z_{-1} = 0;$
$m_{-1} = 0;$
$z_0 = n + 1;$
$m_0 = 0;$
$z_1 = 1;$
$m_1 = n;$
**while** $k \neq 0$ **do begin**
$(m_1 \cdot z_1, m_2 \cdot z_2, \ldots, m_k \cdot z_k);$
$Summa = m_k z_k;$
**if** $m_k = 1$ **then begin**
$k = k - 1;$
$Summa = Summa + m_k z_k;$
**end;**
**if** $z_{k-1} = z_k + 1$ **then begin**
$k = k - 1;$
$m_k = m_k + 1;$
**end**
**else begin**
$z_k = z_k + 1;$
$m_k = 1;$
**end;**
**if** $Summa > z_k$ **then begin**
$z_{k+1} = 1;$
$m_{k+1} = Summa - z_k;$
$k = k + 1;$
**end;**
**end.**

Алгоритм 5.13 линеен, так как число операций, необходимых для перехода от одного разбиения к другому, ограничено константой, не зависящей от $n$ и $k$.

Программа на языке **Pascal** реализации рассмотренного метода генерации разбиений чисел приводится в алгоритме 5.14. Отметим, что в программе число для разбиения $n$ читается из

файла, а порождаемые разбиения сохраняются в файле. Попутно программа вычисляет полное время генерации всех разбиений с точностью до сотых долей секунд.

Алгоритм 5.14. Pascal
$n$

```
Program Start_Divide; {   }
uses CRT,DOS;
Const
  max_n=100; {              }
Type
  Vector=array[-1..max_n] of Integer;
Var
  f  :Text;
  z,m :Vector;

Procedure Print( var m,z:Vector; k:Integer );{ }
Var
  i :Integer;
begin
  Write(f,'{');
  for i:=1 to k do begin
    Write(f,m[i],'*',z[i]);
    if i<>k then Write(f,',');
  end;
  WriteLn(f,'}');
end;

Procedure Divide( n:Integer );{   }
Var
  k,Sum :Integer;
begin;
  k:=1;
  z[-1]:=0;
  m[-1]:=0;
  z[0]:=n+1;
  m[0]:=0;
  z[1]:=1;
```

```
  m[1]:=n;
  while k<>0 do begin
    Print(m,z,k);
    Sum:=m[k]*z[k];
    if m[k]=1 then begin
      k:=k-1;
      Sum:=Sum+m[k]*z[k];
    end;
    if z[k-1]=z[k]+1 then begin k:=k-1; m[k]:=m[k]+1; end
    else begin z[k]:=z[k]+1; m[k]:=1; end;
    if Sum>z[k] then begin
      z[k+1]:=1;
      m[k+1]:=Sum-z[k];
      k:=k+1
    end;
  end;
end;

Var {       }
  n :Integer; { }
  Hour,Minute,Second,Sec100 :Word;
  rHour,rMinute,rSecond,rSec100 :Word;
  delta :LongInt;
begin {       }
  Assign(f,'Divide.in');
  Reset(f); {   }
  ReadLn(f,n); {    }
  Close(f);
  Assign(f,'Divide.out');
  Rewrite(f); {   }
  GetTime(Hour,Minute,Second,Sec100);
  Divide(n);
  GetTime(rHour,rMinute,rSecond,rSec100);
  delta:=rHour-Hour;
  delta:=delta*60+rMinute-Minute;
  delta:=delta*60+rSecond-Second;
  delta:=delta*100+rSec100-Sec100;
```

```
  WriteLn(f,'Время счета=',delta div 100,'.',delta mod 100,' сек');
  Close(f);
end. {      }
```

## § 5.8. Генерация случайных перестановок

Пусть $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — произвольно выбранная перестановка целых чисел $1, 2, \ldots, n$, например, $\pi = (1, 2, \ldots, n)$ — тождественная. Случайную перестановку можно получить за линейное время $O(n)$ из выбранной перестановки $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, выполнив в ней $n$ транспозиций (см. п. 8.6).

Для промежуточных перестановок введем верхний индекс, значение которого будет соответствовать количеству выполненных транспозиций. Один из элементов в каждой транспозиции выбирается случайным образом. Индекс такого элемента устанавливается функцией $\mathrm{rand}(k, l)$, которая порождает независимые случайные целые числа на отрезке $[k, l]$ с равномерным распределением.

Положим $\pi^{(0)} = \left(\pi_1^{(0)}, \pi_2^{(0)}, \ldots, \pi_n^{(0)}\right)$ равной исходной перестановке $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Каждая следующая перестановка $\pi^{(k)} = \left(\pi_1^{(k)}, \pi_2^{(k)}, \ldots, \pi_n^{(k)}\right)$ получается из предыдущей перестановки $\pi^{(k-1)} = \left(\pi_1^{(k-1)}, \pi_2^{(k-1)}, \ldots, \pi_n^{(k-1)}\right)$ транспозицией элементов $\pi_k^{(k-1)}$ и $\pi_{\mathrm{rand}(k,n)}^{(k-1)}$, где $k = 1, 2, \ldots, n$. Между элементами перестановок $\pi^{(0)}, \pi^{(1)}, \ldots, \pi^{(n)}$ выполняются равенства $\pi_k^{(n)} = \pi_k^{(n-1)} = \ldots = \pi_k^{(k)}$, где $k = 1, 2, \ldots, n$.

Покажем, что после выполнения всех указанных $n$ транспозиций равновероятно получение любой из $n!$ возможных перестановок $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ исходных чисел $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Для этого достаточно проверить, что $\Pr(\pi^{(n)} = \sigma) = 1/n!$. С этой целью введем события $A_1, A_2, \ldots, A_n$:

$$A_1 = \left\{\pi_1^{(1)} = \sigma_1\right\} = \left\{\pi_{(\mathrm{rand}(1,n))}^{(1)} = \sigma_1\right\},$$

$$A_k = \left\{\pi_k^{(k)} = \sigma_k \middle| \pi_1^{(k-1)} = \sigma_1 \,\&\, \pi_2^{(k-1)} = \right.$$

$$\left. = \sigma_2 \,\&\, \ldots \,\&\, \pi_{k-1}^{(k-1)} = \sigma_{k-1}\right\} =$$

$$= \left\{ \pi^{(k-1)}_{\mathrm{rand}(k,n)} = \sigma_k \middle| \pi^{(k-1)}_1 = \sigma_1 \,\&\, \pi^{(k-1)}_2 = \right.$$

$$\left. = \sigma_2 \,\&\, \ldots \,\&\, \pi^{(k-1)}_{k-1} = \sigma_{k-1} \right\} =$$

$$= \left\{ \pi^{(k)}_k = \sigma_k \middle| \pi^{(1)}_1 = \sigma_1 \,\&\, \pi^{(2)}_2 = \sigma_2 \,\&\, \ldots \,\&\, \pi^{(k-1)}_{k-1} = \sigma_{k-1} \right\},$$

$$k = 2, \ldots, n.$$

Вероятности данных событий, согласно схеме формирования перестановок $\pi^{(0)}, \pi^{(1)}, \ldots, \pi^{(k)}$, равны

$$\Pr(A_k) = 1/(n-k+1), \text{ где } k = 1, 2, \ldots, n.$$

Условие $\pi^{(k-1)}_1 = \sigma_1 \,\&\, \pi^{(k-1)}_2 = \sigma_2 \,\&\, \ldots \,\&\, \pi^{(k-1)}_{k-1} = \sigma_{k-1}$ в событии $A_k$, $k = 1, 2, \ldots, n$, обеспечивает выбор элемента $\pi^{(k-1)}_{\mathrm{rand}(k,n)}$ из множества $\left\{ \pi^{(k-1)}_k, \pi^{(k-1)}_{k+1}, \ldots, \pi^{(k-1)}_n \right\}$, которое совпадает с множеством $\{\sigma_k, \sigma_{k+1}, \ldots, \sigma_n\}$. Индекс же $\mathrm{rand}(k,n)$ элемента $\pi^{(k-1)}_{\mathrm{rand}(k,n)}$ является независимой случайной величиной с равномерным распределением на отрезке $[k,n]$ целых чисел.

Теперь заметим, что

$$\Pr\left(\pi^{(n)} = \sigma\right) = \Pr\left(\pi^{(n)}_1 = \sigma_1 \,\&\, \pi^{(n)}_2 = \sigma_2 \,\&\, \ldots \,\&\, \pi^{(n)}_n = \sigma_n\right) =$$

$$= \Pr\left(\pi^{(1)}_1 = \sigma_1 \,\&\, \pi^{(2)}_2 = \sigma_2 \,\&\, \ldots \,\&\, \pi^{(n)}_n = \sigma_n\right) =$$

$$= \Pr(A_n)\Pr(A_{n-1}) \cdot \ldots \cdot \Pr(A_1) = \frac{1}{1} \times \frac{1}{2} \times \ldots \times \frac{1}{n} = \frac{1}{n!}.$$

Рассмотренный метод генерации случайной перестановки представлен в алгоритме 5.15.

Алгоритм 5.15.

$$\pi = (\pi_1, \pi_2, \quad, \pi_n)$$

```
for k = 1 to n do π_k = k; { }
for k = 1 to n − 1 do begin { }
    { π_k ↔ π_rand(k,n)}
    m = rand(k, n);
    w = π[k];
    π[k] = π[m];
    π[m] = w;
end.
```

Если генерацию перестановки вести с конца, то алгоритм примет следующий вид.

```
for k = 1 to n do π_k = k; { }
for k = n downto 2 do begin { }
    { π_k ↔ π_rand(1,k)}
    m = rand(1, k);
    w = π[k];
    π[k] = π[m];
    π[m] = w;
end.
```

Глава 6

# СОРТИРОВКА И ПОИСК

Рассматриваемые здесь вопросы можно отнести к наиболее часто встречающимся в задачах машинной обработки данных. Почти во всех компьютерных приложениях множество объектов должно быть переразмещено в соответствии с некоторым заранее определенным порядком. Очевидно, что с сортированными данными легче работать, чем с произвольно расположенными. Сортировка больших объемов данных составляет значительную часть коммерческой обработки данных, эффективные алгоритмы для сортировки важны и с экономической точки зрения. Эффективность оценивается с точки зрения памяти и времени, а также простоты программирования. Простота программирования предполагает и простоту понимания используемого метода, поскольку для того, чтобы написать хорошую программу, важно хорошо понять соответствующий метод. В наших же оценках мы будем учитывать только число сравнений. Самые простые алгоритмы сортировки, основанные на сравнении элементов, имеют сложность порядка $O(n^2)$, а лучшие из них обходятся количеством сравнений $O(n\log_2(n))$.

Задачу сортировки можно сформулировать так: дана последовательность из $n$ элементов $a_1, a_2, \ldots, a_n$, выбранных из множества, на котором задан линейный порядок, т. е. для любых $a_i, a_j$ выполняется либо $a_i < a_j$, либо $a_i \leqslant a_j$, либо, $a_i = a_j$, либо $a_i > a_j$, либо $a_i \geqslant a_j$. Требуется найти перестановку $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ этих $n$ элементов, которая отобразит данную последовательность в неубывающую последовательность $a_{\pi_1} \leqslant a_{\pi_2} \leqslant \ldots \leqslant a_{\pi_n}$. Как правило, далее будем получать саму упорядоченную последовательность, а не упорядочивающую перестановку $\pi$.

Методы сортировки классифицируются на внутренние (когда данные размещаются в оперативной памяти) и внешние (когда данные размещаются на внешней памяти). Внешняя сортировка составляет часть таких приложений, в которых в работу вовлекается гораздо больше элементов, чем можно сразу запомнить в оперативной памяти. Поэтому методы внешней сортировки применимы к данным, находящимся на внешних устройствах памяти, и имеют огромное коммерческое значение.

Внутренняя сортировка важна как для разработки алгоритмов, так и для коммерческих приложений. Сортируемые данные размещаются в оперативной памяти. Здесь рассматриваются именно методы внутренней сортировки данных. Известно много алгоритмов сортировки дан-

ных. Почему же так много методов сортировки? Ответ состоит в том, что каждый метод имеет свои преимущества и недостатки, поэтому он оказывается эффективнее других при некоторых структурах данных и аппаратной части.

Здесь же мы не пытаемся охватить даже те из них, которые считаются важными; скорее, мы ограничимся методами, оказавшимися полезными в разработке алгоритмов и в практической их реализации. Рассматриваемые здесь методы сортировки активно используются при разработке алгоритмов во многих разделах данного пособия. Полезно изучить характеристики каждого метода сортировки, чтобы можно было производить разумный выбор для конкретных приложений. К счастью, задача изучения этих алгоритмов не столь уж громоздка.

## § 6.1. Сортировка вставками

Сортировка вставками элементов $a_1, a_2, \ldots, a_n$ относится к наиболее очевидным методам (алгоритм 6.1). Для компактности алгоритма вводится фиктивный элемент $a_0$, значение которого устанавливается равным $-\infty$. Сортировка проходит цикл для $j = 2, 3, \ldots, n$; для каждого $j$ элемент $a_j$ вставляется в свое правильное место среди $a_1, a_2, \ldots, a_{j-1}$. При вставке элемент $a_j$ временно размещается в $w$ и просматриваются имена $a_{j-1}, a_{j-2}, \ldots, a_1$; они сравниваются с $w$ и сдвигаются вправо, если обнаруживается, что они больше $w$. Имеется фиктивный элемент $a_0$, значение которого $-\infty$ служит для остановки просмотра слева.

Алгоритм 6.1.

```
a_0 = -∞;
for j = 2 to n do begin
    i = j - 1;
    w = a_j;
    while w < a_i do begin
        a_{i+1} = a_i;
        i = i - 1;
    end;
    a_{i+1} = w;
end.
```

Сложность алгоритма определяется числом проверок в цикле условия $w < a_i$. Не уменьшая общности, полагаем, что $a_1, a_2, \ldots, a_n$ — перестановка элементов множества $\{1, 2, \ldots, n\}$.

Сравнение $w < a_i$ для конкретного $w = a_j = k$ $(j \geqslant 2)$ выполняется $1 + d_k$ раз, где $d_k$ — число элементов, больших $k$ и стоящих слева от него, т. е. $d_k$ — это число инверсий, у которых второй элемент равен $k$. Числа $d_k$ составляют таблицу инверсий, а так как $0 \leqslant d_1 \leqslant n-1, 0 \leqslant d_2 \leqslant n-2, \ldots, 0 \leqslant d_{n-1} \leqslant 1, d_n = 0$, то в худшем случае сортировка элементов $a_1, a_2, \ldots, a_n$ потребует числа сравнений не больше, чем $\sum\limits_{k=1}^{n}(1+d_k) \leqslant \sum\limits_{k=1}^{n}(1+n-k) = \frac{n(n+1)}{2} = O(n^2)$. Сложность сортировки является квадратичной.

## § 6.2. Пузырьковая сортировка

Рассматриваемый метод пузырьковой сортировки последовательности $a_1, a_2, \ldots, a_n$ представляет собой наиболее очевидный метод систематического обмена местами слева направо смежных элементов, не отвечающих выбранному порядку, до тех пор пока не оказывается на правильном месте. Эта техника получила название пузырьковой сортировки, так как большие элементы *пузырьками всплывают* вверх в конец списка. Реализация метода представлена алгоритмом 6.2. В алгоритме используется переменная $b$, значение которой при каждом проходе цикла устанавливается равным наибольшему индексу $t$, такому, что все элементы $a_{t+1}, a_{t+2}, \ldots, a_n$ уже находятся на своих окончательных позициях. Ясно, что не имеет смысла продолжать просмотр для указанных элементов.

Алгоритм 6.2.

```
b = n;
while b ≠ 0 do begin
    t = 0;
    for j = 1 to b − 1 do begin
        if a_j > a_{j+1} then begin
            a_j ↔ a_{j+1};
            t = j;
        end;
    end;
    b = t;
end.
```

Сложность алгоритма определяется числом проверок условия $a_j > a_{j+1}$ в цикле и числом обменов $a_j \leftrightarrow a_{j+1}$, которое равно числу инверсий в исходной перестановке элементов $a_1, a_2, \ldots, a_n$. Определим число сравнений. В худшем случае верхняя граница $b-1$ вложенного цикла $for$ на каждом шаге внешнего цикла $while$ будет уменьшаться на 1, тогда число сравнений равно

$$\sum_{b=n}^{1}(b-1) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2} = O(n^2).$$

В алгоритме 6.3 представлена *полная* пузырьковая сортировка. Это наиболее популярный и упрощенный вариант алгоритма 6.2. Ясно, что основным достоинством алгоритма полной пузырьковой сортировки является легкость программирования. Сложность же алгоритма 6.3 остается постоянной, не зависящей от расположения исходных данных, и равной

$$\sum_{i=1}^{n}(n-i) = (n-1) + (n-2) + \ldots + 1 = \frac{n(n-1)}{2} = O(n^2).$$

Алгоритм 6.3.

```
for i = 1 to n do begin
    for j = 1 to n − i do begin
        if a_j > a_{j+1} then begin
            a_j ↔ a_{j+1}; { }
        end;
    end;
end.
```

## § 6.3. Сортировка перечислением

Идея сортировки последовательности $a_1, a_2, \ldots, a_n$ данных перечислением состоит в том, чтобы сравнить попарно все элементы $a_1, a_2, \ldots, a_n$ и подсчитать, сколько из них меньше каждого отдельного элемента (алгоритм 6.4). Для подсчета числа элементов, меньших данного, в алгоритме используется вспомогательный вектор счетчиков $c_1, c_2, \ldots, c_n$. После завершения алгоритма значения $c_j + 1, j = 1, 2, \ldots, n$, определяют окончательное положение элементов $a_j$ в сортированной последовательности $r_1, r_2, \ldots, r_n$.

Алгоритм 6.4.

```
for i = 1 to n do c_i = 0; { }
for i = n downto 2 do begin
    for j = i − 1 downto 1 do begin
        if a_i > a_j then c_i = c_i + 1
        else c_j = c_j + 1;
    end;
end;
for i = 1 to n do begin
    r_{c_i+1} = a_i; {r_i     }
end.
```

Сложность алгоритма сортировки перечислением определяется парой вложенных циклов и составляет $O(n^2)$. Величина сложности не зависит от расположения данных в исходной последовательности $a_1, a_2, \ldots, a_n$.

Пусть перестановка $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, где $\pi_i = c_i + 1$, $i = 1, 2, \ldots, n$. Алгоритм 6.4 сортировки перечислением определяет перестановку $\pi^{-1}$, которая соответствует расположению исходных данных $a_{\pi_1^{-1}} \leqslant a_{\pi_2^{-1}} \leqslant \ldots \leqslant a_{\pi_n^{-1}}$ (см. п. 2.14).

## § 6.4. Сортировка всплытием Флойда

Все ранее упомянутые методы сортировки последовательности $a_1, a_2, \ldots, a_n$ требовали сравнений порядка $O(n^2)$, и «это никуда не годится». Рассмотрим один из наиболее элегантных и эффективных методов сортировки сложности $O(n \log_2(n))$, предложенный Флойдом. До сих пор он остается самым лучшим из существующих методов. В алгоритме активно используется упорядоченное двоичное дерево, пример которого представлен на рис. 6.1. Значение в каждой его вершине не меньше, чем значение в его дочерних вершинах. Двоичное дерево называется *частично упорядоченным*, если свойство упорядоченности выполняется для каждой из его вершин, однако для корня это свойство может нарушаться. Пример частично упорядоченного дерева приведен на рис. 6.2.

Интересно отметить, что в ранее рассмотренных методах сортировки сложности $O(n^2)$ при выборе наибольшего (наименьшего) элемента, *забывали* информацию о других, забракованных элементах на эту роль, хотя эта проверка и выполнялась.

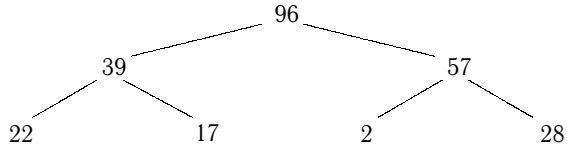



Рис. 6.1. Пример упорядоченного двоичного дерева

Структура же дерева позволяет сохранить состояние процесса сортировки последовательности $a_1, a_2, \ldots, a_n$ на каждом его шаге, с целью использования этого состояния в дальнейших расчетах и уменьшения числа операций сравнений при поиске наибольшего (наименьшего) из оставшихся элементов.

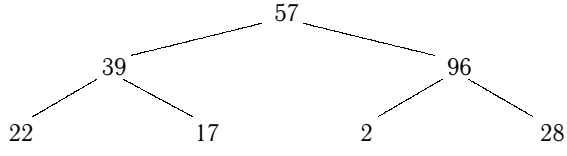



Рис. 6.2. Пример частично упорядоченного двоичного дерева

Метод сортировки Флойда представлен в алгоритме 6.5, где исходная последовательность $a_1, a_2, \ldots, a_n$ данных представляется в виде дерева на смежной памяти (одномерный массив $a[1..n]$). В таком дереве ребра присутствуют неявно и вычисляются с помощью арифметических операций над индексами элементов массива — смежной памяти. Пример двоичного дерева показан на рис. 6.3. Корень дерева — $a[1]$, за каждой вершиной $a[k]$ следуют вершины $a[2k]$ и $a[2k+1]$. Использование смежной памяти для представления дерева имеет и другие преимущества, которые становятся очевидными после анализа алгоритма 6.5.

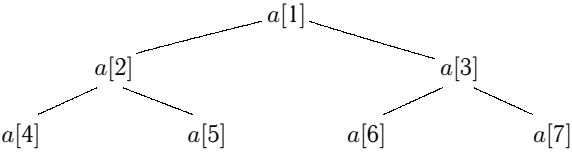



Рис. 6.3. Пример двоичного дерева на смежной памяти

Основу алгоритма 6.5 составляет процедура *Surface*$(a[i..k])$ всплытия Флойда, которая за $O(\log_2(n))$ сравнений преобразует почти упорядоченное поддерево в упорядоченное. Поддерево представляется на одномерном массиве $a[i..k]$, рис. 6.4, где $a[i]$ —

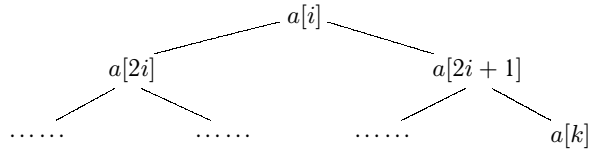



Рис. 6.4. Двоичное поддерево на смежной памяти $a[i..k]$

корень поддерева, $a[k]$ — максимальный элемент массива, который еще может принадлежать поддереву.

Определим сложность процедуры *Surface* всплытия Флойда. Процедура заключается в том, что значение из корня (здесь может нарушаться условие упорядоченности) всплывает по направлению к листьям (последний уровень вершин в дереве) до тех пор, пока дерево не преобразуется в упорядоченное. Во время всплытия на каждом уровне выполняется конечное число $C$ операций сравнения элементов. Если положить, что высота дерева (число уровней в дереве) равна $h$, то сложность одного всплытия составит $C \cdot h = O(h)$. Высота $h$ регулярного двоичного дерева из $n$ вершин легко находится из соотношения $n \leqslant 2^0 + 2^1 + \ldots + 2^{h-1}$, где $2^{i-1}$ — количество вершин на $i$-м уровне дерева, $i = 1, \ldots, h$. Отсюда высота дерева $h = \lceil \log_2(n+1) \rceil$. Таким образом, сложность процедуры *Surface* всплытия Флойда составляет $O(\log_2(n))$.

Рассмотренная процедура *Surface* всплытия Флойда позволяет в почти упорядоченном дереве найти наибольший (наименьший) элемент за число сравнений $O(\log_2(n))$, преобразуя дерево к упорядоченному виду. В результате найденный элемент будет располагаться в вершине дерева. Для сортировки же множества элементов $a_1, a_2, \ldots, a_n$ из них по алгоритму 6.5 сначала организуется почти упорядоченное двоичное дерево при помощи повторного применения алгоритма *Surface* всплытия Флойда сначала к самым мелким его поддеревьям от листьев и затем ко все более крупным. Листья тривиально упорядочены, поэтому можно начать с минимальных поддеревьев, содержащих несколько вершин, и укрупнять их, каждый раз полностью, применяя алгоритм всплытия до тех пор, пока таким образом не будет достигнут корень дерева. Заметим, что каждое из поддеревьев, к которым применяется алгоритм всплытия, удовлетворяет условию почти упорядоченности, поскольку упорядочивание проходит от листьев к корню. Именно таким способом в алгоритме 6.5 осуществляется формирование исходного почти упорядоченного дерева.

После того как дерево упорядочено, наибольший (наименьший) элемент оказывается в его корне. По алгоритму 6.5 найденный элемент меняют местами с самым последним листом в дереве (последний элемент рассматриваемого массива), дерево уменьшается на одну вершину и все готово для определения нового наибольшего (наименьшего) элемента множества при помощи следующего применения процедуры *Surface* всплытия Флойда. На рис. 6.5 показана полная последовательность перестановок и всплытий, которые происходят после формирования из исходного множества почти упорядоченного дерева и вплоть до того, как в этом дереве останется всего одна вершина, а исходное множество окажется отсортированным.

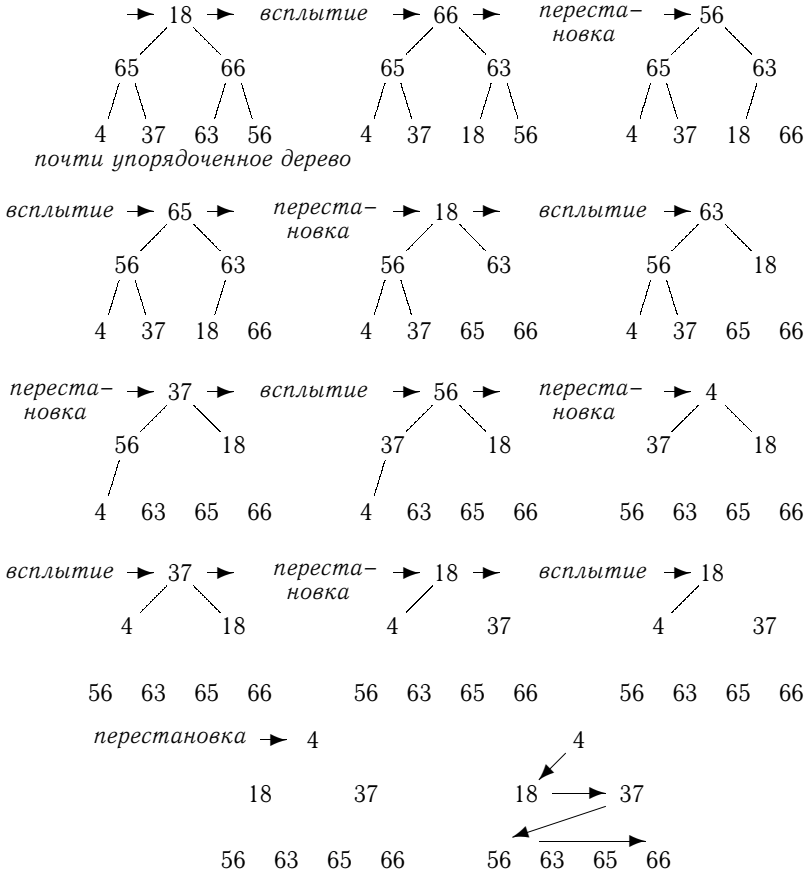



Рис. 6.5. Пример сортировки чисел 18, 4, 56, 65, 37, 63, 66 методом Флойда

Алгоритм 6.5.

$O(\log_2(n))$

```
Program Floid; {    }
uses CRT;
Const
  n = 1000; {   }
Type
  Vector = array[1..n] of Integer;
Var
  f :Text; {   }

procedure Init( var a: vector; n: integer);
{          }
var
  i: integer;
begin
  Randomize;
  for i:=1 to n do a[i]:=Random(100);
end;

procedure Surface( var a: Vector; i,k: integer);
{          }
var
  j,m,copy :Integer;
begin
  copy:=a[i];
  m:=2*i;
  while m<=k do begin
    if m=k then j:=m
    else if a[m]>a[m+1] then j:=m else j:=m+1;
    if a[j]>copy then begin
      a[i]:=a[j];
      i:=j;
      m:=2*i;
    end
    else break; {  }
  end;
  a[i]:=copy;
end;
```

```
procedure Sort( var a: Vector; n: integer);
{           }
var
  i,k,w :Integer;
begin
  {   }
  for i:=n div 2 downto 2 do Surface(a,i,n);
  {     }
  for k:=n downto 2 do begin
    Surface(a,1,k);
    {     }
    w:=a[k]; a[k]:=a[1]; a[1]:=w;
  end;
end;

Var {       }
  a : Vector; {    }
  i : Integer;
begin {       }
  Assign(f,'sort.out');
  Rewrite(f); {   }
  Init(a,n);
  { }
  for i:=1 to n do WriteLn(f,'a[',i:1,']=',a[i]:3);
  Sort(a,n);
  { }
  WriteLn(f);
  for i:=1 to n do WriteLn(f,'s[',i:1,']=',a[i]:3);
  Close(f);
end. {       }
```

Оценим общую сложность алгоритма сортировки данных рассматриваемым методом. Процедура *Surface* всплытия Флойда выполняется $n$ раз сначала для формирования исходного почти упорядоченного дерева (процедура применяется для каждой вершины дерева) и затем $n$ раз для всплытия каждого наибольшего (наименьшего) элемента в почти упорядоченном дереве. Так как сложность процедуры *Surface* всплытия Флойда составляет $O(\log_2(n))$, общая сложность алгоритма сортировки данных $a_1, a_2, \ldots, a_n$ равна $O(n \log_2(n))$.

Это — лучшая оценка, на что вообще можно надеяться при сортировках, в основу которых положены сравнения данных. Действительно, число возможных перестановок из элементов $a_1, a_2, \ldots, a_n$ равно $n!$ и только одна из них удовлетворяет условию нашей сортировки. Двоичный же поиск перестановки среди множества $n!$ перестановок требует $\log_2(n!)$ числа сравнений. Для упрощения воспользуемся формулой Стирлинга $n! \sim \sqrt{2\pi n}\, n^n e^{-n}$. Тогда $\log_2(n!) \sim O(n \log_2(n))$.

Задача. *Длина объединения отрезков*. Текстовый файл содержит целые числа: $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$. Данная последовательность чисел определяет на прямой $n$ отрезков $[a_i, b_i]$, $i = 1, 2, \ldots, n$. Найти длину объединения указанных отрезков. Исходные данные представлены в текстовом файле со следующей структурой. Первая строка файла: $n$ — количество отрезков. Вторая, третья и т. д. строки файла содержат целые числа $a_i, b_i$ — границы соответствующих отрезков. Результаты расчетов длины объединения отрезков сохранить в текстовом файле.

Пример файла исходных данных

```
 3
 0  2
-1  1
 0  1
```

Выходной файл для данного примера

```
3
```

Решение. Алгоритм 5.6 решения задачи основывается на предварительной сортировке абсцисс $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$ в массиве $ab[1..2n]$. Создается вспомогательный массив $g[1..2n]$, который поддерживает после сортировки массива $ab[1..2n]$ отношение границ отрезков: $g[i] = 1$ — левая граница, $g[i] = 0$ — правая граница. Вычисление завершается простым просмотром массива $ab[1..2n]$ за линейное время. Общая сложность алгоритма определяется сложностью сортировки данных. В данном случае используется алгоритм пузырьковой сортировки сложности $O(n^2)$.

Алгоритм 6.6.

```
Program MeasureLength; { }
uses CRT,DOS;
Const
  n_max=100;
Type
```

```
  Vector=array[0..2*n_max] of Integer;
Var
  f  :Text; { }
  ab :Vector; {          }
  g  :Vector; {        }

Procedure Measure( Var m: LongInt; n:Integer );
{ }
Var
  i,c :Integer;
begin
  ab[0]:=ab[1];
  m:=0; { }
  c:=0; {  }
  for i:=1 to n do begin
    if c<>0 then m:=m+ab[i]-ab[i-1];
    if g[i]=1 {  } then c:=c+1 else c:=c-1;
  end;
end;

Procedure SortBubble( n:Integer );
{              }
Var
i,j,w :Integer;
begin
  for i:=1 to n do begin
    for j:=1 to n-i do begin
      if ab[j] > ab[j+1] then begin
        w:=ab[j]; ab[j]:=ab[j+1]; ab[j+1]:=w;
        w:=g[j]; g[j]:=g[j+1]; g[j+1]:=w;
      end;
    end;
  end;
end;

Var {      }
  i,k :Integer;
  n  :Integer; {  }
  m  :LongInt; {  }
```

```
begin {       }
  Assign(f,'Measure.in');
  Reset(f); {   }
  Read(f,n); { }
  for i:=1 to n do begin
    k:=2*i;
    Read(f,ab[k-1],ab[k]);
    g[k-1]:=1; { }
    g[k]:=0; { }
  end;
  Close(f);
  Assign(f,'Measure.out');
  Rewrite(f); {   }
  SortBubble(2*n);
  Measure(m,2*n);
  WriteLn(f,m); { }
  Close(f);
end. {       }
```

## § 6.5. Последовательный поиск

Задача поиска является фундаментальной в алгоритмах на дискретных структурах. Удивительно то, что, накладывая незначительные ограничения на структуру исходных данных, можно получить множество разнообразных стратегий поиска различной степени эффективности.

При последовательном поиске подразумевается исследование элементов множества $a_1, a_2, \ldots, a_n$ в том порядке, в котором они встречаются. *«Начни сначала и продвигайся, пока не найдешь нужный элемент; тогда остановись»*. Такая последовательная процедура является очевидным способом поиска. Алгоритм 6.7 выполняет последовательный поиск элемента $z$ в множестве $a_1, a_2, \ldots, a_n$. Несмотря на свою простоту, последовательный поиск содержит ряд очень интересных идей.

Алгоритм 6.7.

```
c = 0;      z
for i = 1 to n do begin
    if z = a_i then do begin
```

```
            c = 1;
            break;
        end;
    end;
    if c = 1 then
    else
```

Оценим среднюю сложность поиска элементов множества $a_1, a_2, \ldots, a_n$. Для нахождения $i$-го элемента $a_i$ требуется $i$ сравнений. Для вычисления же среднего времени поиска необходимо задать информацию о частоте обращения к каждому элементу множества. Будем предполагать, что частота обращения распределена равномерно, т. е. что ко всем элементам обращаются одинаково часто. Тогда средняя сложность поиска элемента множества является $\frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2} = O(n)$ линейной.

Рассмотрим распределение частот обращения к элементам в общем случае. Пусть $\rho_i$ обозначает частоту (распределение вероятностей) обращения к элементу $a_i$, где $\rho_i \geqslant 0$ и $\sum_{i=1}^{n} \rho_i = 1$. В этом случае средняя сложность (математическое ожидание) поиска элемента будет равна $\sum_{i=1}^{n} i\rho_i$. Хорошим приближением распределения частот к действительности является закон Зипфа: $\rho_i = c/i$ для $i = 1, 2, \ldots, n$. (Дж. К. Зипф заметил, что $n$-е наиболее употребительное в тексте на естественном языке слово встречается с частотой, приблизительно обратно пропорциональной $n$.) Нормирующая константа $c$ выбирается так, что $\sum_{i=1}^{n} \rho_i = 1$. Пусть элементы множества $a_1, a_2, \ldots, a_n$ упорядочены согласно указанным частотам. Тогда $c = 1/\bigl(\sum_{i=1}^{n} 1/i\bigr) = 1/H_n \approx 1/\ln(n)$ и среднее время успешного поиска составит $\sum_{i=1}^{n} i\rho_i = \sum_{i=1}^{n} c = nc = n/H_n \approx n/\ln(n)$, что много меньше $(n+1)/2$.

Последний пример показывает, что даже простой последовательный поиск требует выбора разумной структуры данных множества, который повышал бы эффективность работы алгоритма. Более того, это общепринятая стратегия, состоящая в том, чтобы время от времени переупорядочивать данные, для большинства

последовательных файлов во внешней памяти, когда последовательная природа файла диктуется техническими характеристиками носителя информации.

Алгоритм последовательного поиска данных одинаково эффективно выполняется при размещении множества $a_1, a_2, \ldots, a_n$ на смежной или связанной памяти.

## § 6.6. Логарифмический поиск

Логарифмический (бинарный или метод деления пополам) поиск данных применим к сортированному множеству элементов $a_1 < a_2 < \ldots < a_n$, размещение которого выполнено на смежной памяти. Для большей эффективности поиска элементов надо, чтобы пути доступа к ним стали более короткими, чем просто последовательный перебор. Наиболее очевидный метод: начать поиск со среднего элемента, т. е. выполнить сравнение с элементом $a_{\lfloor \frac{1+n}{2} \rfloor}$. Результат сравнения позволит определить, в какой половине последовательности $a_1, a_2, \ldots, a_{\lfloor \frac{1+n}{2} \rfloor}, \ldots, a_n$ продолжить поиск, применяя к ней ту же процедуру, и т. д. Основная идея бинарного поиска довольно проста, однако *«для многих хороших программистов не одна попытка написать правильную программу закончилась неудачей»*. Чтобы досконально разобраться в алгоритме, лучше всего представить данные $a_1 < a_2 < \ldots < a_n$ в виде двоичного дерева сравнений, которое отвечает бинарному поиску.

Двоичное дерево называется *деревом сравнений*, если для любой его вершины (корня дерева или корня поддерева) выполняется условие:

$$\left\{ \begin{array}{c} \text{Вершины} \\ \text{левого} \\ \text{поддерева} \end{array} \right\} < \left\{ \begin{array}{c} \text{Вершина} \\ \text{корня} \end{array} \right\} < \left\{ \begin{array}{c} \text{Вершины} \\ \text{правого} \\ \text{поддерева} \end{array} \right\}$$

Пусть на очередном шаге деления пополам оказалось, что необходимо выполнить поиск среди элементов $a_i < a_{i+1} < \ldots$ $\ldots < a_j$. В качестве корня принимается элемент $a_{\lfloor \frac{i+j}{2} \rfloor}$, где $\lfloor \frac{i+j}{2} \rfloor$ — наибольшее целое, меньшее или равное $\frac{i+j}{2}$. Левое поддерево располагается в векторе $a_i, a_{i+1}, \ldots, a_{\lfloor \frac{i+j}{2} \rfloor - 1}$, а правое поддерево — в векторе $a_{\lfloor \frac{i+j}{2} \rfloor + 1}, \ldots, a_{j-1}, a_j$. На рис. 6.6 показан пример двоичного дерева сравнений, ребра которого неявно вы-

ражаются рассмотренными выше отношениями между индексами элементов.

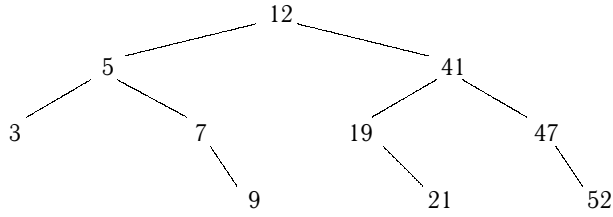



Рис. 6.6. Пример дерева сравнений, отвечающего бинарному поиску среди сортированных элементов: $3, 5, 7, 9, 12, 19, 21, 41, 47, 52$

Поиск элемента $z$ среди $a_1 < a_2 < \ldots < a_n$ методом деления пополам представлен в алгоритме 6.8. Средняя сложность бинарного поиска среди элементов $a_1 < a_2 < \ldots < a_n$ сравнима с высотой двоичного дерева (рис. 6.6). В худшем случае искомый элемент может оказаться либо на последнем уровне, либо вообще не будет найден. На каждом уровне необходимо выполнить определенное число сравнений. В п. 6.4 установлено, что уровней в дереве $\lceil \log_2(n+1) \rceil$ . Значит, сложность поиска является *логарифмической* $O(\log_2(n))$, что оправдывает и название самого метода поиска.

Алгоритм 6.8. $z \quad a_1 < a_2 < \quad < a_n$

$find = 0$; { }
$i = 1$; { }
$j = n$; { }
**while** $i \leqslant j$ **do begin**
  $m = \lfloor \frac{i+j}{2} \rfloor$; { }
  **if** $z = a_m$ **then begin**
    $find = 1$; { }
    **break;**
  **end;**
  **else if** $z > a_m$ **then** $i = m + 1$; { }
  **else** $j = m - 1$; { }
**end;**
**if** $find = 1$ **then**
**else**

Необходимо отметить, что рассмотренный метод бинарного поиска предназначен, главным образом, для сортированных

элементов $a_1 < a_2 < \ldots < a_n$ на смежной памяти фиксированного $n$ размера. Если же размерность вектора динамически меняется, то экономия от использования бинарного поиска не покроет затрат на поддержание упорядоченного расположения $a_1 < a_2 < \ldots < a_n$.

## § 6.7. Сортировка с вычисляемыми адресами

Пусть $a_1, a_2, \ldots, a_n$ — исходная последовательность сортируемых целых чисел и $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ — упорядочивающая перестановка этих элементов $a_{\pi_1} < a_{\pi_2} < \ldots < a_{\pi_n}$. Принятое ограничение, что $a_i$ — целые числа, не ведет к потере общности рассматриваемых ниже алгоритмов. Сортированная последовательность $a_i$ подсказывает очевидный способ использования значений этих элементов в качестве индексов (*адресов*) их расположения в массиве $b_r, b_{r+1}, \ldots, b_s$, где $b_{a_i} = a_i$. Получим упорядоченную последовательность $b_r < b_{r+1} < \ldots < b_s$, а значит, и упорядоченную последовательность исходных данных $a_1, a_2, \ldots, a_n$.

В качестве примера рассмотрим частный случай сортировки элементов последовательности $a_1, a_2, \ldots, a_7$, равных соответственно $6, 3, 5, 7, 2, 4, 1$ — различные целые числа от 1 до 7. Значение каждого элемента $a_i$ показывает его место в сортированном списке $b_1 < b_2 < \ldots < b_7$, где $b_i$ определяются в цикле

$$\textsf{for } i = 1 \textsf{ to } 7 \textsf{ do } b_{a_i} = a_i.$$

Сложность данного метода сортировки является линейной $O(n)$, где $n = 7$. Фактически, значения $a_i$ определяют перестановку $\pi = (\pi_1, \pi_2, \ldots, \pi_n) = (7, 5, 2, 6, 3, 1, 4)$, которая упорядочивает элементы $a_{\pi_1} < a_{\pi_2} < \ldots < a_{\pi_n}$. Значения $\pi_i$ устанавливаются в цикле

$$\textsf{for } i = 1 \textsf{ to } 7 \textsf{ do } \pi_{a_i} = i.$$

Отметим, что перестановка $\pi$ является обратной к исходному расположению элементов $a_1, a_2, \ldots, a_n$ (см. п. 2.14). Сортировка же стала возможной благодаря тому, что значения исходных данных $6, 3, 5, 7, 2, 4, 1$ заполняют сплошной интервал последовательности натуральных чисел. Рассмотрим обобщение идеи сортировки с вычисляемыми адресами на случай произвольных целых $a_1, a_2, \ldots, a_n$. Алгоритм сортировки существенно упрощается, если все значения $a_1, a_2, \ldots, a_n$ различные.

### Значения $a_1, a_2, \ldots, a_n$ — различные

Реализация метода сортировки представлена в алгоритме 6.9. Временный массив $b_r, b_{r+1}, \ldots, b_s$, где $r = \min(a_1, a_2, \ldots, a_n)$ и $s = \max(a_1, a_2, \ldots, a_n)$, используется для сортированной

упаковки элементов $a_1, a_2, \ldots, a_n$. Свободные места в массиве $b_r, b_{r+1}, \ldots, b_s$ инициализируются значением $s+1$, отличным от значений элементов $a_1, a_2, \ldots, a_n$.

Алгоритм 6.9.

$a_1, a_2, \quad , a_n$

```
{                  a_1, a_2,     , a_n}
r = s = a_1;
for i = 2 to n do begin
   if r > a_i then r = a_i;
   else if s < a_i then s = a_i;
end;
{ b_i   s + 1     a_i}
for i = r to s do b_i = s + 1;
{   a_1, a_2,     , a_n}
for i = 1 to n do b_{a_i} = a_i;
{ a_1, a_2,     , a_n   b_r, b_{r+1},     , b_s}
k = 0;
for i = r to s do begin
   if b_i ≠ s + 1 then begin
      k = k + 1;
      a_k = b_i;
   end;
end.
```

Сортированный вектор $a_1 < a_2 < \ldots < a_n$ является результатом последовательного просмотра массива $b_r, b_{r+1}, \ldots, b_s$ при удалении из него оставшихся незанятыми элементов, равных значению $s+1$. Алгоритм не содержит вложенных циклов, а значит, сложность его линейная: $O(n)$.

### Допускаются одинаковые значения среди $a_1, a_2, \ldots, a_n$

Реализация метода представлена в алгоритме 6.10. Наличие одинаковых элементов среди $a_1, a_2, \ldots, a_n$, например, $a_i = a_j$, при упаковке $b_{a_i} = a_i$ и $b_{a_j} = a_j$ ведет к потере данных в исходном множестве $a_1, a_2, \ldots, a_n$. Ситуация, когда одновременно несколько элементов претендуют на одно место, называется *коллизией*. Такие элементы необходимо переразмещать на свободные места, сохраняя свойства сортировки с вычисляемыми адресами. С этой целью на основании величин $a_1, a_2, \ldots, a_n$ рассчитывается

вектор индексов $d_r, d_{r+1}, \ldots, d_s$ сортированной упаковки данных $a_1, a_2, \ldots, a_n$ в массиве $b_1, b_2, \ldots, b_n$. В алгоритме 6.9 роль индексов упаковки могли выполнять непосредственно сами значения элементов $a_1, a_2, \ldots, a_n$, так как они были различные. В данном случае значения $d_r, d_{r+1}, \ldots, d_s$ настраиваются таким образом, что одинаковые по величине элементы из $a_1, a_2, \ldots, a_n$ оказываются смежными при их упаковке в массиве $b_1, b_2, \ldots, b_n$. Вектор $c_r, c_{r+1}, \ldots, c_s$ используется для подсчета количества элементов каждого значения среди $a_1, a_2, \ldots, a_n$ и формирования индексов упаковки $d_r, d_{r+1}, \ldots, d_s$.

Алгоритм 6.10.

$a_1, a_2, \quad , a_n$

{ $a_1, a_2, \quad , a_n$}
$r = s = a_1$;
for $i = 2$ to $n$ do begin
  if $r > a_i$ then $r = a_i$;
  else if $s < a_i$ then $s = a_i$;
end;
{ $a_i$}
for $i = r$ to $s$ do $c_i = 0$;
for $i = 1$ to $n$ do $c_{a_i} = c_{a_i} + 1$;
{ $d_r, d_{r+1}, \quad , d_s$}
$d_r = 1$;
for $i = r + 1$ to $s$ do $d_i = d_{i-1} + c_{i-1}$;
{ $a_1, a_2, \quad , a_n \quad b_1, b_2, \quad , b_n$}
for $i = 1$ to $n$ do begin
  $k = a_i$;
  $b_{d_k} = a_i$;
  $d_k = d_k + 1$;
end.

Результирующий сортированный вектор исходных данных $a_1, a_2, \ldots, a_n$ располагается в массиве $b_1 \leqslant b_2 \leqslant \ldots \leqslant b_n$. Алгоритм не содержит вложенных циклов, а значит, сложность его линейная $O(n)$. Сортировка с вычисляемыми адресами является очень быстрым методом, но она может быть крайне неэффективной при больших значениях $s - r$ с точки зрения использования оперативной памяти, необходимой для хранения временных массивов данных $c_r, c_{r+1}, \ldots, c_s$ и $d_r, d_{r+1}, \ldots, d_s$.

Глава 7

# ТЕОРИЯ ГРАФОВ. АЛГОРИТМЫ НА ГРАФАХ

Множество самых разнообразных задач естественно формулируется в терминах точек и связей между ними, т. е. в терминах графов. Так, например, могут быть сформулированы задачи составления расписания, анализа сетей в электротехнике, анализа цепей Маркова в теории вероятностей, в программировании, в проектировании электронных схем, в экономике, в социологии и т. д. Поэтому эффективные алгоритмы решения задач теории графов имеют большое практическое значение.

## § 7.1. Основные понятия и определения

Определение. *Конечным графом* называется тройка $\Gamma = (X, U, \Phi)$, где $X$ — конечное множество вершин; $U$ — конечное множество ребер (дуг); $\Phi$ — отношение инцидентности; $X \cap U = \varnothing$. Отношение инцидентности $\Phi$ является трехместным отношением $\Phi(x, u, y)$, где $x, y \in X$, $u \in U$, которое может либо выполняться (быть истинным), либо не выполняться (быть ложным) и удовлетворяет свойствам:

1. $\forall u \in U\ \exists x, y \in X\ \ \Phi(x, u, y)$ — ребро соединяет пару вершин;
2. $\Phi(x, u, y) \wedge \Phi(x', u, y') \Rightarrow (x = x' \wedge y = y') \vee (x = y' \wedge y = x')$ — ребро и соответствует не более чем одной паре вершин $x, y$.

**Графическое представление графов**

| Элементы графов | Геометрические элементы |
|---|---|
| 1. $x \in X$ — вершина. | 1. x ■ — точка в пространстве. |
| 2. $\Phi(x, u, y) \wedge \neg\Phi(y, u, x)$ — ориентированное ребро, дуга. | 2. x ■⟶ y — направленный отрезок. |
| 3. $\Phi(x, u, y) \wedge \Phi(y, u, x)$ — неориентированное ребро. | 3. x ■——■ y — отрезок. |
| 4. $\Phi(x, u, x)$ — петля. | 4. x ■◯ — замкнутый отрезок. |

Определение. $\Gamma_1(X_1, U_1, \Phi_1)$ и $\Gamma_2(X_2, U_2, \Phi_2)$ называются *изоморфными* ($\Gamma_1 \cong \Gamma_2$), если существуют два взаимно однозначных соответствия $\varphi : X_1 \to X_2$ и $\psi : U_1 \to U_2$, со-

храняющих отношение инцидентности: $\Phi_2(\varphi(x_1), \psi(u_1), \varphi(y_1)) = \Phi_1(x_1, u_1, y_1)$.

Из определения следует, что изоморфные графы можно одинаково изображать графически и отличаться они будут только метками вершин (рис. 7.2).

Определение. Граф называется *ориентированным* (*орграф*), если каждое его ребро ориентировано: $\forall x \neq y \in X\ \forall u \in U\ \Phi(x, u, y) \Rightarrow \neg\Phi(y, u, x)$. Иногда удобно преобразовать неориентированный граф в ориентированный — заменой каждого неориентированного ребра парой ориентированных ребер с противоположным направлением.

Определение. *Подграфом* графа $\Gamma(X, U, \Phi)$ называется такой граф $\Gamma'(X', U', \Phi)$, что $X' \subseteq X, U' \subseteq U$. Обозначают $\Gamma' \subseteq \Gamma$.

Определение. Граф называется *псевдографом*, если в нем допускаются петли и кратные ребра, т. е. две вершины могут быть соединены более чем одним ребром. Псевдограф без петель называется *мультиграфом* (рис. 7.3).

Определение. Неориентированный граф называется *простым*, если он не имеет петель и любая пара вершин соединена не более чем одним ребром.

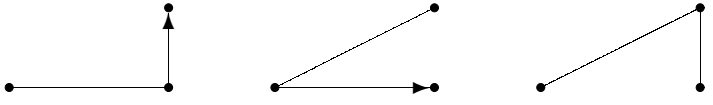


Рис. 7.1. Графы с тремя вершинами и двумя ребрами

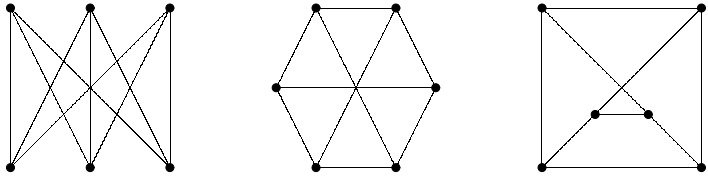


Рис. 7.2. Три изоморфных графа

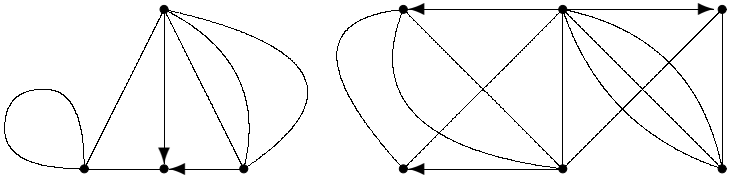


Рис. 7.3. Псевдограф (слева) и мультиграф (справа)

Определение. Простой граф называется *полным*, если каждая пара вершин соединена ребром. Такой граф с $n$ вершинами содержит $C_n^2$ ребер (рис. 7.4).

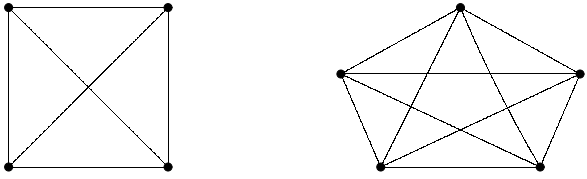


Рис. 7.4. Полные неориентированные графы

Определение. *Дополнением* простого графа $\Gamma$ называется граф $\overline{\Gamma}$, имеющий те же вершины, а его ребра являются дополнением $\Gamma$ до полного графа (рис. 7.5).

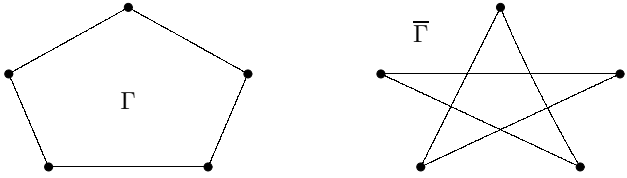



Рис. 7.5. Исходный граф $\Gamma$ и дополнительный $\overline{\Gamma}$

Определение. Граф называется *плоским* (*планарным*), если он может быть изображен на плоскости так, что все пересечения ребер являются его вершинами.

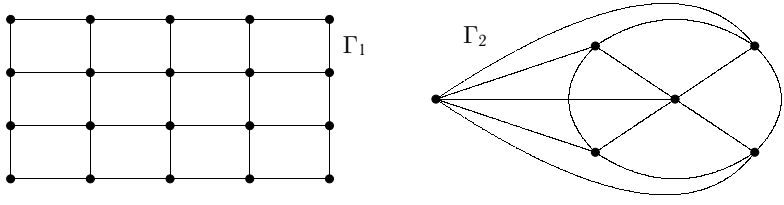



Рис. 7.6. Граф $\Gamma_1$ — плоский, а граф $\Gamma_2$ — неплоский

Определение. Если вершины $x$ и $y$ соединены ребром $u$, то говорят, что вершины $x, y$ *смежные*, а ребро $u$ инцидентно вершинам $x$ и $y$. Два ребра называются *смежными*, если они имеют общую вершину.

Определение. *Степенью вершины* графа называется количество ребер, инцидентных данной вершине. Вершина графа,

имеющая степень 0, называется *изолированной*, а если степень ее равна 1, то такая вершина называется *висячей*.

Определение. Граф называется *помеченным* (или перенумерованным), если его вершины отличаются друг от друга какими–либо метками (рис. 7.7).

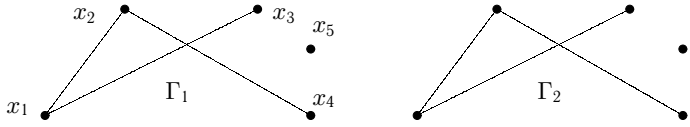



Рис. 7.7. Граф $\Gamma_1$ — помеченный, а граф $\Gamma_2$ — непомеченный. Вершины $x_3$, $x_4$ — висячие, а вершина $x_5$ — изхолированная

Определение. *Путь* (*маршрут*) на графе $\Gamma(X, U, \Phi)$ есть последовательность вершин и ребер $x_1u_1x_2u_2x_3 \dots x_nu_nx_{n+1}$, где $x_i \in X$, $u_i \in U$. Ребро $u_i$ соединяет вершину $x_i$ с вершиной $x_{i+1}$, т. е. выполняется отношение инцидентности $\Phi(x_i, u_i, x_{i+1})$.

- ◇ Маршрут называется *цепью*, если все его ребра различные.
- ◇ Маршрут называется *замкнутым*, если $x_1 = x_{n+1}$.
- ◇ Замкнутая цепь называется *циклом*.
- ◇ Цепь называется *простой*, если не содержит одинаковых вершин.
- ◇ Простая замкнутая цепь называется *простым циклом*.
- ◇ *Гамильтоновой цепью* называется простая цепь, содержащая все вершины графа.
- ◇ *Гамильтоновым циклом* называется простой цикл, содержащий все вершины графа.

Определение. Граф $\Gamma(X, U, \Phi)$ называется *связным*, если для всех $x, y \in X$ существует путь из вершины $x$ в вершину $y$ (вершины $x$ и $y$ связаны маршрутом). Связный ориентированный граф называется *сильно связным*. Орграф называется *слабо связным*, если соответствующий ему неориентированный граф (игнорируется ориентация ребер) связный (рис. 7.8).


Рис. 7.8. $\Gamma_1$ — слабо связный, $\Gamma_1$ — сильно связный

Определение. Связный неориентированный ациклический граф называется *деревом*, множество деревьев называется *лесом*.

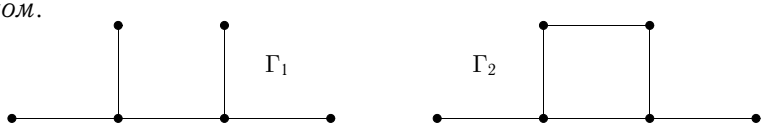



Рис. 7.9. $\Gamma_1$ — дерево, $\Gamma_2$ — не дерево

Большинство задач на графах касается определений компонент связности, поиска маршрутов, расстояний и т. п. Далее будут рассмотрены решения подобных вопросов. Однако при решении реальных задач соответствующие им графы весьма велики, и анализ возможен лишь с привлечением современной вычислительной техники. Поэтому конечной целью рассмотрения каждой из задач будет являться описание и реализация практического алгоритма решения данной задачи на ЭВМ.

## § 7.2. Представления графов

Наиболее известный и популярный способ представления графов состоит в геометрическом изображении точек (вершин) и линий (ребер) на бумаге. При численном решении задач на вычислительных машинах граф должен быть представлен дискретным способом. Существует довольно много способов такого рода представления графов. Однако простота использования представления графа, как и эффективность алгоритма, в основе которого он лежит, в полной мере зависит от конкретного выбора этого представления. Одно из направлений теории графов связано с их матричным представлением. Существуют различные виды матриц, ассоциированные с графами. Эти алгебраические формы используются для решения многих задач теории графов. Ниже рассматриваются две такие матричные формы и несколько нестандартных представлений, которые наиболее широко используются в алгоритмах на графах.

### 7.2.1. Матрица смежности графа

Определение. *Матрицей смежности* ориентированного помеченного графа с $n$ вершинами называется матрица $A = [a_{ij}]$, $i, j = 1, 2, \ldots, n$, в которой

$$a_{ij} = \begin{cases} 1, & \text{если существует ребро } (x_i, x_j), \\ 0, & \text{если вершины } x_i, x_j \text{ не связаны ребром } (x_i, x_j). \end{cases}$$

Матрица смежности однозначно определяет структуру графа. Примеры орграфа и его матрицы смежности приведены соответственно на рис. 7.10 и рис. 7.11. Отметим, что петля в матрице

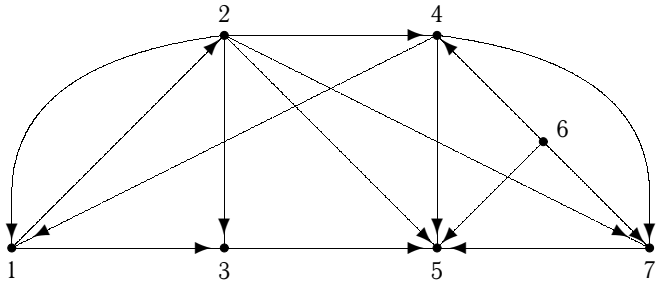



Рис. 7.10. Ориентированный граф

смежности может быть представлена соответствующим единичным диагональным элементом. Кратные ребра можно представить, позволив элементу матрицы быть больше 1, но это не принято, обычно же представляют каждый элемент матрицы одним двоичным разрядом.

$A =$

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Рис. 7.11. Матрица смежности ориентированного графа (рис. 7.10)

### 7.2.2. Матрица инцидентности графа

Определение. *Матрицей инцидентности для неориентированного графа* с $n$ вершинами и $m$ ребрами называется матрица $B = [b_{ij}]$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, строки которой соответствуют вершинам, а столбцы — ребрам. Ее элементы

$$b_{ij} = \begin{cases} 1, & \text{если вершина } x_i \text{ инцидентна ребру } u_j; \\ 0, & \text{если вершина } x_i \text{ не инцидентна ребру } u_j. \end{cases}$$

Определение. *Матрицей инцидентности для ориентированного графа* с $n$ вершинами и $m$ ребрами называется матрица $B = [b_{ij}]$, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$, строки которой соответствуют вершинам, а столбцы — ребрам. Ее элементы

$$b_{ij} = \begin{cases} +1, & \text{если ребро } u_j \text{ выходит из вершины } x_i; \\ -1, & \text{если ребро } u_j \text{ входит в вершину } x_i; \\ 0, & \text{если вершина } x_i \text{ не инцидентна ребру } u_j. \end{cases}$$

Матрица инцидентности однозначно определяет граф. На рис. 7.12 представлена такая матрица для орграфа на рис. 7.10.

$B =$

| | $\frac{1}{2}$ | $\frac{1}{3}$ | $\frac{2}{1}$ | $\frac{2}{3}$ | $\frac{2}{4}$ | $\frac{2}{5}$ | $\frac{2}{7}$ | $\frac{3}{5}$ | $\frac{4}{1}$ | $\frac{4}{5}$ | $\frac{4}{7}$ | $\frac{6}{4}$ | $\frac{6}{5}$ | $\frac{6}{7}$ | $\frac{7}{5}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | +1 | +1 | −1 | 0 | 0 | 0 | 0 | 0 | −1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | −1 | 0 | +1 | +1 | +1 | +1 | +1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | −1 | 0 | −1 | 0 | 0 | 0 | +1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | −1 | 0 | 0 | 0 | +1 | +1 | +1 | −1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | −1 | 0 | −1 | 0 | −1 | 0 | 0 | −1 | 0 | −1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | +1 | +1 | +1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | −1 | 0 | 0 | 0 | −1 | 0 | 0 | −1 | +1 |

Рис. 7.12. Матрица инцидентности ориентированного графа (рис. 7.10)

### 7.2.3. Матрица весов графа

Определение. Граф называется *взвешенным*, если каждому его ребру сопоставлено число. Простой взвешенный граф может быть представлен своей матрицей весов $W = [w_{ij}]$, где $w_{ij}$ — вес ребра, соединяющего вершины $i, j = 1, 2, \ldots, n$. Веса несуществующих ребер полагают равными $\pm\infty$ или 0 в зависимости от приложений. Заметим, что матрица весов является простым обобщением матрицы смежности.

### 7.2.4. Список ребер графа

При описании графа списком его ребер каждое ребро представляется парой инцидентных ему вершин. Это представление можно реализовать двумя массивами: $r = (r_1, r_2, \ldots, r_m)$ и $t = (t_1, t_2, \ldots, t_m)$, где $m$ — число ребер в графе. Каждый элемент в массиве есть метка вершины, а $i$-е ребро графа выходит из

вершины $r_i$ и входит в вершину $t_i$. Например, соответствующие массивы представления графа на рис. 7.10 имеют вид

$$r = (\ 1,\ 1,\ 2,\ 2,\ 2,\ 2,\ 2,\ 3,\ 4,\ 4,\ 4,\ 6,\ 6,\ 6,\ 7),$$
$$t = (\ 2,\ 3,\ 1,\ 3,\ 4,\ 5,\ 7,\ 5,\ 1,\ 5,\ 7,\ 4,\ 5,\ 7,\ 5).$$

• *Интересно, что данное представление позволяет легко описать петли и кратные ребра.*

### 7.2.5. Структура смежности графа

Ориентированный или неориентированный граф может быть однозначно представлен структурой смежности своих вершин. Структура смежности состоит из списков $Adj[x]$ вершин графа, смежных с вершиной $x$. Списки $Adj[x]$ составляются для каждой вершины графа. В качестве примера на рис. 7.13 представлена структура смежности графа на рис. 7.10. Структуры смежности могут быть удобно реализованы массивом из $n$ (число вершин в графе) линейно связанных списков. Каждый список содержит вершины, смежные с вершиной, для которой составляется список. Хранение же списков смежности на сцепленной памяти желательно в алгоритмах, в основе которых лежат операции добавления и удаления вершин из списков. Следует отметить, что во многих задачах на графах выбор представления является решающим для эффективности алгоритмов.

| $x$ | $Adj[x]$ |
|---|---|
| 1 | 2, 3 |
| 2 | 1, 3, 4, 5, 7 |
| 3 | 5 |
| 4 | 1, 5, 7 |
| 5 | - |
| 6 | 4, 5, 7 |
| 7 | 5 |

Рис. 7.13

## § 7.3. Метод поиска в глубину

Один из наиболее естественных способов систематического исследования всех вершин графа исходит из процедуры прохождения графа методом поиска с возвращением, который исследует граф в глубину (см. п. 5.1). На неориентированном графе $\Gamma(X, U, \Phi)$ поиск в глубину осуществляется следующим образом. Когда посещаем вершину $x \in X$, то далее идем по одному из ребер $(x, y)$, инцидентному вершине $y \in X$. Если вершина $y$ уже пройдена (посещалась ранее), то возвращаемся в $x$ и выбираем другое ребро. Если вершина $y$ не пройдена, то заходим в нее и применяем процесс прохождения рекурсивно уже с вершиной $y$. Если все ребра, инцидентные вершине $x$, просмотрены, то идем назад по ребру $(s, x)$, по которому пришли в $x$, и продолжаем исследование ребер, инцидентных вершине $s \in X$. Процесс

заканчивается, когда попытаемся вернуться из вершины, с которой начали просмотр графа.

Поиск в глубину можно также осуществлять и на ориентированном графе. Если граф ориентированный, то, находясь в узле $x$, необходимо выбирать ребро $(x, y)$, только выходящее из $x$. Исследовав все ребра, выходящие из $y$, возвращаемся в $x$ даже тогда, когда в $y$ входят другие ребра, еще не рассмотренные. Данная техника просмотра в глубину полезна в практических приложениях при определении различных свойств как ориентированных, так и неориентированных графов.

Метод поиска в глубину на простом неориентированном графе представлен в алгоритме 7.1. Рекурсивная процедура $Depth(x, w)$ осуществляет поиск в глубину на графе $\Gamma(X, U, \Phi)$, содержащем $x \in X$, и строит для графа дерево $T$ поиска, которое является ориентированным остовным деревом $\Gamma_0(X, T, \Phi)$ (если исходный граф не связен, то $\Gamma_0$ будет лесом); $w \in X$ является отцом $x \in X$ в строящемся дереве, где $x$ — исследуемая вершина. Граф задан структурой смежности $Adj[x]$, где $Adj[x]$ означает множество вершин, смежных с $x \in X$. Элементы $T$ — это ребра строящегося дерева поиска, а элементы $B$ — это обратные ребра, которые не могут принадлежать $\Gamma_0$, так как они ведут назад в пройденные ранее вершины. Заметим, что обратное ребро должно идти от потомка к предку по дереву поиска. Чтобы отличить уже пройденные вершины от непройденных, вводится вектор $Mark[x]$ меток вершин, которые постепенно нумеруются от 1 до $|X|$ по мере того, как попадаем в них. Сначала полагается $Mark[x] = 0$ для всех $x \in X$ в знак того, что ни одна вершина не пройдена, и когда попадаем в вершину $x$ первый раз, $Mark[x]$ получает ненулевое значение. Ребро $(x, v) \in T$, если метка вершины $Mark[v] = 0$. Если же $Mark[v] \neq 0$, то условием того, что $(x, v) \in B$ будет обратным ребром, являются соотношения $Mark[v] < Mark[x]$ и $v \neq w$. Условие $Mark[v] < Mark[x]$ означает, что вершина $v$ была пройдена раньше вершины $x$. Поэтому ребро $(x, v) \in B$ будет обратным, если оно не является ребром дерева $T$, пройденным от отца $w$ к $x$, т. е. $v \neq w$.

• *Сложность поиска в глубину*. Поскольку для каждой вершины, которую проходим впервые, выполняется обращение к процедуре *Depth* ровно один раз, то всего обращений будет $|X|$. При каждом обращении количество производимых действий пропорционально числу ребер, инцидентных рассматриваемой вершине. Поэтому сложность поиска составляет $O(|X| + |U|)$.

Алгоритм 7.1.

```
for v ∈ X do Mark[v] = 0;
count = 0;
T = ∅;
B = ∅;
for v ∈ X do if Mark[v] = 0 then Depth(v, 0);
Procedure Depth(x, w);
    count = count + 1;
    Mark[x] = count;
    for v ∈ Adj[x] do begin
        if Mark[v] = 0 then begin
            T = T ∪ {(x, v)}; { }
            Depth(v, x);
        end
        else if Mark[v] < Mark[x] and v ≠ w then
            B = B ∪ {(x, v)}; { }
    end;
end.
```

Программная реализация алгоритма 7.1 представлена алгоритмом 7.2. Реализация близко соответствует основному алгоритму 7.1. Программа представлена тремя процедурами *Init*, *Depth*, *WayDepth*, где *WayDepth* — основная программа поиска в глубину; *Depth* — рекурсивная процедура поиска, один к одному соответствующая аналогичной процедуре в алгоритме 7.1; *Init* — процедура контроля исходных данных и изменения меток вершин. Изменение нумерации меток вершин является существенным для алгоритма. Новые метки вершин — это натуральные числа от 1 до $|X|$. Данная нумерация позволяет обращаться к элементам массивов, содержащих информацию о вершинах, по номерам соответствующих вершин. Такой прием позволяет очень близко подойти в программной реализации структуры смежности $Adj[x]$ к ее множественному описанию. В этом случае множественное описание выражения for $v \in Adj[x]$ do ... в программной реализации представляется как for $i = 1$ to $Nbr[x]$ do $v = Adj[Fst[x] + i]$..., где $Nbr[x]$ — количество вершин в структуре смежности для вершины $x \in X$; $Adj[x]$ — вектор, содержащий все вершины структуры смежности по строкам; $Fst[x] + 1$ — номер первой вершины в структуре смежности для соответствующей вершины $x \in X$, тогда $Fst[x] + i$ — номер $i$-й

вершины в структуре смежности для $x \in X$. Например, для следующей структуры смежности $Adj[x]$ графа:

| $x$ | $Adj[x]$ |
|---|---|
| 1 | 2, 3, 5 |
| 2 | 1, 3, 4 |
| 3 | 1, 2, 4, 5 |
| 4 | 2, 3 |
| 5 | 1, 3, 6, 7 |
| 6 | 5, 7 |
| 7 | 5, 6 |

соответствующие массивы в программной реализации принимают вид

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $Nbr[x]$ | 3 | 3 | 4 | 2 | 4 | 2 | 2 |
| $Fst[x]$ | 0 | 3 | 6 | 10 | 12 | 16 | 18 |
| $Adj[x]$ | 2 3 5 | 1 3 4 | 1 2 4 5 | 2 3 | 1 3 6 7 | 5 7 | 5 6 |

Исходные данные для расчета по программе алгоритма 7.2 представляются в текстовом файле со следующей структурой смежности $Adj[x]$:

- в первой строке файла содержится количество строк в структуре смежности, которое равно числу вершин в графе;
- далее для каждой вершины в отдельной строке указывается номер самой вершины, количество вершин смежных с данной и список этих вершин.

Рассмотрим пример расчета по программе алгоритма 7.2 обхода графа, представленного на рис. 7.14. Жирными линиями отмечены ребра, которые были пройдены во время обхода графа в глубину, тонкими — обратные ребра.

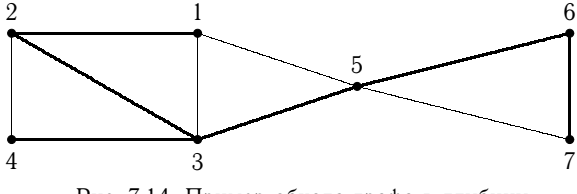



Рис. 7.14. Пример обхода графа в глубину

Исходные данные структуры смежности графа рис. 7.14 задаются в текстовом файле Depth.in:

```
7
1  3    2  3  5
2  3    1  3  4
3  4    1  2  4  5
4  2    2  3
5  4    1  3  6  7
6  2    5  7
7  2    5  6
```

Результаты расчетов сохраняются в выходном файле Depth.out со следующей структурой:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 3 | 4 | 3 | 5 | 5 | 6 | 7 | $x_i$ |
| 2 | 3 | 1 | 4 | 2 | 5 | 1 | 6 | 7 | 5 | $y_i$ |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | $T \vee B$ |

Каждая колонка таблицы выходного файла соответствует ребру $(x_i, y_i)$ прохода графа, где последнее значение является признаком 1 или 0, что отвечает при проходе либо основному ребру, либо обратному.

Алгоритм 7.2.

```
Program PgmWayDepth; {  }
uses CRT,DOS;
Const
  nVertex=100; { }
  nAdjacent=1000; {  }
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeAdjacent=array[1..nAdjacent] of Integer;
Var
  f  :Text; {}
  n  :Integer; {}
  nT :Integer; {  }
  Adj :TypeAdjacent; { }
  Fst :TypeVertex; {  }
  Nbr :TypeVertex; {  }
  Vtx :TypeVertex; { }
  Mark :TypeVertex; {  }
  T  :TypeVertex; {  }
```

```
  B  :TypeVertex; {    }

Procedure Init( Var yes :Boolean );
  { }
  {    }
  {          }
Var
  i,j,m :Integer;
begin
  for i:=1 to n do
    for j:=1 to Nbr[i] do begin
      yes:=FALSE;
      for m:=1 to n do
        if Adj[Fst[i]+j]=Vtx[m] then begin
          yes:=TRUE;
          Adj[Fst[i]+j]:=m;
          break;
        end;
      if not yes then exit;
    end;
end;

Procedure Depth( x,u:Integer; var count :Integer);
  { }
Var
i,v :Integer;
begin
  count:=count+1;
  Mark[x]:=count;
  for i:=1 to Nbr[x] do begin
    v:=Adj[Fst[x]+i];
    if Mark[v]=0 then begin
      nT:=nT+2; T[nT-1]:=x; T[nT]:=v;
      B[nT div 2]:=1; Прямое ребро
      Depth(v,x,count);
    end
    else if (Mark[v]<Mark[x]) and (v<>u) then begin
        { }
      nT:=nT+2; T[nT-1]:=x; T[nT]:=v;
```

```
        B[nT div 2]:=0; { }
      end;
    end;
end

Procedure WayDepth; { }
Var
  v,count :Integer;
begin
  nT:=0; {       }
  count:=0;
  for v:=1 to n do Mark[v]:=0;
  for v:=1 to n do if Mark[v]=0 then Depth(v,0,count);
end;
Var {       }
  i,j :Integer;
  yes :Boolean;
begin {       }
  Assign(f,'Depth.in');
  Reset(f); {   }
  {    }
  Read(f,n); {   }
  Fst[1]:=0; {    }
  for i:=1 to n do begin
    Read(f,Vtx[i]); { }
    Read(f,Nbr[i]); {   }
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]);
          { }
    Fst[i+1]:=Fst[i]+Nbr[i];
          {      }
  end;
  Close(f);
  Assign(f,'Depth.out');
  Rewrite(f); {   }
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа!');
    Close(f);
    exit;
```

```
  end;
  WayDepth;
  for i:=1 to nT div 2 do Write(f,Vtx[T[2*i-1]]:3); Writeln(f);
  for i:=1 to nT div 2 do Write(f,Vtx[T[2*i]]:3); Writeln(f);
  for i:=1 to nT div 2 do Write(f,B[i]:3); Writeln(f);
  Close(f);
end. {      }
```

## § 7.4. Отношение эквивалентности

Бинарное отношение $\sim$, определенное на множестве $S$, называется *отношением эквивалентности*, если оно удовлетворяет свойствам *рефлексивности*, *симметричности и транзитивности* (п. 1.6.8):

1. $\forall s_1 \in S \quad s_1 \sim s_1$;
2. $\forall s_1, s_2 \in S \quad s_1 \sim s_2 \to s_2 \sim s_1$;
3. $\forall s_1, s_2, s_3 \in S \quad s_1 \sim s_2 \wedge s_2 \sim s_3 \to s_1 \sim s_3$.

Элементы $s_i, s_j \in S$, для которых выполняется отношение $s_i \sim s_j$, называются *эквивалентными*.

Определение. *Классом эквивалентности*, порожденным элементом $s_i \in S$, называется подмножество $\widehat{S}_i = \{s \in S \mid s_i \sim s\}$ элементов, эквивалентных данному $s_i$.

Теорема 7.4.1. *Пусть на множестве $S$ задано отношение эквивалентности $\sim$. Для каждого элемента $s_i \in S$ составим соответствующий ему класс эквивалентности $\widehat{S}_i = \{s \in S \mid s_i \sim s\}$. Данные классы $\widehat{S}_i$ удовлетворяют следующим условиям*:

1) *для каждого $s_i \in S$ верно, что $s_i \in \widehat{S}_i$ принадлежит порожденному им классу эквивалентности*;
2) *классы, порожденные эквивалентными элементами $s_i \sim s_j$, совпадают: $\widehat{S}_i = \widehat{S}_j$*;
3) *классы эквивалентности совпадают ($\widehat{S}_i = \widehat{S}_j$), если их пересечение непусто: $\widehat{S}_i \cap \widehat{S}_j \neq \varnothing$.*

Доказательство. 1). Пусть класс $\widehat{S}_i = \{s \in S \mid s_i \sim s\}$ порожден элементом $s_i \in S$. Рефлексивность отношения эквивалентности $s_i \sim s_i$ позволяет заключить, что $s_i \in \widehat{S}_i$.

2). Пусть $z \in \widehat{S}_j = \{s \in S \,|\, s_j \sim s\}$ — произвольный элемент класса. Следовательно, $s_j \sim z$ . По условию $s_i \sim s_j$. Из транзитивности отношения имеем $s_i \sim s_j \wedge s_j \sim z \;\rightarrow s_i \sim z$, а значит, $z \in \widehat{S}_i$. Отсюда $\widehat{S}_j \subseteq \widehat{S}_i$. Подобным образом проверяется включение в обратную сторону $\widehat{S}_j \supseteq \widehat{S}_i$. Итак, классы эквивалентных элементов совпадают $\widehat{S}_j = \widehat{S}_i$.

3). Пусть $s \in \widehat{S}_i \cap \widehat{S}_j \neq \varnothing$ — непустое пересечение. Тогда верно, что $s_i \sim s$ и $s_j \sim s$. Из симметричности отношения имеем $s_j \sim s \rightarrow s \sim s_j$, а из транзитивности имеем $s_i \sim s \wedge s \sim s_j \rightarrow s_i \sim s_j$. Итак, $s_i$ и $s_j$ — эквивалентные. Классы, порожденные такими элементами, совпадают: $\widehat{S}_i = \widehat{S}_j$ (см. условие 2). □

Следствие. *Пусть на множестве $S$ задано отношение эквивалентности* ($\sim$). *Тогда множество $S$ разбивается на классы эквивалентности $S = \widehat{S}_{k_1} \cup \widehat{S}_{k_2} \cup \ldots \cup \widehat{S}_{k_m}$, где $\widehat{S}_{k_i} \cap \widehat{S}_{k_j} = \varnothing$, если $k_i \neq k_j$.*

Вследствие попарного непересечения $\widehat{S}_{k_i}$, разложение является прямым. Совокупность классов $\widehat{S}_{k_i}$, где $i = 1, 2, \ldots, m$, называется фактормножеством множества $S$ относительно отношения $\sim$ и обозначается как $S/\sim$.

Пример. Пусть $S$ — множество треугольников. Определим на $S$ бинарное ($\sim$) отношение. Будем считать, что для треугольников $\triangle a, \triangle b \in S$ выполняется отношение $\triangle a \sim \triangle b$, если они подобные. Данное отношение является отношением эквивалентности, так как свойства 1–3 выполняются для подобных треугольников. Введенное отношение разбивает множество треугольников на классы эквивалентности подобных треугольников.

Пример. Пусть $S$ — множество $n$-мерных векторов. Определим на $S$ бинарное ($\sim$) отношение. Будем полагать, что для $\overline{a}, \overline{b} \in S$ выполняется отношение $\overline{a} \sim \overline{b}$, если они колинеарные. Данное отношение является отношением эквивалентности, так как свойства 1–3 выполняются для колинеарных векторов. Множество векторов под действием введенного отношения разбивается на классы эквивалентности колинеарных векторов.

Пример. Пусть $S = \{1, 2, \ldots, n\}$. Определим на $S$ бинарное ($\sim$) отношение. Будем полагать, что для $p, q \in S$ выполняется отношение $p \sim q$, если они имеют одинаковые остатки от деления на целое положительное число $m$. Данное отношение является отношением эквивалентности. Множество $S$ под действием

введенного отношения разбивается на классы эквивалентности чисел с одинаковыми остатками от деления на $m$.

Пример. Примером отношения эквивалентности ($\sim$) может служить какая-либо классификация объектов множества $S$. Отношение $s_i \sim s_j$ для элементов $s_i, s_j \in S$ выполняется, если они имеют выделенные общие признаки. Объекты с общими признаками объединяются в отдельный класс эквивалентности.

Пример. Пусть $S$ — множество звезд. Определим на $S$ бинарное ($\sim$) отношение $s_i \sim s_j =$ «Звезды $s_i$ и $s_j$ принадлежат одному созвездию», где $s_i, s_j \in S$. Данное отношение является отношением эквивалентности. Произвольный класс эквивалентности такого отношения может состоять либо из звезд одного созвездия, либо из одной звезды, если такая звезда не входит ни в одно из созвездий.

## § 7.5. Связные компоненты

Пусть псевдограф $\Gamma(X, U, \Phi)$ является неориентированным. Две вершины $x_1, x_2 \in X$ называются *связанными*, если существует маршрут из $x_1$ в $x_2$. Определим на множестве вершин $X$ бинарное ($\sim$) отношение. Для $x_1, x_2 \in X$ отношение $\sim$ будет выполняться, т. е. $x_1 \sim x_2$, если эти вершины связанные. Введенное отношение является отношением эквивалентности. Действительно, если вершина $x_1$ связана с $x_2$, а вершина $x_2$ связана с $x_3$, то очевидно, что вершина $x_1$ связана с $x_3$. Следовательно, существует такое разложение множества вершин

$$X = \bigcup_i X_i$$

на попарно непересекающиеся подмножества, что все вершины в каждом $X_i$ связаны, а вершины из различных $X_i$ не связаны. Тогда можно записать разложение

$$\Gamma = \bigcup_i \widehat{\Gamma}(X_i, U_i, \Phi)$$

графа $\Gamma(X, U, \Phi)$ на непересекающиеся связные подграфы $\widehat{\Gamma}(X_i, U_i, \Phi)$. Вследствие попарного непересечения подграфов, разложение называется *прямым*, а сами подграфы называются *компонентами связности графа* $\Gamma$. Таким образом, справедливо следующее утверждение.

Утверждение 7.5.1. *Каждый неориентированный граф распадается единственным образом в прямую сумму своих компонент связности.*

Количество компонент связности находится в определенном отношении с основными параметрами графа — числом вершин и его ребер.

Утверждение 7.5.2. *Пусть* $\Gamma(X, U, \Phi)$ *является простым графом с* $n$ *вершинами и* $k$ *компонентами связности. Число ребер в таком графе не может превосходить величины* $C^2_{n-k+1} = (n-k+1)(n-k)/2$.

Доказательство. Рассмотрим прямое разложение $\Gamma = \bigcup_{i=1}^{k} \widehat{\Gamma}(X_i, U_i, \Phi)$ исходного графа на компоненты связности. Если положить, что число вершин в компоненте связности $X_i$ равно $n_i$, то число ребер в таком графе не превосходит $\sum_{i=1}^{k} C^2_{n_i}$. Данная величина достигается в том случае, когда каждая из компонент связности является полным подграфом. Допустим, что среди компонент связности $\widehat{\Gamma}(X_i, U_i, \Phi)$ найдутся хотя бы две, которые имеют более одной вершины, например, $n_2 \geqslant n_1 > 1$. Перенесем одну вершину из $\widehat{\Gamma}_1$ в $\widehat{\Gamma}_2$. Легко видеть, что это увеличивает число ребер в модифицируемом исходном графе с $k$ компонентами связности. Отсюда следует, что максимальное число ребер должен иметь граф, состоящий из $k-1$ изолированной вершины и одного полного подграфа с $n-k+1$ вершинами. □

Следствие. *Граф с* $n$ *вершинами и числом ребер, большим чем* $C^2_{n-1} = (n-1)(n-2)/2$, *связен.*

## § 7.6. Выделение компонент связности

Рассмотрим алгоритм нахождения числа компонент связности, а также выделения этих компонент на неориентированном графе. Подобным образом решается задача и для ориентированного графа. В основу рассматриваемого алгоритма 7.3 выделения компонент связности положена описанная ранее техника поиска в глубину на графе $\Gamma(X, U, \Phi)$. Структура алгоритма 7.3 является модификацией в сторону упрощения основного алгоритма 7.1 поиска в глубину. Работа алгоритма 7.3 направлена

на формирование вектора $Mark[x]$ меток вершин $x \in X$ графа. Элементу присваивается общий номер той компоненты, которой принадлежит вершина $x \in X$. Сложность алгоритма 7.3, как и алгоритма 7.1, составляет $O(|X| + |U|)$.

Алгоритм 7.3.

```
for v ∈ X do Mark[v] = 0; { }
count = 0; { }
for v ∈ X do if Mark[v] = 0 then begin
    count = count + 1;
    Component(v, count);
end;
Procedure Component(x, count);
    Mark[x] = count;
    for v ∈ Adj[x] do if Mark[v] = 0 then begin
        Component(v, count);
    end;
end.
```

Программная реализация выделения компонент связности представлена в алгоритме 7.4, который близко соотносится с соответствующим множественным описанием алгоритма 7.3. Рассмотрим пример расчета по программе алгоритма 7.4 выделения компонент связности графа, представленного на рис. 7.15.

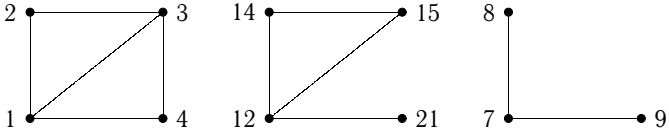



Рис. 7.15. Пример выделения компонент связности графа

Для программы алгоритма 7.4 исходные данные структуры смежности $Adj[x]$ графа на рис. 7.15 задаются в текстовом файле Connecnt.in. Структура (правило) заполнения файла одинакова с той, которая описана в рассмотренном примере поиска в глубину при расчете по программе алгоритма 7.2.

Данные файла Connect.in для примера на рис. 7.15:

```
11
1   3      2   3   4
2   2      3   1
```

```
9   1     7
8   1     7
12  3     14  15  21
14  2     12  15
3   3     2   1   4
4   2     1   3
7   2     9   8
15  2     14  12
21  1     12
```

Результаты расчетов сохраняются в выходном файле Connect.out со следующей структурой:

```
1  2  9  8  12 14 3  4  7  15 21  — номера вершин графа;
1  1  2  2  3  3  1  1  2  3  3   — номера компонент связности.
```

Алгоритм 7.4.

```
Program ConnectComponent; { }
uses CRT,DOS;
Const
  nVertex=100; { }
  nAdjacent=1000; {  }
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeAdjacent=array[1..nAdjacent] of Integer;
Var
  f  :Text; {}
  n  :Integer; {}
  Adj :TypeAdjacent; {  }
  Fst :TypeVertex; {  }
  Nbr :TypeVertex; {  }
  Vtx :TypeVertex; {  }
  Mark :TypeVertex; {  }

Procedure Init( Var yes :Boolean );
  { }
  {  }
  {  }
Var
  i,j,m :Integer;
begin
```

```
  for i:=1 to n do
    for j:=1 to Nbr[i] do begin
      yes:=FALSE;
      for m:=1 to n do
        if Adj[Fst[i]+j]=Vtx[m] then begin
          yes:=TRUE;
          Adj[Fst[i]+j]:=m;
          break;
        end;
      if not yes then exit;
    end;
end;

Procedure Component( x,count :Integer);
  {   }
Var
  i,v :Integer;
begin
  Mark[x]:=count;
  for i:=1 to Nbr[x] do begin
    v:=Adj[Fst[x]+i];
    if Mark[v]=0 then Component(v,count);
  end
end;

Procedure Connect; { }
Var
  v,count :Integer;
begin
  for v:=1 to n do Mark[v]:=0;
  count:=0; { }
  for v:=1 to n do begin
    if Mark[v]=0 then begin
      count:=count+1;
      Component(v,count);
    end;
  end;
end;
```

```
Var {       }
  i,j :Integer;
  yes :Boolean;
begin {       }
  Assign(f,'Connect.in');
  Reset(f); {   }
  {   }
  Read(f,n); {   }
  Fst[1]:=0; {    }
  for i:=1 to n do begin
    Read(f,Vtx[i]); { }
    Read(f,Nbr[i]); {   }
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]); {
                                                          }
    Fst[i+1]:=Fst[i]+Nbr[i]; {
                                                           }
  end;
  Close(f);
  Assign(f,'Connect.out');
  Rewrite(f); {   }
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа!');
    Close(f);
    exit;
  end;
  Connect;
  for i:=1 to n do Write(f,Vtx[i]:3); Writeln(f);
  for i:=1 to n do Write(f,Mark[i]:3);
  Close(f);
end. {       }
```

## § 7.7. Эйлеровы графы

Классической в теории графов является следующая задача. Город стоит на реке. Имеются два острова $A$ и $D$, соединенных семью мостами с берегами реки $C$ и $B$ и друг с другом, как показано на рис. 7.16. Задача состоит в следующем: осуществить

прогулку по городу таким образом, чтобы, пройдя по каждому мосту один раз, вернуться обратно. Решение этой задачи сводится к нахождению некоторого специального маршрута на графе.

О п р е д е л е н и е. Пусть $\Gamma(X, U, \Phi)$ — неориентированный псевдограф. *Цепь* в $\Gamma$ называется *эйлеровой*, если она проходит по одному разу через каждое ребро псевдографа $\Gamma$. Такой граф называется эйлеровым. Замкнутая эйлерова цепь называется *эйлеровым циклом*.

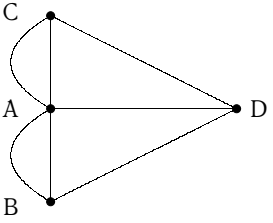


Рис. 7.16. План расположения мостов и соответствующий ему мультиграф

Поставим в соответствие плану расположения суши и мостов на рис. 7.16 мультиграф, в котором каждой части суши соответствует вершина, а каждому мосту — ребро, соединяющее соответствующие вершины. Теперь задача звучит так: найти эйлерову цепь (цикл) в мультиграфе. Решение этой задачи было дано Л. Эйлером.

Т е о р е м а 7.7.1 (Л. Э й л е р а). *Эйлерова цепь в* $\Gamma(X, U, \Phi)$ — *псевдографе без изолированных вершин существует тогда и только тогда, когда выполняются следующие условия*:

1) *граф связный*;
2) *степени внутренних вершин четные* (*внутренние вершины не являются началом и концом цепи*);
3) *если вершины* $a$ *и* $b$ *являются началом и концом цепи и* $a \neq b$, *то степени их нечетные*;
4) *если вершины* $a$ *и* $b$ *являются началом и концом цепи и* $a = b$, *то степени их четные*.

Д о к а з а т е л ь с т в о. ($\Rightarrow$) Дано, что существует эйлерова цепь $au_1x_2u_2 \ldots x_nu_nb$, где $a, b, x_i \in X$, $u_i \in U$, в которой содержатся все ребра по одному разу. Такая цепь включает все вершины графа, так как граф не содержит изолированных вершин.

Докажем условия 1)–4): 1) по данной цепи из любой вершины можно попасть в любую другую, значит, граф связный; 2) каждая тройка цепи $u_{i-1}x_iu_i$ привносит вершине $x_i$ степень два, а так как все ребра $u_i$ в цепи различные, то степени внутренних вершин четные; доказательства 3) и 4) повторяют доказательство 2).

($\Leftarrow$) Даны условия 1)–4). Построим эйлерову цепь. Предварительно приведем условие 3) к условию 4) включением в граф

фиктивного ребра $u^*$, которым свяжем вершины $a$ и $b$. Теперь и в случае 3) все вершины будут иметь четную степень. Пусть $A \in X$ — произвольная вершина (рис. 7.17). Из нее будем строить цепь, выбирая в качестве продолжения пути ребро, которое еще не пройдено. Эта цепь (цикл $\Gamma_1$) может закончиться только в вершине $A$, так как, войдя в любую другую вершину, всегда существует ребро, по которому можно выйти из нее (степени вершин четные).

Возможны два случая: 1) построенный цикл $\Gamma_1$ содержит все ребра графа, тогда теорема доказана; 2) $\Gamma_1$ содержит не все ребра графа. Во втором случае рассмотрим граф $\Gamma \setminus \Gamma_1$, полученный удалением из $\Gamma$ всех ребер, входящих в $\Gamma_1$. Граф $\Gamma \setminus \Gamma_1$ вновь содержит вершины только с четными степенями (у каждой вершины удалили по четному числу ребер). Так как $\Gamma$ — связный граф, то существует вершина в $\Gamma_1$, инцидентная ребру из $\Gamma \setminus \Gamma_1$. Пусть это вершина $B \in X$. Построим из нее цикл $\Gamma_2$ так же, как строили цикл $\Gamma_1$. Построим общий цикл $\Gamma_{12}$ из $\Gamma_1$ и $\Gamma_2$ так, как это сделано на рис. 7.17. Для $\Gamma_{12}$ вновь проверяем рассмотренные выше два случая, как для $\Gamma_1$.

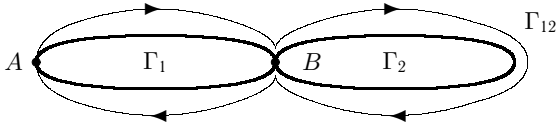



Рис. 7.17. Объединение двух циклов в один цикл

Процесс расширения продолжаем до тех пор, пока не будут включены все ребра графа в один цикл: $Au_1x_2 \dots au^*b \dots u_nA$. Разрывая данный цикл по ребру $u^*$, получим эйлерову цепь $aw_1t_2 \dots t_nw_nb$, где $t_i \in X, w_i \in U$. □

Конструктивный характер доказательства теоремы позволяет на формальном уровне в алгоритме 7.5 записать рассмотренный поиск эйлеровой цепи. Исходный граф представлен своей структурой смежности $Adj[x]$, где $Adj[x]$ — множество вершин, смежных с $x \in X$. Результирующая эйлерова цепь формируется в множестве $Z$.

Алгоритм 7.5.

$\exists v \in X$; { }
$Z = \{v\}$; { }
$R = \varnothing$; { }

```
repeat
  Cycle(v, R);
  Z = Z ∪ R; {   }
until Not ∃v ∈ Z  |Adj[v]| > 0
Procedure Cycle(v, R)
  R = {v}; {   }
  repeat
    w = Adj[v];
    R = R ∪ {w};
    Adj[v] = Adj[v] \ {w}; {  (v, w)}
    if v ≠ w then Adj[w] = Adj[w] \ {v}; {  (w, v)}
  until Not |Adj[v]| > 0 {   }
end.
```

Частные же циклы, расширяющие $Z$, представляются множеством $R$. Ребра, включенные в частный цикл $R$ (а значит, и в $Z$), удаляются как пройденные из структуры смежности $Adj[x]$ (из графа). Формирование расширяющих циклов $R$ осуществляется до тех пор, пока структура смежности графа содержит хотя бы одно ребро.

• *Сложность алгоритма поиска эйлеровой цепи.* Число обращений к процедуре *Cycle* выполняется не более, чем число вершин $|X|$. При каждом обращении количество производимых действий пропорционально числу ребер, входящих в выделенный цикл. Сложность суммарной работы процедуры *Cycle* пропорциональна количеству ребер $U$ в графе. Поэтому сложность выделения эйлеровой цепи составляет $O(|X| + |U|)$.

Программная реализация поиска эйлеровой цепи представлена в алгоритме 6.6, который соответствует множественному описанию соответствующего алгоритма 6.5.

Алгоритм 7.6.

```
Program EilerWay; {   }
uses CRT,DOS;
Const
  nVertex=100; { }
  nAdjacent=1000; {   }
Type
  TypeVertex=array[1..nVertex] of Integer;
```

```
  Type Adjacent=array[1..n Adjacent] of Integer;
Var
  f  :Text; { }
  ks :Integer; {  }
  n  :Integer; { }
  Adj :Type Adjacent; {  }
  Fst :TypeVertex; {  }
  Nbr :TypeVertex; {  }
  Vtx :TypeVertex; {  }
  Deg :TypeVertex; {  }
  kz :Integer; {  }
  z  :Type Adjacent; {  }
  r  :Type Adjacent; {  }

Procedure Init( Var yes :Boolean );
Var
  i,j,k,m :Integer;
begin
  {  }
  {    }
  {          }
  for i:=1 to n do
    for j:=1 to Nbr[i] do begin
      yes:=FALSE;
      for m:=1 to n do
        if Adj[Fst[i]+j]=Vtx[m] then begin
          yes:=TRUE;
          Adj[Fst[i]+j]:=m;
          break;
        end;
      if not yes then exit;
    end;
  {    }
  for i:=1 to n do begin
    k:=1;
    for j:=1 to Nbr[i] do if Adj[Fst[i]+j]=i then begin
      Adj[Fst[i]+j]:=Adj[Fst[i]+k];
      Adj[Fst[i]+k]:=i;
      k:=k+1;
```

```
      end;
   end;
   { }
   for i:=1 to n do begin
      Deg[i]:=0;
      for j:=1 to Nbr[i] do begin
         Deg[i]:=Deg[i]+1;
         if Adj[Fst[i]+j]=i then Deg[i]:=Deg[i]+1; {}
      end;
   end;
   {      }
   k:=0; ks:=1;
   for i:=1 to n do if ( Deg[i] mod 2 ) > 0 then begin
      k:=k+1; ks:=i;
   end;
   if ( k<>2 ) and ( k<>0 ) then yes:=FALSE; { }
end;

Procedure Cycle( v :Integer; var count :Integer );
{      }
Var
   w  :Integer;
   i,j :Integer;
begin
   count:=1; r[count]:=v;
   repeat
      w:=Adj[Fst[v]+1]; { }
      count:=count+1; r[count]:=w;
      { (v, w)      v}
      Fst[v]:=Fst[v]+1;
      Nbr[v]:=Nbr[v]-1;
      { (w, v)  }
      {       w}
      if v<>w then
         for i:=1 to Nbr[w] do if Adj[Fst[w]+i]=v then begin
            for j:=i+1 to Nbr[w] do Adj[Fst[w]+j-1]:=Adj[Fst[w]+j];
            Nbr[w]:=Nbr[w]-1;
            break;
         end;
```

```
    v:=w;
  until Not( Nbr[v]>0 );
end;

Procedure Eiler; {      }
Var
  v,w :Integer;
  i,j,kt :Integer;
  count :Integer;
  yes :Boolean;
begin
  v:=ks; kz:=1;
  kt:=kz; z[kz]:=v;
  Write(f,'Z='); { }
  for i:=1 to kz do Write(f,Vtx[z[i]]:3); WriteLn(f);
  repeat
    Cycle(v,count);
    Write(f,'R=');
    for i:=1 to count do Write(f,Vtx[r[i]]:3); WriteLn(f);
    for i:=1 to count-1 do begin
      z[kz+i]:=z[kt+i];
      z[kt+i]:=r[i+1];
    end;
    kz:=kz+count-1;
    Write(f,'Z='); { }
    for i:=1 to kz do Write(f,Vtx[z[i]]:3); WriteLn(f);
    yes:=FALSE;
    for i:=kz downto 1 do if Nbr[z[i]]>0 then begin
      v:=z[i];
      kt:=i;
      yes:=TRUE;
      break;
    end;
  until Not yes;
end;

Var {      }
  i,j :Integer;
  yes :Boolean;
```

```
begin {       }
  Assign(f,'Eiler.in');
  Reset(f); {   }
  { }
  Read(f,n); {   }
  Fst[1]:=0; {    }
  for i:=1 to n do begin
    Read(f,Vtx[i]); { }
    Read(f,Nbr[i]); {   }
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]);
        { }
    Fst[i+1]:=Fst[i]+Nbr[i];
        {      }
  end;
  Close(f);
  Assign(f,'Eiler.out');
  Rewrite(f); {   }
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа');
    WriteLn(f,' или граф не эйлеровый!');
    Close(f);
    exit;
  end;
  Eiler;
  Write(f,'O=');
  for i:=1 to kz do Write(f,Vtx[z[i]]:3); WriteLn(f);
  Close(f);
end. Main
```

Рассмотрим пример расчета по программе алгоритма 7.6 поиска эйлеровой цепи в графе, изображенном на рис. 7.18.

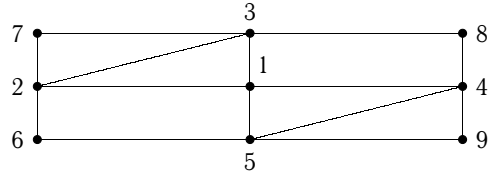



Рис. 7.18. Пример расчета эйлеровой цепи графа

Для программы алгоритма 7.6 исходные данные структуры смежности $Adj[x]$ графа на рис. 7.18 задаются в текстовом файле Eiler.in. Структура (правило) заполнения файла совпадает с той, которая описана в рассмотренном примере поиска в глубину при расчете по программе алгоритма 7.2.

Данные файла Eiler.in для примера на рис. 7.18:

```
9
1  4     2  3  4  5
2  4     1  3  6  7
3  4     1  2  7  8
4  4     1  5  8  9
5  4     1  4  6  9
6  2     2  5
7  2     2  3
8  2     3  4
9  2     4  5
```

Результаты расчетов сохраняются в выходном файле Eiler.out со следующей структурой:

```
Z=  1
R=  1  2  3  1  4  5  1
Z=  1  2  3  1  4  5  1
R=  5  6  2  7  3  8  4  9  5
Z=  1  2  3  1  4  5  6  2  7  3  8  4  9  5  1
O=  1  2  3  1  4  5  6  2  7  3  8  4  9  5  1
```

На каждом шаге показана строящаяся эйлерова цепь $Z$, которая расширяется выделенным $R$ циклом. Результирующая эйлерова цепь графа отмечена признаком $O$ в начале строки.

## § 7.8. Остовные деревья

Определение. *Остовным деревом* связного неориентированного графа $\Gamma(X, U, \Phi)$ называется дерево $\Gamma_0(X, U_0, \Phi)$, являющееся подграфом графа $\Gamma$ и содержащее все его вершины (рис. 7.19).

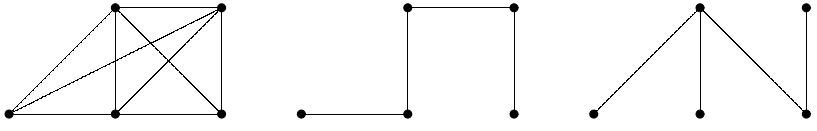


Рис. 7.19. Связный граф и два остовных дерева

Утверждение 7.8.1. *В дереве с $n$ вершинами имеется $n-1$ ребро.*

Доказательство проведем по индукции числа вершин. Заметим, что в дереве найдется вершина, степень которой равна единице (висячая вершина). Действительно, в противном случае, если все степени вершин $\geqslant 2$, можно построить цикл, что противоречило бы определению дерева. Цикл строится следующим образом. Выполним проход по ребрам графа, начиная с произвольной вершины. Так как степени $\geqslant 2$, то, попав в вершину первый раз, можно всегда из нее выйти. Исходный граф — конечный и связный. Следовательно, наступит момент, когда вновь попадем в пройденную уже вершину, что доказывает существование цикла.

Условие индукции для $n = 2$ выполняется, дерево с двумя вершинами содержит одно ребро. Предположим теперь, что утверждение выполняется для деревьев, число вершин у которых меньше $n$. Рассмотрим дерево с $n$ вершинами. Удалим из этого дерева висячую вершину и инцидентное ей ребро. Очевидно, что оставшийся граф будет связным и без циклов, т. е. будет деревом с $n-1$ вершиной. Тогда по предположению индукции оставшаяся часть графа содержит $n-2$ ребра, а значит, исходное дерево должно иметь их $n-1$. □

Практическую значимость остовных деревьев дает популярная форма задачи Кэли. Необходимо соединить $n$ городов железнодорожными линиями так, чтобы не строить лишних дорог. Известна стоимость строительства дорог для каждой пары городов. Какова должна быть сеть дорог, соединяющая все города и имеющая минимальную возможную стоимость? Аналогичные вопросы возникают при проектировании линий электропередач, сетей ЭВМ и др.

В терминах теории графов задачу можно сформулировать следующим образом. Рассмотрим граф $\Gamma(X, U, \Phi)$, где $X$ — города, $U$ — дороги. Каждому ребру $u \in U$ назначим вес $\omega(u)$ — стоимость строительства дороги $u$. Задача состоит в том, чтобы построить связный граф $\Gamma_0(X, U_0, \Phi)$, содержащий все вершины, с минимальным весом $W(\Gamma_0) = \sum\limits_{u \in U_0} \omega(u)$. Очевидно, что $\Gamma_0$ — дерево, в противном случае можно было бы удалить одно ребро, не нарушая связности $\Gamma_0$ и уменьшая сумму весов его ребер.

Определение. *Минимальным* остовным деревом (лесом) называется остовное дерево (лес) с минимальным общим весом его ребер.

### 7.8.1. Жадный алгоритм построения минимального остовного дерева

Минимум остовных деревьев графа $\Gamma(X, U, \Phi)$ можно найти, применяя процедуру исследования ребер в порядке возрастания их весов. Другими словами, на каждом шаге выбирается новое ребро с наименьшим весом, не образующее циклов с уже выбранными ребрами. Процесс продолжается до тех пор, пока не будет выбрано $|X| - 1$ ребро. Рассмотренная процедура называется *жадным алгоритмом*.

Реализация данной схемы может быть выполнена следующим образом. Для каждой вершины $X = \{x_1, x_2, \ldots, x_n\}$ графа $\Gamma(X, U, \Phi)$ формируются начальные тривиальные компоненты связности $T_i(X_i, U_i, \Phi)$, где $X = \{x_i\}$, $U_i = \varnothing$, $X_i \cap X_j = \varnothing$, $i \neq j$, $X = \bigcup\limits_{i=1}^{|X|} X_i$, $i, j = 1, 2, \ldots, |X|$. Компоненты $T_i$ являются деревьями, объединение $T = \bigcup\limits_i T_i$ которых дает начальное приближение строящегося остовного дерева $\Gamma_0(X, U_0, \Phi)$.

Включение в строящееся остовное дерево $\Gamma_0$ выбранного ребра на очередном шаге жадного алгоритма выполняется слиянием $T_i = T_i \cup T_j$ ($X_i = X_i \cup X_j$ и $U_i = U_i \cup U_j$) двух компонент $T_i$ и $T_j$, которым принадлежит по вершине нового ребра, и включением самого ребра в объединенное множество $U_i = U_i \cup U_j$ ребер. Процесс роста объединения $T = \bigcup\limits_i T_i$ компонент к остовному дереву $\Gamma_0(X, U_0, \Phi)$ продолжаем до тех пор, пока не будет включено $|X| - 1$ ребро. Справедливость жадного алгоритма является следствием следующих двух лемм.

Лемма 7.8.1. *Пусть $\Gamma(X, U, \Phi)$ — связный неориентированный граф и $\Gamma_0(X, U_0, \Phi)$ — произвольное остовное дерево для него. Тогда*:
1) *$\forall x_1, x_2 \in X$ существует единственная между ними цепь в $\Gamma_0$;*
2) *если к $\Gamma_0$ добавить ребро из $U \setminus U_0$, то возникнет ровно один цикл.*

Доказательство. Утверждение 1 верно, так как в противном случае в $\Gamma_0$ существовал бы цикл, что противоречит дереву $\Gamma_0$. Утверждение 2 верно, поскольку между вершинами добавляемого ребра уже есть одна цепь, а значит, возникнет один цикл. □

Лемма 7.8.2. *Пусть* $\Gamma(X, U, \Phi)$ *— связный неориентированный граф, для каждого ребра* $u \in U$ *определен вес* $\omega(u)$, *и* $T_i(X_i, U_i, \Phi)$ *— компоненты связности жадного алгоритма, объединение которых* $T = \bigcup_i T_i$, *согласно алгоритму, растет к остовному дереву* $\Gamma_0(X, U_0, \Phi)$, *где* $i = 1, 2, \ldots, k$ *и* $k > 1$. *Пусть следующим найденным для включения ребром является ребро* $\varepsilon = (x, y)$ *наименьшего веса из оставшихся* $U \setminus \bigcup_i U_i$ *и пусть* $x \in X_1$ *и* $y \notin X_1$. *Тогда найдется остовное дерево* $\Gamma_0$ *для* $\Gamma(X, U, \Phi)$, *содержащее ребра* $\bigcup_i U_i \cup \{\varepsilon\}$, *вес которого не больше любого другого остовного дерева, содержащего ребра* $\bigcup_i U_i$.

Доказательство. Допустим противное. Пусть существует остовное дерево $\Gamma_0(X, U_0', \Phi)$ для $\Gamma$, содержащее $\bigcup_i U_i$ и не содержащее ребра $\varepsilon$, вес которого меньше веса любого остовного дерева для $\Gamma$, содержащего $\bigcup_i U_i \cup \{\varepsilon\}$. По утверждению 1 леммы 7.8.1, при добавлении $\varepsilon$ к $\Gamma_0'$ образуется цикл. Этот цикл должен содержать такое ребро $\varepsilon' = (x', y')$, отличное от $\varepsilon$, что $x' \in X_1$ и $y' \notin X_1$, так как цикл входит в $U_1$ по ребру $(x, y)$ и $y \notin X$, то он должен и выйти из $U_1$ (рис. 7.20). Из условия следует, что вес $\omega(\varepsilon) \leqslant \omega(\varepsilon')$, так как $\bigcup_i U_i \subseteq U_0$ и $\bigcup_i U_i \subseteq U_0'$ и ребро $\varepsilon$ выбиралось с минимальным весом из оставшихся ребер $U \setminus \bigcup_i U_i$ без образования циклов, в противном же случае выбор должен был бы пасть на $\varepsilon'$, так как оно тоже не образует циклов с $\bigcup_i U_i$.

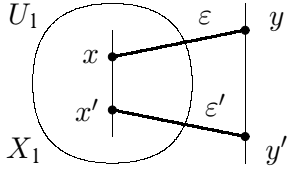



Рис. 7.20

Рассмотрим граф $\Gamma_0(X, U_0, \Phi)$, образованный добавлением $\varepsilon$ к $\Gamma_0'$ и удалением $\varepsilon'$ из $\Gamma_0$. В $\Gamma_0$ нет циклов, так как единственный цикл разорван удалением ребра $\varepsilon'$. $\Gamma_0$ остался связным, так как осталась цепь между $x'$ и $y'$. Таким образом, $\Gamma_0$ — остовное дерево. Учитывая, что $\omega(\varepsilon) \leqslant \omega(\varepsilon')$, то вес дерева $\Gamma_0$ не больше веса $\Gamma_0'$. Остовное дерево $\Gamma_0$ содержит $\bigcup_i U_i$ и $\varepsilon$, а это противоречит предположению минимальности $\Gamma_0'$, что доказывает справедливость *жадного алгоритма*. □

Реализация «жадной» схемы формирования остовного дерева представлена в алгоритме 7.7. Используемые при его описании обозначения соответствуют тому, что вводилось при обосновании жадной схемы. Вектор $\mathrm{Mark}[x]$ меток вершин $x \in X$ графа поддерживает их принадлежность компонентам связности $U_i$, из которых формируется реберный список остовного дерева. Начальные величины $\mathrm{Mark}[x_i]$ устанавливаются равными порядковым номерам соответствующих вершин $x_i \in X$. Далее значения $\mathrm{Mark}[x_i]$ корректируются по мере слияния компонент $U_i$, сохраняя соответствие принадлежности им вершин. Исходный граф задается реберным списком $U$, выходное остовное дерево также формируется реберным списком $U_0$.

Алгоритм 7.7.

```
for x_i ∈ X do Mark[x_i] = i;
Sort(U); {     }
U_0 = ∅;
while |U_0| < |X| − 1 do begin
    (x, y) ∈ U; {          }
    if Mark[x] ≠ Mark[y] then begin
        U_0 = U_0 ∪ {(x, y)}; {    }
        z = Mark[y]; { U_x  U_y}
        for v ∈ X do if Mark[v] = z then
            Mark[v] = Mark[x];
    end;
    U = U \ {(x, y)}; {          }
end.
```

• *Сложность жадного алгоритма*. Жадный алгоритм требует предварительной сортировки ребер по их весам. Стандартные методы сортировки имеют сложность $O(|U|^2)$. Сложность алгоритма 7.7 помимо сложности сортировки зависит от сложности реализации слияния подмножеств $U_x$ и $U_y$. Сложность данной операции при полном переборе вершин данных подмножеств (как представлено в алгоритме 7.7) составляет $O(|X|^2)$. Значит, сложность жадного алгоритма $O(|X|^2 + |U|^2)$.

Программная реализация жадного алгоритма представлена в алгоритме 7.8, который близко соответствует множественному описанию соответствующего алгоритма 7.7.

Алгоритм 7.8.

```
Program HungryOstov; {   }
uses CRT,DOS;
Const
  nVertex=50; { }
  nRib=1000; { }
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeRib=array[1..nRib] of Integer;
Var
  f  :Text; {}
  nX :Integer; {   }
  nU :Integer; {   }
  Mark :TypeVertex; { }
  X  :TypeVertex; {  }
  U  :TypeRib; {  }
  nUo :Integer; {    }
  Uo :TypeRib; {  }
  We :TypeRib; {  }
  Wt :LongInt; {   }

Procedure Init; { }
Var
  i,j,m :Integer;
begin
  for i:=1 to 2*nU do Uo[i]:=1;
  for i:=1 to 2*nU do
    for j:=i+1 to 2*nU do if Uo[j]=1 then
      if U[j]=U[i] then Uo[j]:=0;
  nX:=0;
  for i:=1 to 2*nU do
    if Uo[i]=1 then begin
      nX:=nX+1;
      X[nX]:=U[i];
    end;
  for i:=1 to 2*nU do Новые метки
```

```
    for m:=1 to nX do
      if U[i]=X[m] then begin U[i]:=m; break; end;
end;

Procedure Sort; {     }
Var
  i,j,k :Integer;
  w  :Integer;
begin
  for i:=1 to nU do
    for j:=1 to nU-i do
      if We[j]>We[j+1] then begin
        w:=We[j]; We[j]:=We[j+1]; We[j+1]:=w;
        w:=U[2*j-1]; U[2*j-1]:=U[2*(j+1)-1]; U[2*(j+1)-1]:=w;
        w:=U[2*j]; U[2*j]:=U[2*(j+1)]; U[2*(j+1)]:=w;
      end;
end;

Procedure Ostov; {   }
Var
  i,x,y,z :Integer;
  sU :Integer;
begin
  for i:=1 to nX do Mark[i]:=i;
  Sort; {   }
  nUo:=0; {  U0}
  sU:=1; {    U}
  while nUo<nX-1 do begin
    x:=U[2*sU]; {    }
    y:=U[2*sU-1];
    if Mark[x]<>Mark[y] then begin
      nUo:=nUo+1;
      Uo[nUo]:=sU; {    }
      z:=Mark[y]; { Ux  Uy}
      for i:=1 to nX do if Mark[i]=z then Mark[i]:=Mark[x];
    end;
    sU:=sU+1; {  (x, y)   U}
```

```
  end;
end;

Var {      }
  i,j :Integer;
begin {      }
  Assign(f,'Hungry.in');
  Reset(f);Файл открыт для чтения
  Read(f,nU); {    }
  for i:=1 to nU do Read(f,U[2*i-1]); { }
  for i:=1 to nU do Read(f,U[2*i]); { }
  for i:=1 to nU do Read(f,We[i]); { }
  Close(f);
  Assign(f,'Hungry.out');
  Rewrite(f); {   }
  Init;
  Sort;
  WriteLn(f,'nU =',nU:3);
  WriteLn(f,'nX =',nX:3);
  Write(f,'X =');for i:=1 to nX do Write(f,X[i]:3);WriteLn(f);
  Write(f,'u1 =');for i:=1 to nU do Write(f,X[U[2*i-1]]:3);WriteLn(f);
  Write(f,'u2 =');for i:=1 to nU do Write(f,X[U[2*i]]:3);WriteLn(f);
  Write(f,'We =');for i:=1 to nU do Write(f,We[i]:3);WriteLn(f);
  Ostov;
  Write(f,'uo1=');for i:=1 to nUo
                  do Write(f,X[U[2*Uo[i]-1]]:3);WriteLn(f);
  Write(f,'uo2=');for i:=1 to nUo
                  do Write(f,X[U[2*Uo[i]]]:3);WriteLn(f);
  Write(f,'Woe=');for i:=1 to nUo do Write(f,We[Uo[i]]:3);WriteLn(f);
  Wt:=0;
  for i:=1 to nUo do Wt:=Wt+We[Uo[i]];
  Write(f,'Bec=',Wt:3);
  Close(f);
end. {      }
```

Рассмотрим пример построения минимального остовного дерева графа, изображенного на рис. 7.21, по программе алгоритма 7.8.

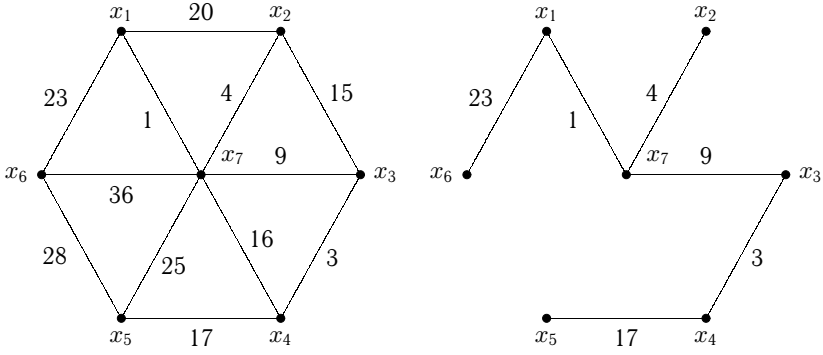



Рис. 7.21. Пример расчета остовного дерева

Для программы этого алгоритма исходные данные графа на рис. 7.21 задаются реберным списком в текстовом файле Hungry.in со следующей структурой:

- в первой строке файла содержится количество ребер в списке (12);
- во второй и третьей строках указываются ребра своими вершинами: одна вершина во второй строке, другая вершина ребра в третьей строке;
- в четвертой строке располагаются значения весов соответствующих ребер.

```
12
7  7  7  7  7  7  1  2  3  4  5  6
1  2  3  4  5  6  2  3  4  5  6  1
1  4  9  16 25 36 20 15 3  17 28 23
```

Результаты расчетов сохраняются в выходном файле Hungry.out со следующей структурой:

```
nU  =  12
nX  =  7
X   =  7   1   2   3   4   5   6
u1  =  7   3   7   7   2   7   4   1   6   7   5   7
u2  =  1   4   2   3   3   4   5   2   1   5   6   6
We  =  1   3   4   9   15  16  17  20  23  25  28  36
uo1 =  7   3   7   7   4   6
uo2 =  1   4   2   3   5   1
Woe =  1   3   4   9   17  23
Вес =  57
```

Обозначения данных в файле Hungry.out:

| | |
|---|---|
| nU | — число ребер в графе; |
| nX | — число вершин в графе; |
| X | — список вершин графа; |
| (u1, u2) | — сортированный список ребра графа; |
| We | — веса ребер согласно их сортировке; |

(uo1, uo2) — ребра остовного дерева;
Woe — веса ребер остовного дерева;
Вес — сумма весов ребер остовного дерева.

### 7.8.2. Алгоритм ближайшего соседа построения остовного дерева

Данный метод построения *минимального остовного дерева* не требует ни сортировки, ни проверки на цикличность на каждом шаге.

1. Построение остовного дерева $\Gamma_0$ начинается с произвольной вершины $x_1$.
2. Затем среди ребер, инцидентных $x_1$, выбираем ребро $(x_1, x_2)$ с наименьшим весом и включаем его в дерево $\Gamma_0$.
3. Повторяя процесс, выполняем поиск наименьшего по весу ребра, соединяющего вершины $x_1$ или $x_2$ с некоторой другой вершиной графа $x_3$.
4. Процесс включения ребер продолжаем до тех пор, пока все вершины исходного графа $\Gamma$ не будут включены в дерево $\Gamma_0$. Построенное дерево будет минимальным остовным.

Доказательство того, что последовательность шагов 1–4 приводит к построению минимального остовного дерева, аналогично доказательству для жадного алгоритма. Реализация схемы ближайшего соседа формирования остовного дерева выполнена в алгоритме 7.9, где исходный граф $\Gamma(X, U, \Phi)$ представляется матрицей весов $We = [w_{ij}]$, веса несуществующих ребер полагаются равными $+\infty$. Под весами ребер понимаются их длины. Остовное дерево $\Gamma_0(X_0, U_0, \Phi)$ формируется посредством реберного списка $U_0$ и списка вершин $X_0$. В качестве меток вершин устанавливаются их порядковые номера $X = \{1, 2, \ldots, n\}$. Для каждой вершины $x \in X$ графа, еще не включенной в остовное дерево, поддерживается минимальное расстояние до множества ранее включенных вершин в $X_0$. Это осуществляется с помощью двух векторов $dist[x]$ и $prev[x]$, где $dist[x]$ равно минимальному расстоянию от $x \in X$ до вершины $prev[x] \in X_0$. Обновление значений векторов $dist[x]$ и $prev[x]$ выполняется на каждом шаге алгоритма при пополнении $X_0$ новой вершиной.

Алгоритм 7.9.

$X = \{1, 2, \ldots, n\}$; { }
$nX = |X|$; { }
$v = rand(1, |X|)$; { $v \in X$}

$X_0 = \{v\}$; { }
$X = X \setminus \{v\}$; { $v$ $X$}
$U_0 = \varnothing$; { }
$W_T = 0$; { }
for $x \in X$ do begin
$dist[x] = We[x, v]$; { $x$ $v \in X_0$}
$prev[x] = v$; { $(v, x)$ $dist[x]$}
end;
while $|X_0| \neq nX$ do begin
$dist[v] = \min\limits_{x \in X} dist[x]$; {$v \in X$ }
$X_0 = X_0 \cup \{v\}$; { }
$X = X \setminus \{v\}$; { }
$U_0 = U_0 \cup \{(prev[v], v)\}$; { }
$W_T = W_T + dist[v]$; { }
{ $dist[x]$ }
for $x \in X$ do if $dist[x] > We[v, x]$ then begin
$dist[x] = We[v, x]$;
$prev[x] = v$;
end;
end.

• *Сложность алгоритма ближайшего соседа*. Сложность алгоритма определяется двумя вложенными циклами по числу вершин. В каждом из циклов выполняется константное число операций. Следовательно, сложность составляет $O(|X|^2)$.

Программная реализация алгоритма ближайшего соседа представлена в алгоритме 7.10, который близко соответствует множественному описанию соответствующего алгоритма 7.9.

Алгоритм 7.10.

```
Program NearOstov; {
                                                    }
uses CRT,DOS;
Const
  nVertex=50; { }
  nRib=1000; { }
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeRib=array[1..nRib] of Integer;
```

```
  TypeWeight=array[1..nVertex,1..nVertex] of Integer;
Var
  f  :Text; { }
  nX :Integer; {  }
  nXo :Integer; {  }
  nUo :Integer; {  }
  X  :TypeVertex; {  }
  Xo :TypeVertex; {  }
  Uo :TypeRib; {  }
  Prev :TypeVertex; {  }
  Dist :TypeRib; {  }
  We :TypeWeight; {  }
  Wt :LongInt; {  }
Procedure minDist( Var v :Integer );
{ v   }
Var i,d :Integer;
begin
  v:=X[1];
  d:=Dist[v];
  for i:=2 to nX do
    if d>Dist[X[i]] then begin
      v:=X[i];
      d:=Dist[v];
    end;
end;

Procedure newDist( v :Integer );
{ dist[]   }
Var i :Integer;
begin
  for i:=1 to nX do
    if Dist[X[i]]>We[X[i],v] then begin
      Dist[X[i]]:=We[X[i],v];
      Prev[X[i]]:=v;
    end;
end;

Procedure Ostov; { }
Var
```

```
  i,nStop,v:Integer;
begin
  nStop:=nX;
  for i:=1 to nX do X[i]:=i;
  v:=1;
  for i:=1 to nX do begin
    Dist[i]:=We[i,v];
    Prev[i]:=v;
  end;
  nXo:=0; {          }
  nUo:=0; {          }

  nXo:=nXo+1;
  Xo[nXo]:=X[v]; {                          }
  X[v]:=X[nX]; nX:=nX-1; {X = X \ {v}     v   X}


  Wt:=0;
  while nXo<>nStop do begin
    minDist(v);
    nXo:=nXo+1;
    Xo[nXo]:=v; {X0 = X0 ∪ {v}       v  }
    for i:=1 to nX do {X = X \ {v}     v   X}
      if X[i]=v then begin X[i]:=X[nX]; break; end;
    nX:=nX-1;
    nUo:=nUo+1; {U0 = U0 ∪ {(prev[v], v)}
                                                    }
    Uo[2*nUo-1]:=Prev[v];
    Uo[2*nUo]:=v;
    Wt:=Wt+Dist[v]; {   }
    newDist(v);
  end;
end;


Var {       }
  i,j :Integer;
begin {       }
  Assign(f,'Near.in');
  Reset(f); {   }
```

```
  Read(f,nX); {  }
  for i:=1 to nX do begin
    for j:=i to nX do begin
      Read(f,We[i,j]); {    }
      if We[i,j]=0 then We[i,j]:=$7fff; {  }
      We[j,i]:=We[i,j];
    end;
  end;
  Close(f);
  Assign(f,'Near.out');
  Rewrite(f); {  }
  nUo:=0; {    }
  for i:=1 to nX do
    for j:=i+1 to nX do begin if We[i,j]=$7fff then continue;
      nUo:=nUo+1;
      Uo[2*nUo-1]:=i;
      Uo[2*nUo]:=j;
    end;
  WriteLn(f,'nU =',nUo:3);
  WriteLn(f,'nX =',nX:3);
  Write(f,'X =');for i:=1 to nX do Write(f,i:3);WriteLn(f);
  Write(f,'u1 =');for i:=1 to nUo do Write(f,Uo[2*i-1]:3);WriteLn(f);
  Write(f,'u2 =');for i:=1 to nUo do Write(f,Uo[2*i]:3);WriteLn(f);
  Write(f,'We =');for i:=1 to nUo do Write(f,We[Uo[2*i-1],Uo[2*i]]:3);
  WriteLn(f);
  Ostov;
  Write(f,'uo1=');for i:=1 to nUo do Write(f,Uo[2*i-1]:3);WriteLn(f);
  Write(f,'uo2=');for i:=1 to nUo do Write(f,Uo[2*i]:3);WriteLn(f);
  Write(f,'Woe=');for i:=1 to nUo do Write(f,We[Uo[2*i-1],Uo[2*i]]:3);
  WriteLn(f);
  Wt:=0;
  for i:=1 to nUo do Wt:=Wt+We[Uo[2*i-1],Uo[2*i]];
  Write(f,'Bec=',Wt:3);
  Close(f);
end. {Main}
```

Рассмотрим пример расчета по программе алгоритма 7.10 построения минимального остовного дерева графа, изображенного на рис. 7.21. Исходные данные графа представляются матрицей

весов его ребер в текстовом файле Near.in со следующей структурой:

- в первой строке содержится количество вершин в графе;
- в следующих строках задаются верхние диагональные элементы (нулевые диагональные элементы включаются) строк матрицы весов.

```
7
0   20   0   0   0   23   1
     0  15   0   0    0   4
         0   3   0    0   9
             0  17    0  16
                 0   28  25
                     0   36
                         0
```

Результаты расчетов сохраняются в выходном файле Near.out со следующей структурой:

```
nU  =  12
nX  =  7
X   =  1  2  3  4  5  6  7
u1  =  1  1  1  2  2  3  3  4  4  5  5  6
u2  =  2  6  7  3  7  4  7  5  7  6  7  7
We  =  20 23 1  15 4  3  9  17 16 28 25 36
uo1 =  1  7  7  3  4  1
uo2 =  7  2  3  4  5  6
Woe =  1  4  9  3  17 23
Bec =  57
```

Обозначения данных в файле Near.out соответствуют принятым обозначениям в файле Hungry.out при контрольном расчете остовного дерева по жадному алгоритму 7.8 (см. п. 7.8.1).

## § 7.9. Кратчайшие пути на графе

Рассматриваемый алгоритм определяет расстояния между вершинами в простом орграфе с неотрицательными весами. К таким орграфам сводятся многие типы графов. Если граф не является простым, его можно сделать таковым, отбрасывая все петли и заменяя каждое множество параллельных ребер кратчайшим ребром (ребром с наименьшим весом) из этого множества; каждое неориентированное ребро заменяется парой ориентированных ребер. Если граф не взвешен, то можно считать, что все ребра имеют один вес.

Пусть $\Gamma(X, U, \Phi)$ — простой орграф, для каждого ребра $u \in U$ определен вес $\omega(u) \geqslant 0$. Найдем кратчайший путь между

выделенными вершинами $x_0$ и $z$ (рис. 7.22). Несуществующие ребра будем считать ребрами с бесконечными весами. Сумму весов ребер в пути будем называть весом или длиной пути.

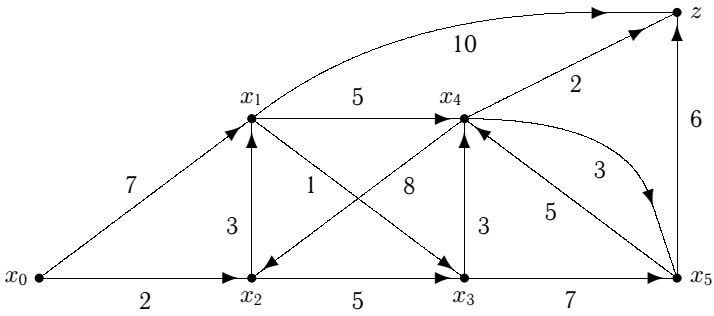



Рис. 7.22. Простой взвешенный орграф

Обозначим $w_{ij} = \omega(u)$ — вес ребра $u = (x_i, x_j)$. Алгоритм поиска кратчайшего пути, начиная из вершины $x_0$, просматривает граф в ширину, помечая вершины $x_j$ значениями-метками их расстояний от $x_0$. Метки могут быть вре́менные или окончательные. Вре́менная метка вершины $x_j$ — это минимальное расстояние от $x_0$ до $x_j$, когда в определении пути на графе учитываются не все маршруты из $x_0$ в $x_j$. Окончательная метка $x_j$ — это минимальное расстояние на графе от $x_0$ до $x_j$. Таким образом, в каждый момент времени работы алгоритма некоторые вершины будут иметь окончательные метки, а остальная их часть — временные. Алгоритм заканчивается, когда вершина $z$ получает окончательную метку, т. е. расстояние от $x_0$ до $z$.

Вначале вершине $x_0$ присваивается окончательная метка 0 (нулевое расстояние до самой себя), а каждой из остальных вершин $|X| - 1$ присваивается временная метка $\infty$ (бесконечность). На каждом шаге одной вершине с временной меткой присваивается окончательная и поиск продолжается дальше. На каждом шаге метки меняются следующим образом.

1. Каждой вершине $x_j$, не имеющей окончательной метки, присваивается новая временная метка — наименьшая из ее временной и числа ($w_{ij}+$ окончательная метка $x_i$), где $x_i$ — вершина, которой присвоена окончательная метка на предыдущем шаге.
2. Определяется наименьшая из всех временных меток, которая и становится окончательной меткой своей вершины. В случае равенства меток выбирается любая из них.

Циклический процесс п.1+п.2 продолжается до тех пор, пока вершина $z$ не получит окончательной метки. Легко видеть, что окончательная метка каждой вершины — это кратчайшее расстояние от этой вершины до начала $x_0$.

Рассмотрим пример поиска кратчайшего пути на графе, представленного на рис. 7.22.

Процесс назначения меток вершинам графа на каждом шаге удобно представить в виде таблицы рис. 7.23.

| | $x_0$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $z$ |
|---|---|---|---|---|---|---|---|
| 0 | **0** | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | | 7 | **2** | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 2 | | **5** | | 7 | $\infty$ | $\infty$ | $\infty$ |
| 3 | | | | **6** | 10 | $\infty$ | 15 |
| 4 | | | | | **9** | 13 | 15 |
| 5 | | | | | | 12 | **11** |

Рис. 7.23. Результаты вычислений кратчайшего пути

Квадратами выделены окончательные метки, т. е. расстояния от них до $x_0$. По такой таблице легко восстановить путь перемещения от $z$ к $x_0$, который отмечен ломаной кривой.

Реализация рассмотренной схемы поиска кратчайшего пути представлена в алгоритме 7.11, где граф $\Gamma(X, U, \Phi)$ представляется матрицей весов $We = [w_{ij}]$, веса несуществующих ребер полагаются равными $\infty$. Вектор $Mark[x]$ меток вершин устанавливает принадлежность вершины $x \in X$ постоянной (TRUE) или временной (FALSE) метке. Вектор $dist[x]$ в алгоритме фиксирует текущие значения меток вершин. Вектор $prev[x]$ позволяет восстановить в обратной последовательности вершины кратчайшего пути.

Вектор $prev[x]$ указывает на ранее пройденную вершину с окончательной меткой, ближайшую к вершине $x$. Последовательность вершин кратчайшего пути будет имеет следующий вид:

$$z, prev[z], prev[prev[z]], prev[prev[prev[z]]], \ldots, x_0,$$

а значение $dist[z]$ составит длину пути из $x_0$ в $z$. Очередная новая вершина, претендующая на постоянную метку, обозначается через $y$.

Алгоритм 7.11.

```
for x ∈ X do begin
  Mark[x] = FALSE;
  dist[x] = ∞;
end;
y = x_0;
Mark[x_0] = TRUE;
dist[x_0] = 0;
while not Mark[z] do begin
  for x ∈ X do begin
    if not Mark[x] and dist[x] > dist[y] + w[y,x] then begin
      dist[x] = dist[y] + w[y,x];
      prev[x] = y;
    end;
  end;
  { y ∈ X }
  dist[y] = min_{x∈X and Mark[x]=FALSE} dist[x];
  Mark[y] = TRUE;
end.
```

• *Сложность алгоритма*. Алгоритм обращается к телу цикла **while** не более $|X|-1$ раз, и число операций, требующихся при каждом таком обращении, равно $O(|X|)$. Тогда сложность алгоритма составит $O(|X|^2)$.

Интересно заметить, что если требуется найти длины кратчайших путей от $x_0$ до всех вершин графа, то в алгоритме 7.11 условие цикла **while not** $Mark[z]$ **do begin** надо заменить на условие дизъюнкции **while** $\bigvee\limits_{x\in X}$ **not** $Mark[x]$ **do begin**. При этом сложность алгоритма останется прежней.

Справедливость алгоритма поиска кратчайшего пути 7.11 является следствием следующей леммы.

Лемма 7.9.1. *Пусть* $\Gamma(X,U,\Phi)$ *— простой орграф с длинами* $\omega(u)\geqslant 0$ *ребер* $u\in U$. *На этом графе, согласно алгоритму* 7.11, *осуществляется поиск кратчайшего пути из вершины* $x_0\in X$ *в вершину* $z\in X$. *Тогда для каждой вершины* $x\in X$ *с окончательной меткой величина* $dist[x]$ *является минимальным расстоянием от* $x_0$ *до* $x$.

Доказательство — это индукция по количеству вершин с окончательными метками. Отметим, что на каждом шаге (цик-

ле) алгоритма одна из вершин с временной меткой переходит в разряд вершин с окончательными метками.

*Установочный шаг индукции.* Вершина $x_0 \in X$ — начало пути. Ей назначается первая окончательная метка и значение $dist[x_0] = 0$ — это минимальное расстояние от $x_0$ до $x_0$. Условие индукции выполнено.

*Шаг предположения индукции.* Пусть верно, что если в графе $\Gamma(X, U, \Phi)$ количество вершин $x \in X$ с окончательными метками не превосходит $k$, то для каждой такой вершины $x$ величина $dist[x]$ — это минимальное расстояние для нее от $x_0$.

*Шаг распространения индукции.* Обозначим через $M = \{x_0, x_1, x_2, \ldots, x_{k-1}\} \subset X$ множество вершин с окончательными метками, установленными в течение первых $k$ шагов алгоритма. По предположению индукции значения $dist[x_i]$ являются минимальными расстояниями от $x_0$ до $x_i \in M$.

Пусть $y \in X$ — это вершина, которой на $(k+1)$-м шаге назначается окончательная метка. Необходимо показать, что установленное значение $dist[y]$ является минимальным расстоянием от $x_0$ до $y$. Обозначим этот маршрут и его длину как $W_s(y)$. Допустим противное. Пусть существует маршрут $W_t(y)$ из $x_0$ в $y$, длина которого меньше $dist[y]$, т. е. $W_t(y) < W_s(y)$. Для нового пути $W_t(y)$ возможны два случая конфигурации составляющих его ребер.

*Случай 1.* Новый маршрут $W_t(y)$ включает ребро $(x_2, y)$, где $x_2 \in M$ (рис. 7.24, a) ). По предположению индукции величина $dist[x_2]$ — минимальное расстояние от $x_0$ до $x_2$. По алгоритму построения маршрута $W_s(y)$ его величина $W_s(y) = dist[x_1] + w[x_1, y] = \min\limits_{x \in M}\{dist[x] + w[x, y]\} \leqslant dist[x_2] + w[x_2, y] = W_t(y)$, где $x_1, x_2 \in M$. Это противоречит предположению $W_t(y) < W_s(y)$, а значит, в рассматриваемой конфигурации ребер маршрута $W_t(y)$ его длина не меньше длины алгоритмического пути $W_s(y) = dist[y]$.

*Случай 2.* Новый маршрут $W_t(y)$ включает вершины с временными метками. Не уменьшая общности, полагаем, что этот

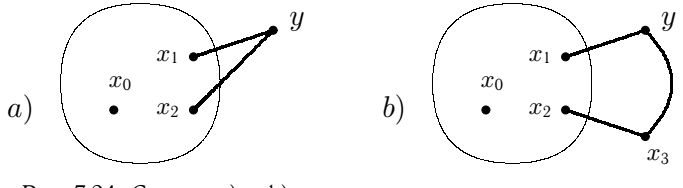



Рис. 7.24. Случаи a) и b) изменения минимального маршрута

маршрут $W_t(y)$ включает ребра $(x_2, x_3)$, $(x_3, y)$, где $x_2 \in M$, $x_3 \notin M$, $x_3 \neq y$ (рис. 7.24, b)). По предположению индукции величина $dist[x_2]$ — это минимальное расстояние от $x_0$ до $x_2$. Если считать, что $W_t(y)$ — это минимальный путь из $x_0$ в $y$ через $x_3 \notin M$, то $W_t(x_3) < W_t(y)$, так как $w[x_3, y] > 0$. Это означает, что по построению в алгоритме маршрута $W_s(y)$ на его $(k+1)$-м шаге в качестве вершины $y \in X$ должны были бы выбрать вершину $x_3 \in X$, так как $W_t(x_3) = dist[x_2] + w[x_2, x_3] < W_t(y) < W_s(y) = dist[x_1] + w[x_1, y]$, где $x_1, x_2 \in M$. Следовательно, и эта возможная конфигурация ребер нового маршрута $W_t(y)$ (рис. 7.24, b)) не позволяет построить маршрут меньшей длины, чем путь $W_s(y)$ в алгоритме (лемма доказана). □

Программная реализация алгоритма поиска кратчайшего пути представлена в алгоритме 7.12 на **Pascal**'е, который близко соответствует множественному описанию алгоритма 7.11.

Алгоритм 7.12.

```
Program Short; {  }
uses CRT,DOS;
Const
  nVertex=50; { }
Type
  TypeMark=array[0..nVertex] of Boolean;
  TypeDist=array[0..nVertex] of LongInt;
  TypePrev=array[0..nVertex] of Integer;
  TypeWeight=array[0..nVertex,0..nVertex] of Integer;
Var
  f  :Text; {}
  nX :Integer; {  }
  Mark :TypeMark; {   }
  Dist :TypeDist; {     }
  Prev :TypePrev; {  }
  We :TypeWeight; {  }
  x0 :Integer; { }
  z  :Integer; { }
  y  :Integer; {   }
Var
  i,j,x :Integer;
  weight :LongInt;
```

```
begin
  Assign(f,'Short.in');
  Reset(f); {  }
  { }
  Read(f,x0); { }
  Read(f,z); { }
  Read(f,nX); {  }
  nX:=nX-1;
  for i:=0 to nX do begin
    for j:=0 to nX do begin
      Read(f,We[i,j]); {  }
      if We[i,j]=0 then We[i,j]:=$7fff; {  }
    end;
  end;
  Close(f);
  Assign(f,'Short.out');
  Rewrite(f); {  }
  for x:=0 to nX do begin
    Mark[x]:=FALSE;
    Dist[x]:=$7fffffff;
  end;
  y:=x0; {   }
  Mark[y]:=TRUE;
  Dist[y]:=0;
  while not Mark[z] do begin
    { }
    for x:=0 to nX do
      if not Mark[x] and ( Dist[x]>Dist[y]+We[y,x] ) then begin
        Dist[x]:=Dist[y]+We[y,x];
        Prev[x]:=y;
      end;
    {   }
    weight:=$7fffffff;
    for x:=0 to nX do if not Mark[x] then if weight>Dist[x] then begin
      weight:=Dist[x];
      y:=x;
    end;
    Mark[y]:=TRUE;
```

```
  end;
  Write(f,'Вершины пути=');
  x:=z;
  while x<>x0 do begin
    Write(f,x:2);
    x:=Prev[x];
  end;
  WriteLn(f,x:2);
  WriteLn(f,'Длина пути= ',Dist[z]);
  Close(f);
end.
```

Рассмотрим пример расчета по программе алгоритма 7.12 поиска кратчайшего пути на графе, показанном на рис. 7.22. Исходные данные графа представляются матрицей весов его ребер в текстовом файле Short.in со следующей структурой:

- в первой строке — номер вершины $x_0$ начала пути;
- во второй строке — номер вершины $z$ окончания пути;
- в третьей строке указывается количество $nX$ вершин в графе;
- в следующих $nX$ строках определяются строки матрицы весов $[w_{ij}]$ графа.

```
0
6
7
0  7  2  0  0  0  0
0  0  0  1  5  0  10
0  3  0  5  0  0  0
0  0  0  0  3  7  0
0  0  8  0  0  3  2
0  0  0  0  5  0  6
0  0  0  0  0  0  0
```

Результаты расчетов сохраняются в выходном файле Short.out со следующей структурой:

```
Вершины пути = 6  4  3  1  2  0
Длина пути = 11.
```

## § 7.10. Потоки в сетях

Определение. *Транспортной сетью* называется связный ориентированный граф без петель $\Gamma(X, U, \Phi)$ с выделенной парой вершин $x_0$ и $z$ (рис. 7.25). Вершина $x_0$ — начало транспортной

сети, из которой дуги только выходят. Вершина $z$ — конец транспортной сети, в которую дуги только входят. На множестве дуг $u \in U$ задана целочисленная функция $c(u) \geqslant 0$, где $c(u)$ — пропускная способность дуги.

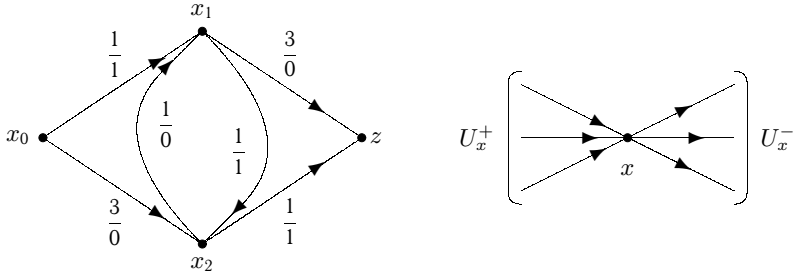



Рис. 7.25. Транспортная сеть

Определение. *Потоком* по транспортной сети называется целочисленная функция $\varphi(u) \geqslant 0$, заданная на множестве дуг $u \in U$ и обладающая следующими свойствами:

$$\forall u \in U \quad \varphi(u) \leqslant c(u) \qquad \text{и} \qquad \sum_{u \in U_x^+} \varphi(u) = \sum_{u \in U_x^-} \varphi(u), \tag{7.1}$$

где $x$ — внутренняя вершина графа, т. е. $x \neq x_0, x \neq z$;
$U_x^+$ — множество дуг, входящих в вершину $x$;
$U_x^-$ — множество дуг, выходящих из вершины $x$ (рис. 7.25).

На рис. 7.25 напротив каждой дуги стоит дробь, числитель которой — пропускная способность дуги, знаменатель — поток по дуге. Свойство 7.1 утверждает, что поток, входящий в вершину, равен выходящему потоку (поток в вершинах не скапливается). Обозначим

$$\varphi(z) = \sum_{u \in U_z^+} \varphi(u) \text{ и } \varphi(x_0) = \sum_{u \in U_{x_0}^-} \varphi(u), \tag{7.2}$$

где $\varphi(z)$ — поток, входящий в вершину $z$; $\varphi(x_0)$ — поток, выходящий из вершины $x_0$.

Утверждение 7.10.1. $\varphi(z) = \varphi(x_0)$.

Доказательство. Действительно, сумма $\sum\limits_{x \in X} [\sum\limits_{u \in U_x^+} \varphi(u) - \sum\limits_{u \in U_x^-} \varphi(u)] = 0$, так как $\forall u \in U$ величина $\varphi(u)$ суммируется

дважды — со знаком «+» и «−». Здесь

$$\sum_{x\in X}\Big[\sum_{u\in U_x^+}\varphi(u)-\sum_{u\in U_x^-}\varphi(u)\Big]=$$
$$=\sum_{x\in X\setminus\{x_0,z\}}\Big[\sum_{u\in U_x^+}\varphi(u)-\sum_{u\in U_x^-}\varphi(u)\Big]+[\varphi(z)-\varphi(x_0)]=0,$$

где первая часть выражения равна нулю вследствие (7.1). □

Определение. Пусть $A\subset X$ — множество вершин транспортной сети: $x_0\notin A, z\in A$. Обозначим через $U_A^+$ множество дуг, входящих в $A$. Это множество дуг будем называть *разрезом транспортной сети*.

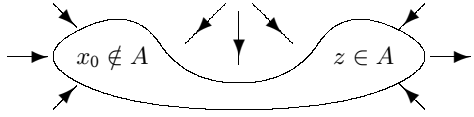



Рис. 7.26. Разрез транспортной сети

$C(A)=\sum\limits_{u\in U_A^+}c(u)$ — называется мощностью разреза. Это максимально возможный поток, входящий в $A$ по дугам разреза.

$\varphi(A)=\sum\limits_{u\in U_A^+}\varphi(u)$ — поток, входящий в $A$, по дугам разреза. Ясно, что $\varphi(A)\leqslant C(A)$, так как $\forall u\in U\ \varphi(u)\leqslant c(u)$.

Утверждение 7.10.2. $\varphi(z)\leqslant C(A)$.

Действительно, из рис. 7.26 видно, что не весь поток, входящий в $A$, скатывается в $z$. Часть потока может выходить из $A$. Значит, $\varphi(z)\leqslant\varphi(A)$, но $\varphi(A)\leqslant C(A)$, тогда и $\varphi(z)\leqslant C(A)$. □

Теорема 7.10.1 (Форда и Фалкерсона). *Максимальный поток по транспортной сети равен мощности минимального разреза, т. е.*

$$\max_{\varphi}\varphi(z)=\min_{A}C(A).$$

Доказательство теоремы — это алгоритм определения максимального потока по сети. Алгоритм состоит из двух частей.

1. *Насыщение потока*. Поток называется насыщенным, если любой путь из $x_0$ в $z$ содержит дугу $u\in U$, для которой

$\varphi(u) = c(u)$. Задача первой части алгоритма состоит в насыщении потока.

1.1. Зададим произвольный начальный поток. Например, нулевой на всех дугах: $\forall u \in U\ \varphi(u) = 0$.

1.2. Поиск пути из $x_0$ в $z$. Если путь найден, то переход к пункту 1.3. Если путь не найден, то переход к пункту 1.5.

1.3. Увеличиваем поток по найденному пути таким образом, чтобы одна из дуг стала насыщенной.

1.4. Условно разрываем насыщенную дугу и переходим к пункту 1.2 на поиск пути из $x_0$ в $z$.

1.5. Сеть насыщена и *«разорвана»*.

2. *Перераспределение потока*. Итак, поток насыщен, как в примере на рис. 7.27. Пометим рекурсивным образом все возможные вершины $x_i$ сети.

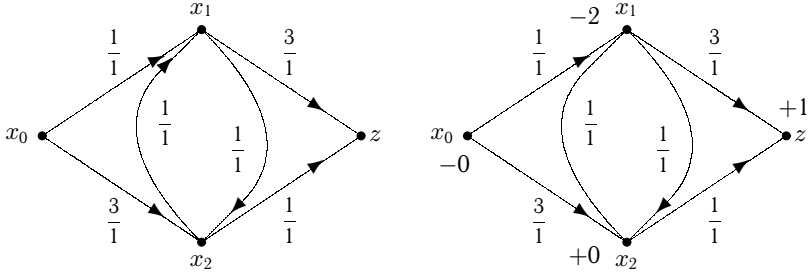



Рис. 7.27. Насыщенная транспортная сеть и пометка вершин

2.1. Вершину $x_0$ пометим $-0$.

2.2. Пусть $x_i$ — любая из уже *помеченных* вершин; $y$ — произвольная *непомеченная* вершина, смежная $x_i$. Вершину $y$ помечаем $+i$, если данные вершины соединены ненасыщенным ребром $x_i \to y(+i)$, и помечаем $-i$, если соединены непустым ребром $x_i \leftarrow y(-i)$. После пометки вершин возможны два случая: вершина $z$ оказалась либо помеченной, либо непомеченной.

2.3. Вершина $z$ оказалась помеченной. Значит, существует последовательность помеченных вершин от $x_0$ к $z$. В этой последовательности каждая последующая вершина помечена номером предыдущей, как на рисунках 7.27 и 7.28. Определим на дугах новый поток, увеличивая на единицу поток на дугах, ориентированных по направлению движения от $x_0$ к $z$, и уменьшая поток на единицу на дугах, направленных против этого движения, как на рис. 7.28.


Рис. 7.28. Перераспределение потока на основе пометки вершин

Заметим, что поток можно увеличивать (уменьшать) на прямых (обратных) дугах настолько, пока одна из дуг не станет насыщенной (пустой). Далее вновь переходим к пометке вершин (пункт 2.1). Выполненное перераспределение потока сохраняет все его свойства и увеличивает на единицу поток в вершину $z$. Таким образом, пометка вершины $z$ позволяет увеличить поток как минимум на единицу, а значит, алгоритм конечен, т. е. наступит момент, когда вершина $z$ останется непомеченной.

2.4. Вершина $z$ осталась непомеченной. В рассматриваемом примере это показано на рис. 7.29. Пусть $A^*$ — множество всех непомеченных вершин. Тогда дуги, входящие в эти вершины, насыщенные, а выходящие — пустые. В примере $A^* = \{x_1, z\}$. Множество $A^*$ определяет разрез, так как $x_0 \notin A^*, z \in A^*$. Таким образом, мы нашли поток $\varphi^*$ и разрез $A^*$, для которых выполняется $\varphi^*(A^*) = C(A^*)$. Так как выходящие дуги из $A^*$ пустые, то весь поток $\varphi^*(A^*)$ скатывается в $z$, т. е. $\varphi^*(z) = C(A^*)$.

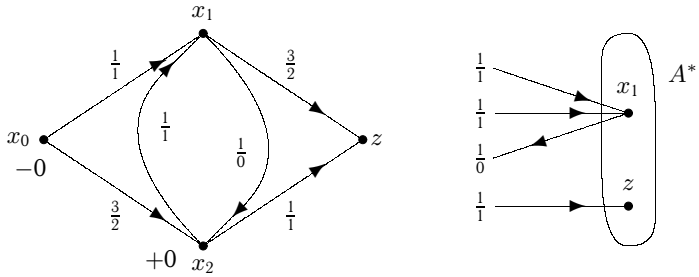



Рис. 7.29. Максимальный поток

Покажем, что $\varphi^*$ — максимальный поток, а $A^*$ определяет минимальный разрез. Так как для любого потока $\varphi$ и любого разреза $A$ справедливо $\varphi(z) \leqslant C(A)$, то выполняется соотношение

$$\varphi^*(z) \leqslant \max_{\varphi} \varphi(z) \leqslant \min_{A} C(A) \leqslant C(A^*).$$

Для найденного разреза и потока справедливо равенство $\varphi^*(z) = C(A^*)$. Отсюда следует, что $\max\limits_{\varphi} \varphi(z) = \min\limits_{A} C(A)$. Рассмотрен-

ный алгоритм позволяет найти именно максимальный поток $\varphi^*$ и множество $A^*$, определяющее минимальный разрез.

В рассматриваемом примере $\varphi^*(z) = 3$ и $C(A^*) = c(x_0, x_1) + c(x_2, x_1) + c(x_2, z) = 1 + 1 + 1 = 3$. □

Для примера на рис. 7.25 найдем все разрезы рассмотренной транспортной сети:

$A_1 = \{z\},\ C(A_1) = c(x_1, z) + c(x_2, z) = 3 + 1 = 4;$
$A_2 = \{z, x_1\},\ C(A_2) = c(x_0, x_1) + c(x_2, x_1) + c(x_2, z) = 1 + 1 + 1 = 3;$
$A_3 = \{z, x_2\},\ C(A_3) = c(x_0, x_2) + c(x_1, x_2) + c(x_1, z) = 3 + 1 + 3 = 7;$
$A_4 = \{z, x_1, x_2\},\ C(A_4) = c(x_0, x_1) + c(x_0, x_2) = 1 + 3 = 4.$

• *Сложность алгоритма*. Если следовать утверждению теоремы, то значение максимального потока можно найти прямым перебором всех разрезов. Подсчитаем общее число разрезов. Обозначим через $n = |X| - 2$, $n$ — количество узлов в сети без вершин $x_0$ и $z$. Число сочетаний $C_n^k$ равно числу разрезов, каждый из которых содержит $k$ вершин $x_i$ и вершину $z$. Тогда число разрезов равно $C_n^0 + C_n^1 + \ldots + C_n^n = 2^n$, из которых надо выбрать один разрез. Следовательно, сложность полного перебора всех разрезов является экспоненциальной $O(2^{|X|})$.

Оценим количество операций в алгоритме Форда и Фалкерсона. Часть алгоритма, относящаяся к пометке вершин, требует перебора всех вершин, т. е. количества операций порядка $O(|X|)$. Каждое увеличение потока приводит к появлению дуги с нулевым или насыщенным потоком по ней. Такая дуга далее не участвует в пометке вершин. Всего дуг $|U|$, а значит, для окончания алгоритма требуется не более $O(|X| \cdot |U|)$ операций приращения потока. Сложность алгоритма является степенной.

## § 7.11. Клики, независимые множества

Определение. Пусть $\Gamma(X, U, \Phi)$ — ориентированный или неориентированный граф. Подмножество $D \subseteq U$ называется *доминирующим множеством* для графа, если каждая не принадлежащая $D$ вершина является конечной вершиной некоторого ребра от вершины, принадлежащей $D$. Например, если рассмотреть $\Gamma$ как граф, представляющий игру, с вершинами, соответствующими позициям, и ребрами–ходами, то доминирующим множеством $D$ будет такое множество позиций, что во все остальные позиции можно попасть из $D$ за один ход.

Определение. *Минимальным доминирующим множеством* называется такое доминирующее множество, что никакое его подмножество не обладает этим свойством. *Числом доминирования* $\delta(\Gamma)$ графа называется наименьшее число вершин, составляющих минимальное доминирующее множество. Число $\delta(\Gamma)$ появляется в различных задачах. Например, требуется разместить на шахматной доске минимальное число ферзей так, чтобы они держали под боем каждую клетку доски. Для решения задачи достаточно 5 ферзей; меньшего числа недостаточно. Тогда для графа этой игры число доминирования $\delta(\Gamma) = 5$, где клетки доски — это вершины графа; две клетки связаны неориентированным ребром, если ферзь, поставленный на одну из клеток, угрожает другой.

В качестве примера рассмотрим вариант этой игры, когда расстановка ферзей выполняется на доске $3 \times 3$ (рис. 7.30, *a*) ). На рис. 7.30, *b*) представлен граф данной игры. Вершины 1 и 9 соединены ребром, так как ферзь в клетке 1 держит под боем клетку с номером 9. Вершины 1 и 8 не соединены, так как ферзь в клетке 1 не может угрожать клетке 8 и т. д. Вершины 1 и 8 (рис. 7.30, *c*) ) составляют доминирующее множество $D = \{1, 8\}$, так как из этих вершин по ребрам графа можно за один ход попасть в любую другую вершину. Одна вершина с номером 5 (рис. 7.30, *d*) ) также составляет доминирующее множество $D = \{5\}$. Ясно, что для графа этой игры число доминирования равно $\delta(\Gamma) = 1$.

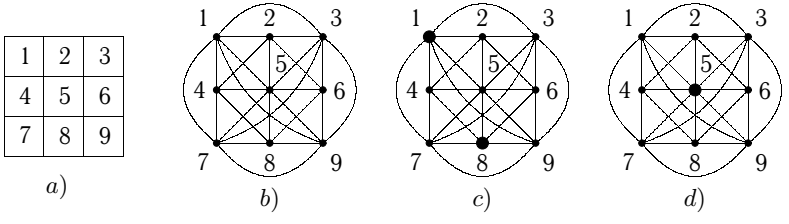



Рис. 7.30

Определение. Пусть граф $\Gamma(X, U, \Phi)$ — неориентированный и без петель. *Множество $I \subseteq U$ вершин* называется *независимым*, если между любыми его вершинами нет соединяющих ребер. В *зависимом множестве* хотя бы две вершины соединены ребром. Множество $Q$ *полностью зависимое*, если каждая пара его вершин соединена. Вершины графа, составляющие $Q$, образуют полный подграф.

Определение. *Максимальное независимое множество* есть независимое множество, которое становится зависимым после добавления к нему любой вершины. Заметим, что каждое независимое множество содержится в некотором максимальном независимом множестве. Максимальное число $\beta(\Gamma)$ вершин, составляющих независимое множество, называется числом (*вершинной*) *независимости графа*.

Определение. Подобно независимым множествам вершин рассматриваются *независимые множества ребер*, состоящие из ребер, не имеющих общих вершин. Каждое независимое множество ребер содержится в некотором максимальном независимом множестве. Число ребер в максимальном независимом множестве называется *числом реберной независимости* $\beta_u(\Gamma)$.

При представлении игр графами независимые множества вершин являются такими множествами позиций, что никакая из них не может быть достигнута из другой за один ход. Примером является задача о расположении максимального числа ферзей на шахматной доске так, чтобы ни один из них не мог побить другого. Это максимальное число равно $\beta(\Gamma) = 8$.

Утверждение 7.11.1. *Независимое множество максимально тогда и только тогда, когда оно доминирующее, а значит,* $\beta(\Gamma) \geqslant \delta(\Gamma)$ *— число* (*вершинной*) *независимости не может быть меньше числа доминирования.*

Доказательство. ($\Rightarrow$) Если $I \subseteq U$ — максимальное независимое множество, то не может быть вершин $k \in \overline{I}$, не соединенных с множеством $I$ ребром, так как в противном случае множество $k \cup I$ также было бы независимым, но $I$ — максимальное по условию. Отсюда $I$ — доминирующее.

($\Leftarrow$) Пусть $I$ — независимое доминирующее множество. Тогда никакое $k$ нельзя перевести из $\overline{I}$ в $I$ так, чтобы $I \subseteq U$ осталось независимым, а значит, $I$ — максимальное. □

Определение. *Клика* есть полностью зависимое множество, которое теряет это свойство после добавления любой вершины. Клики графа представляют «естественные» группировки вершин в максимальные полные подграфы.

Определение клик графа полезно в кластерном анализе в таких областях, как информационный поиск, в социологии и др. В качестве примера на рис. 7.31 показан граф и его клики.

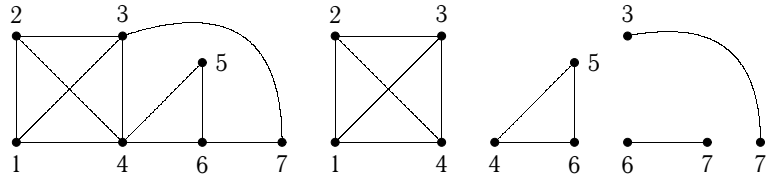



Рис. 7.31. Граф и его клики

Замечание. Если предположить, что граф $\Gamma(X, U, \Phi)$ простой, то полностью зависимые множества (клики) в $\Gamma$ становятся максимально независимыми множествами в дополнительном графе $\overline{\Gamma}$, верно и обратное.

При алгоритмическом подходе к выделению клик в графе применяют *метод поиска с возвращением* по специальному *дереву поиска*, устроенному следующим образом. Каждый узел в дереве поиска соответствует полному подграфу исходного графа, и каждое ребро дерева поиска соответствует вершине исходного дерева. Вершины (множества) дерева поиска определим рекурсивно. Корень дерева поиска — пустое начальное множество $S = \varnothing$. Пусть теперь $S$ — произвольная вершина дерева поиска какого-либо уровня. Тогда вершиной следующего уровня дерева поиска будет вершина $S \cup \{x\}$, если $x \notin S$ и $x$ смежна с каждой вершиной из $S$. В дереве поиска такие вершины $S$ и $S \cup \{x\}$ соединяются ребром, которое соответствует вершине $x$. На рис. 7.32 показаны некоторый граф $\Gamma$ и дерево поиска $T$, которое исследуется в процессе поиска с возвращениями клик графа полным перебором.

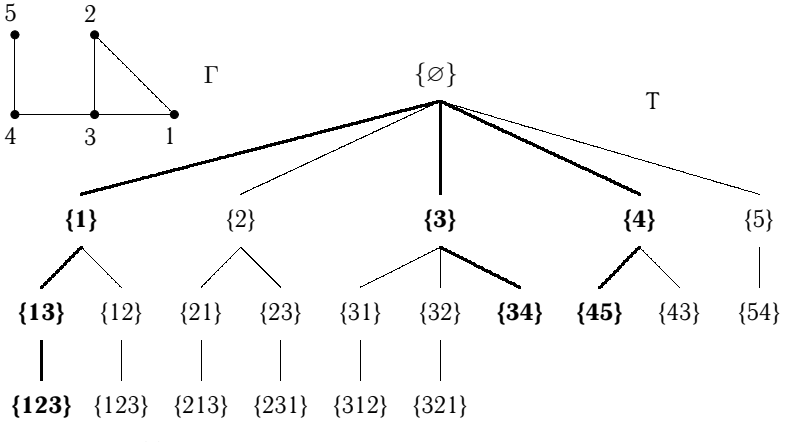



Рис. 7.32. Граф $\Gamma$ и полный перебор дерева $T$ поиска клик

Заметим, что каждая клика порождается много раз: клика $\{1,2,3\}$ порождается 3! раз, клики $\{3,4\}$, $\{4,5\}$ порождаются 2!. В общем случае клика размера $k$ порождается $k!$ раз. Все тонкие ребра на рис. 7.32 исследования дерева поиска можно оборвать, они не приводят к новым кликам. Следующие два утверждения позволяют обрывать такие *«тонкие»* ребра (не исследовать их), обеспечивая целенаправленный проход по дереву поиска клик графа.

Утверждение 7.11.2. *Пусть* $\Gamma(X,U,\Phi)$ *— исходный граф,* $S$ *— узел в дереве поиска* $T$ ($S \subseteq X$ *— подмножество вершин графа*). *Вершина* $S$ *дерева поиска уже обработана и первой вершиной, которую надо исследовать, является множеством* $S \cup \{x\}$, *как на рис.* 7.33, *вершина* $x$ *смежна с каждой вершиной из* $S$. *Пусть все поддеревья узла* $S \cup \{x\}$ *в дереве* $T$ *уже исследованы и порождены все клики, включающие* $S \cup \{x\}$. *Тогда необходимо исследовать только те из вершин* $S \cup \{v_i\}$, *для которых* $v_i \notin Adj[x]$ (*рис.* 7.33).

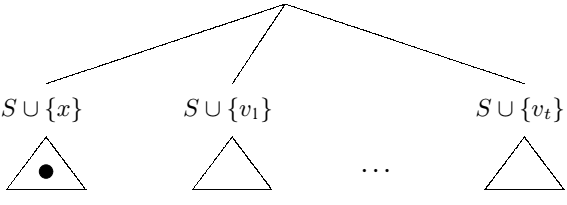



Рис. 7.33. Поддеревья с корнями $S \cup \{v_i\}$ (эти вершины смежны с $S$)

Утверждение 7.11.3. *Пусть* $S$ *— узел в дереве поиска* $T$, *и пусть* $\widetilde{S} \subseteq S$ *— предок* $S$ *в* $T$. *Если все поддеревья узла* $\widetilde{S} \cup \{x\}$ *уже исследованы, так что порождены все клики, включающие* $\widetilde{S} \cup \{x\}$, *то все неисследованные поддеревья с корнями* $S \cup \{x\}$ *можно игнорировать.*

Алгоритм 7.13 порождения клик графа представляет собой процедуру поиска с возвращением и является наиболее сложным из всех ранее рассмотренных алгоритмов. Рекурсивная процедура $CLIQUE$ имеет два параметра $N$ и $D$. Для рассматриваемого узла поиска $S$ объединение $N \cup D$ представляет множество вершин, смежных с каждой вершиной из $S$, т. е. $N \cup D$ — множество вершин, которые могут выступать в качестве продолжения поиска клик из вершины $S$. Множество $N$ состоит только из новых вершин, которые могут быть добавлены к $S$ в процессе поиска

клик, т. е. $N = \{x \in X \mid (x, s) \in U\, \forall s \in S \wedge \forall \widetilde{S} \subseteq S$ *ни одно из поддеревьев* $\widetilde{S} \cup \{x\}$ *не исследовано*$\}$.

Алгоритм 7.13.

$S = \varnothing$; { $T$}
$N = X$;
$D = \varnothing$;
$Z = X$;
$CLIQUE(N, D, Z)$; { $T$}
Procedure $CLIQUE(N, D, Z)$;
  if $N \cup D = \varnothing$ then $S$
  else if $N \neq \varnothing$ then begin
    $x \in N$; { }
    $View(x)$; { }
    { }
    $Z = Z \setminus \{x\}$;
    $Z = Z \setminus Adj[x]$;
    while $Z \neq \varnothing$ do begin
      $v \in Z$;
      $View(v)$; { }
      $Z = Z \setminus \{v\}$;
    end;
  end;
end;
Procedure $View(v)$; { }
  $N = N \setminus \{v\}$;
  $S = S \cup \{v\}$;
  $CLIQUE(N \cap Adj[v], D \cap Adj[v], N \cap Adj[v])$;
  $S = S \setminus \{v\}$;
  $D = D \cup \{v\}$;
end.

Процедура $CLIQUE$ выбирает произвольную вершину $x \in$ $\in X$, удаляет ее из $N$ и исследует поддерево $S = S \cup \{x\}$, обращаясь к процедуре $View$. Далее, согласно утверждениям 7.11.2 и 7.11.3 при помощи процедуры исследуются только поддеревья $S = S \cup \{v\}$, где $v \in N, v \notin Adj[x]$, что соответствует условию $v \in Z$. Множество $Z$ состоит из вершин одного уровня дерева, которые должны быть исследованы при рекурсивном проходе по дереву поиска, чтобы не потерять ни одной клики графа.

Второй параметр $D$ процедуры $CLIQUE$ представляет собой множество вершин, смежных со всеми вершинами из $S$, но таких, которые не надо добавлять к $S$ на предмет продолжения формирования клик. $D = \{x \in X \mid (x,s) \in U \ \forall s \in S$ *и поддерево* $\widetilde{S} \cup \{x\}$ *исследовалось для некоторых* $\widetilde{S} \subseteq S\}$.

По алгоритму множество $S \subseteq X$ является полным подграфом графа $\Gamma(X, U, \Phi)$ и $N \cup D$ — множество всех вершин, смежных с каждой вершиной в $S$.

- Множество $S$ будет кликой тогда и только тогда, когда $N \cup \cup D = \varnothing$.
- Условие $N = \varnothing$ и $D \neq \varnothing$ обозначает, что все клики, включающие $S$, уже ранее порождались.
- При $N \neq \varnothing$ могут оставаться клики, включающие $S$, которые еще не порождались. Исследование таких поддеревьев $S \cup \cup \{x\}, x \in N$, необходимо продолжить.

Основные усилия алгоритма 7.13 порождения клик графа направлены на поддержание множеств $N$ и $D$, текущее состояние которых, согласно перечисленным выше условиям, предопределяет исследования по дереву поиска. Программная реализация алгоритма порождения клик графа представлена в алгоритме 7.14 на Pascal'е, который близко соответствует множественному описанию алгоритма 7.13. Отметим, что в программной реализации передача множеств $N, D, Z$ в качестве параметров процедур выполнена посредством указателей $kN, nN, kD, nD, kZ, nZ$, где $kN, kD, kZ$ — указатели начала вершин в множествах соответственно $N, D, Z$, а $nN, nD, nZ$ — количество вершин в каждом из этих множеств.

Алгоритм 7.14.

```
Program PgmClique; { }
uses CRT,DOS;
Const
  nVertex=100; { }
  nAdjacent=1000; {  }
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeAdjacent=array[1..nAdjacent] of Integer;
Var
  f :Text; { }
```

```
  m  :Integer; {    }
  Adj :TypeAdjacent; {   }
  Fst :TypeVertex; {    }
  Nbr :TypeVertex; {     }
  Vtx :TypeVertex; {   }
  S  :TypeVertex; {    }
  nS :Integer; {       }
  N  :TypeVertex; {      }
  {                    }
  D  :TypeVertex; {      }
  {                    }
  Z  :TypeVertex; {    }
  {                    }

Procedure PrintClique; FORWARD;
Procedure Subtract( x:Integer; Var kZ,nZ:Integer ); FORWARD;
Procedure InterSection( v,kM,nM:Integer; Var kMw,nMw:Integer;
                                          Var M:TypeVertex ); FORWARD;
Procedure View(v:Integer; Var kN,nN, kD,nD, kZ,nZ:Integer );
                                                          FORWARD;
Procedure Clique( kN,nN,kD,nD,kZ,nZ:Integer ); FORWARD;

Procedure Init( Var yes :Boolean );
{ }
{       }
{            }
Var
  i,j,k :Integer;
begin
  for i:=1 to m do
    for j:=1 to Nbr[i] do begin
      yes:=FALSE;
      for k:=1 to m do
      if Adj[Fst[i]+j]=Vtx[k] then begin
        yes:=TRUE;
        Adj[Fst[i]+j]:=k;
        break;
      end;
```

```
        if not yes then exit;
      end;
end
Procedure PrintClique; {    }
Var
  i :Integer;
begin
  for i:=1 to nS do Write(f,Vtx[S[i]]:3); WriteLn(f);
end;

Procedure InterSection( v,kM,nM:Integer;
          Var kMw,nMw:Integer; Var M:TypeVertex );
{     Adj          Adj     }
Var
  i,j,k :Integer;
  yes :Boolean;
begin
  {                  Adj    }
  {                  Adj    }
  kMw:=kM+nM;
  nMw:=0;
  for i:=1 to nM do
    for j:=1 to Nbr[v] do
      if M[kM+i]=Adj[Fst[v]+j] then begin
      nMw:=nMw+1;
      M[kMw+nMw]:=M[kM+i];
      break;
    end;
end;

Procedure Subtract( x:Integer; Var kZ,nZ:Integer );
     Z = Z\{x}   Z = Z\Adj[x]
Var
  i,j,k :Integer;
  yes :Boolean;
begin
      Z = Z\{x}
  for i:=1 to nZ do
    if Z[kZ+i]=x then begin
```

```
        nZ:=nZ-1;
        for k:=i to nZ do Z[kZ+k]:=Z[kZ+k+1];
        break;
      end;
      Z = Z\Adj[x]
  for j:=1 to Nbr[x] do
    for i:=1 to nZ do
      if Z[kZ+i]=Adj[Fst[x]+j] then begin
        nZ:=nZ-1;
        for k:=i to nZ do Z[kZ+k]:=Z[kZ+k+1];
        break;
      end;
end

Procedure Clique( kN,nN,kD,nD,kZ,nZ:Integer );
{   }
Var
  i,j,x :Integer;
begin
  if (nN=0) and (nD=0) then { }
  PrintClique { }
  else if nN<>0 then begin {  }
    x:=N[kN+nN]; {       }
    View(x,kN,nN,kD,nD,kZ,nZ); {  }
    Subtract(x,kZ,nZ);     Z = Z\{x}   Z = Z\Adj[x]
    while nZ<>0 do begin {
                                                          }
      x:=Z[kZ+nZ];
      View(x,kN,nN,kD,nD,kZ,nZ);
      nZ:=nZ-1;     Z = Z\{x}
    end;
  end;
end;

Procedure View( v:Integer; Var kN,nN,kD,nD,kZ,nZ:Integer );
{   }
Var
  i,k :Integer;
  kNw,nNw :Integer;
```

```
  kDw,nDw :Integer;
  kZw,nZw :Integer;
begin
      N = N\{v}
  for i:=1 to nN do
    if N[kN+i]=v then begin
      nN:=nN-1;
      for k:=i to nN do N[kN+k]:=N[kN+k+1];
      break;
    end;
      S = S + {v}
  nS:=nS+1;
  S[nS]:=v;

  InterSection(v,kN,nN,kNw,nNw,N); N и Adj[v]
  InterSection(v,kD,nD,kDw,nDw,D); D и Adj[v]

  kZw:=kZ+nZ;
  nZw:=nNw;
  for i:=1 to nNw do Z[kZw+i]:=N[kNw+i];

  Clique(kNw,nNw,kDw,nDw,kZw,nZw);
      S = S\{v}
  nS:=nS-1;
      D = D + {v}
  nD:=nD+1;
  D[kD+nD]:=v;
end;

Var
  kN,nN, kD,nD, kZ,nZ :Integer;
  i,j :Integer;
  yes :Boolean;
begin
  Assign(f,'Clique.in');
  Reset(f); {  }
  { }
  Read(f,m); {  }
  Fst[1]:=0; {  }
```

```
  for i:=1 to m do begin
    Read(f,Vtx[i]); { }
    Read(f,Nbr[i]); {  }
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]); {
                                                          }
    Fst[i+1]:=Fst[i]+Nbr[i]; {
                                                           }
  end;
  Close(f);
  Assign(f,'Clique.out');
  Rewrite(f); {  }
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа!');
    Close(f);
    exit;
  end;
  {              }
  nS:=0; S – пустое
  kD:=0; nD:=0; {      }
  kN:=0; nN:=m; {       }
  for i:=1 to m do N[i]:=i;
  kZ:=0; nZ:=m; {        }
  for i:=1 to m do Z[i]:=i;
  Clique(kN,nN,kD,nD,kZ,nZ); {
                                                          }
  Close(f);
end.
```

Воспользуемся данной программой, в качестве примера, для решения следующей задачи.

З а д а ч а. *Симпатичный прием*. Генерал желает устроить свой юбилей с максимальным числом гостей из своих знакомых. Стремясь сделать юбилейный вечер приятным, он должен организовать все так, чтобы на нем присутствовали люди, симпатизирующие друг другу. Для достижения цели ему придется находить максимальную клику графа своих знакомых. Этот граф устроен следующим образом. Вершины его — знакомые юбиляра. Две вершины смежные, если соответствующие

знакомые симпатизируют друг другу. Нетрудно понять, что клика этого графа с максимальным числом вершин и представляет тот самый максимальный контингент приглашенных, который может позволить себе юбиляр. *«Симпатичный»* граф знакомых генерала представлен на рис. 7.34.

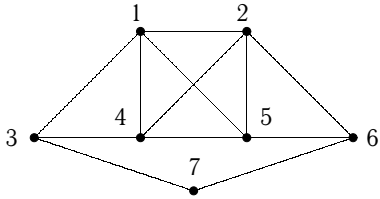


Рис. 7.34. Пример порождения клик графа

Для программы алгоритма 7.14 исходные данные структуры смежности $Adj[x]$ этого графа задаются в текстовом файле Clique.in. Структура (правило) заполнения файла совпадает с той, которая описана в примере поиска в глубину при расчете по программе алгоритма 7.2.

Данные файла Clique.in для примера на рис. 7.34:

```
7
3  3     1  4  7
1  4     3  4  5  2
4  4     3  1  2  5
7  2     3  6
2  4     1  4  5  6
5  4     4  1  2  6
6  3     7  5  2
```

Результаты расчетов сохраняются в выходном файле Clique.out со следующей структурой:

```
6  5  2
6  7
4  5  2  1
4  3  1
3  7
```

Строки файла Clique.out — это номера вершин соответствующих клик «симпатичного» графа на рис. 7.34. Отсюда видно, что в данном случае на вечер могут быть приглашены лишь четыре близких друга генерала.

## § 7.12. Циклы, фундаментальные множества циклов

В данном разделе рассматриваются алгоритмы решения задач, имеющих отношение к структуре циклов графа. Подобного рода задачи возникают при изучении вычислительных программ, в системах контроля при размыкании обратных связей и т. п. Рассмотрим остовное дерево $\Gamma_0(X, U_0, \Phi)$ графа $\Gamma(X, U, \Phi)$. Любое ребро, не принадлежащее $U_0$, т. е. любое ребро из $U \setminus U_0$,

порождает в точности один цикл при добавлении его к $U_0$. Такой цикл является элементом фундаментального множества циклов графа $\Gamma$ относительно дерева $\Gamma_0$. Так как каждое остовное дерево графа $\Gamma$ включает $|X|-1$ ребро, в фундаментальном множестве циклов относительно любого остовного дерева графа $\Gamma$ имеется $|U|-|X|+1$ циклов.

Полезность фундаментального множества циклов вытекает из того факта, что это множество полностью определяет циклическую структуру графа: каждый цикл в графе может быть представлен комбинацией циклов из фундаментального множества. Пусть $F=\{C_1, C_2, \ldots, C_{|U|-|X|+1}\}$ — фундаментальное множество циклов, где каждый цикл $C_i$ является подмножеством ребер $C_i \subseteq U$. Тогда любой цикл графа $\Gamma$ можно записать в виде $((\ldots(C_{i_1} \oplus C_{i_2}) \oplus \ldots) \oplus C_{i_r})$, где символ $\oplus$ обозначает операцию симметрической разности, $C_p \oplus C_q = \{u \,|\, u \in C_p \cup C_q \wedge u \notin C_p \cap C_q\}$.

Например, на рис. 7.35 показан граф и фундаментальное множество циклов, получающихся из выделенного жирной линией остовного дерева графа. Цикл графа $(x_1, x_2, x_5, x_3)$ есть $C_1 \oplus C_3 \oplus C_4$ или цикл $(x_1, x_2, x_4, x_3)$ есть $C_1 \oplus C_2$. Отметим, что в общем случае не каждая сумма циклов будет являться циклом графа.

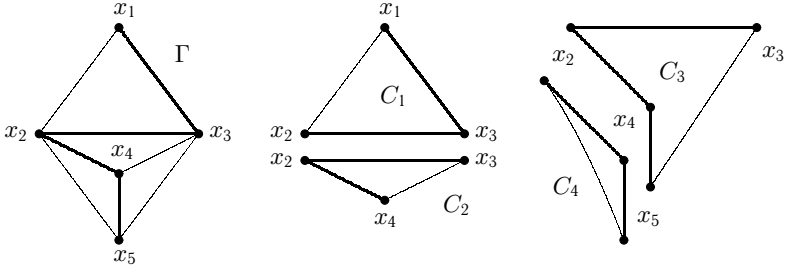



Рис. 7.35. Граф, его остовное дерево и фундаментальное множество циклов

При порождении фундаментального множества циклов удобно использовать метод поиска в глубину; он строит остовное дерево и каждое обратное ребро порождает цикл относительно этого дерева. Для того чтобы следить за ребрами дерева, используется поиск в глубину со стеком, в котором хранятся все текущие вершины пройденного пути в данный момент. Когда попадаем на обратное ребро, обнаруженный цикл будет состоять из этого ребра и ребер, соединяющих вершины из верха

стека. Реализация рассмотренного подхода представлена в алгоритме 7.15, который строит фундаментальное множество циклов $F = \{C_1, C_2, \ldots, C_{|U|-|X|+1}\}$ графа $\Gamma(X, U, \Phi)$.

Программная реализация поиска фундаментального множества циклов представлена в алгоритме 7.16.

Алгоритм 7.15.

```
for v ∈ X do Mark[v] = 0; { }
count = 0;
jC = 0; { }
nC = 0; { }
for v ∈ X do if Mark[v] = 0 then begin
   nC = nC + 1;
   C[nC] = v; { }
   Cycle(v, 0);
   nC = nC − 1;
end;
Procedure Cycle(x, y);
   count = count + 1;
   Mark[x] = count; { }
   for v ∈ Adj[x] do begin
      nC = nC + 1;
      C[nC] = v; { }
      if Mark[v] = 0 then Cycle(v, x)
      else if Mark[v] < Mark[x] ∧ v ≠ y then begin
         jC = jC + 1; { (x, v) }
         WriteCycle(v, C, nC); { }
      end;
      nC = nC − 1; { }
   end;
end;
Procedure WriteCycle(x, C, nC);
   print jC; { }
   repeat
      print C[nC]; { }
      nC = nC − 1;
   until C[nC] = x;
end.
```

Алгоритм 7.16.

```
Program GraphCycle; { }
uses CRT,DOS;
Const
  nVertex=100; Максимальное количество вершин
  nAdjacent=1000; Максимальная длина списка смежности
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeAdjacent=array[1..nAdjacent] of Integer;
Var
  f  :Text; { }
  n  :Integer; { }
  nC :Integer; {   }
  Adj :TypeAdjacent; {  }
  Fst :TypeVertex; {   }
  Nbr :TypeVertex; {   }
  Vtx :TypeVertex; {  }
  Mark :TypeVertex; {   }
  C  :TypeVertex; {   }
  B  :TypeVertex; {   }
  jC :Integer; {  }
  count:Integer; {  }

Procedure Init( var yes :Boolean );
{ }
{    }
{          }
Var
  i,j,m :Integer;
begin
  for i:=1 to n do
    for j:=1 to Nbr[i] do begin
      yes:=FALSE;
      for m:=1 to n do
        if Adj[Fst[i]+j]=Vtx[m] then begin
          yes:=TRUE;
          Adj[Fst[i]+j]:=m;
```

```
            break;
          end;
        if not yes then exit;
      end;
end;

Procedure PrintCycle( x:Integer; var C:TypeVertex; nC:Integer);
{  }
begin
  Write(f,jC,')');
  repeat
    Write(f,Vtx[C[nC]]:3);
    nC:=nC-1;
  until C[nC]=x;
  Writeln(f);
end;

Procedure Cycle( x,y:Integer );
Var
  i,v :Integer;
begin
  count:=count+1;
  Mark[x]:=count;
  for i:=1 to Nbr[x] do begin
    v:=Adj[Fst[x]+i];
    nC:=nC+1; C[nC]:=v;
    if Mark[v]=0 then Cycle(v,x)
    else if (Mark[v]<Mark[x]) and (v<>y) then begin
      {        }
      jC:=jC+1;
      PrintCycle(v,C,nC);
    end;
    nC:=nC-1;
  end;
end;

Procedure DepthCycle; {    }
Var
  v:Integer;
```

```
begin
  jC:=0; { }
  nC:=0; {          }
  count:=0; { }
  for v:=1 to n do Mark[v]:=0;
  for v:=1 to n do if Mark[v]=0 then begin
    nC:=nC+1; C[nC]:=v;
    Cycle(v,0);
    nC:=nC-1;
  end;
end;


Var {      }
  i,j :Integer;
  yes :Boolean;
begin {      }
  Assign(f,'Cycle.in');
  Reset(f); {  }
  { }
  Read(f,n); {  }
  Fst[1]:=0; {   }
  for i:=1 to n do begin
    Read(f,Vtx[i]); { }
    Read(f,Nbr[i]); {  }
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]); {
                                                    }
    Fst[i+1]:=Fst[i]+Nbr[i]; {
                                                     }
  end;
  Close(f);
  Assign(f,'Cycle.out');
  Rewrite(f); {  }
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа!');
    Close(f);
    exit;
  end;
```

```
  DepthCycle;
  Close(f);
end. {      }
```

Рассмотрим пример расчета по программе алгоритма 7.16 фундаментального множества циклов для графа, представленного на рис. 7.35. Исходные данные структуры смежности графа задаются в текстовом файле Cycle.in:

```
5
1  2      3  2
2  4      1  3  4  5
3  4      1  2  4  5
4  3      2  3  5
5  3      2  4  3
```

Результаты расчетов сохраняются в текстовом файле Cycle.out со следующей структурой:

```
1)  1  2  3
2)  3  4  2
3)  2  5  4
4)  3  5  4  2
```

Каждая строка в выходном файле — это найденный цикл, который представляется последовательностью вершин. Начальный номер в строке — номер цикла по порядку.

## § 7.13. Листы и блоки

В данном разделе рассматриваются вопросы, позволяющие глубже проникнуть в структуру произвольных графов, понять их устройство. Важное место в этом будут занимать такие понятия, как цикличность и связность. В завершении будут приведены практические алгоритмы выделения рассматриваемых здесь структур графа.

### 7.13.1. Листы

Определение. Пусть $\Gamma(X, U, \Phi)$ — неориентированный граф. Ребро $u = (a, b) \in U$ называется *циклическим ребром*, если оно принадлежит некоторому циклу. Петля является циклическим ребром. Никакое концевое ребро (инцидентное висячей вершине) не может быть циклическим. Например, дерево не имеет циклических ребер, и, обратно, связный граф без циклических ребер является деревом.

Определение. Ребро $u = (x, y) \in U$ называется *разделяющим ребром* (или мостом, или разрезающим ребром) в $\Gamma$, если в графе $\widetilde{\Gamma}$, получающимся после удаления ребра $u$, вершины $x$ и $y$ не связаны. Тогда граф $\widetilde{\Gamma}$ можно представить как объединение графов

$$\widetilde{\Gamma} = \Gamma_1 \cup \Gamma_2, \tag{7.3}$$

где $\Gamma_1 \cap \Gamma_2 = \varnothing$ и $\Gamma_1$ содержит $x$, $\Gamma_2$ содержит $y$.

Утверждение 7.13.1. *Ребро $u = (x, y) \in U$ является разделяющим тогда и только тогда, когда оно не является циклическим.*

Доказательство. ($\Rightarrow$) Допустим, что разделяющее ребро $u$ является циклическим. Тогда после его удаления вершины $x$ и $y$ останутся связанными, а значит, разложение 7.3 невозможно, что противоречит условию: $u$ — разделяющее ребро.

($\Leftarrow$) Теперь $u = (x, y)$ — не циклическое ребро, т. е. не существует цепей, соединяющих $x$ и $y$ и не содержащих $u$. Отсюда $u$ — разделяющее ребро. □

Определение. Будем говорить, что две вершины $x_0$ и $x_n$ *циклически-реберно связаны*, если существует такая последовательность простых циклов $C_1, C_2, \ldots, C_k$, что $x_0 \in C_1, x_n \in C_k$ и каждая пара соседних циклов имеет хотя бы одну общую вершину. Условно взаимное расположение вершин и циклов можно представлять так, как показано на рис. 7.36.

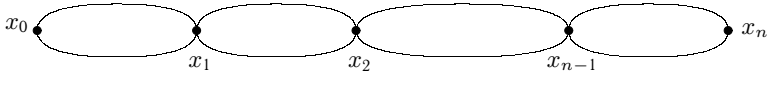



Рис. 7.36. Вершины связаны циклически-реберно

Утверждение 7.13.2. *Циклически-реберная связность определяет отношение эквивалентности* (*см. п.* 7.4) *на множестве вершин графа* $\Gamma(X, U, \Phi)$.

На рис. 7.37 показано разбиение $\Gamma$ на компоненты циклически-реберной связности.

Определение. Множество всех вершин, циклически-реберно связанных с данной вершиной $x$, называется *листовым множеством* $L(x)$, которому принадлежит $x$. Отметим, что листовое множество может состоять только из одной вершины $x$, тогда листовое множество называется *особым*. Далее будет по-

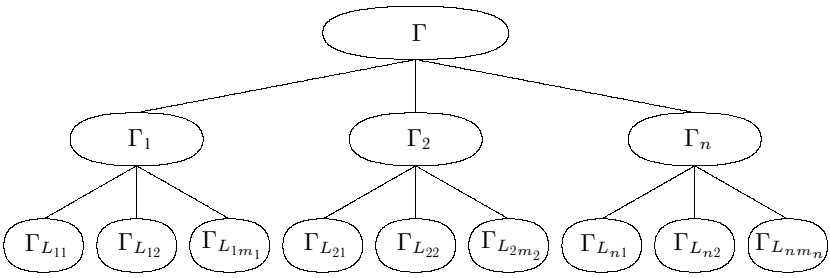



Рис. 7.37. Компоненты $\Gamma(L_{ij})$ циклически-реберной связности графа. Компоненты $\Gamma(L_k)$ связности графа. Компоненты $\Gamma(L_{ij}) \subseteq \Gamma(L_i)$ связаны мостами

казано, что это возможно тогда и только тогда, когда все ребра, инцидентные вершине $x$, являются петлями или разделяющими ребрами.

Определение. Подграф $\Gamma(L)$, определяемый листовым множеством $L$, называется *листом*.

Утверждение 7.13.3. *Отметим следующие очевидные свойства листа* $\Gamma(L)$.

1. *Граф* $\Gamma(L)$ *циклически замкнут, т.е. если какой-нибудь простой цикл* $C$ *в* $\Gamma$ *имеет общую вершину с* $L$, *то весь простой цикл* $C$ *содержится в* $\Gamma(L)$.
2. *Не может быть более одного ребра, связывающего два различных листовых множества* $L_1$ *и* $L_2$, *так как иначе эти ребра оказались бы циклическими и множества* $L_1$ *и* $L_2$ *должны были бы совпадать. Связывающие одиночные ребра различные листовые множества* $L_1$ *и* $L_2$ *являются мостами для графа. Ниже показано, что они будут являться и блоками, состоящими из одного ребра.*
3. *Все ребра в* $\Gamma(L)$ *являются циклическими. Пусть ребро* $(x, y)$ *лежит в* $\Gamma(L)$. *Покажем, что оно циклическое. Действительно, если, например, вершины* $x$ *и* $y$ *в* $L$ *соединены ребром, тогда* $(x, y)$ *циклическое по построению* $L$. *Если же* $x$ *и* $y$ *в* $L$ *не соединены ребром, то, добавив данное ребро* $(x, y)$ *к* $L$, *получим цикл, так как вершины* $x, y$ *связаны в* $L$ *по построению. Отсюда следует, что ребро* $(x, y)$ *циклическое.*
4. *По определению граф* $\Gamma(L)$ *связный. Из предыдущего пункта следует, что маршрут в* $\Gamma(L)$, *соединяющий произвольные две его вершины, будет состоять исключительно из циклических ребер.*

### 7.13.2. Блоки

Определение. Пусть $\Gamma(X, U, \Phi)$ — неориентированный граф. Два ребра $u, v \in U$ называются *сильно циклически связанными*, если существует такая последовательность простых циклов $C_1, C_2, \ldots, C_k$, что $u \in C_1$, $v \in C_k$ и любая пара соседних циклов $C_i, C_{i+1}$ имеет, по крайней мере, одно общее ребро. Условно взаимное расположение ребер и циклов можно представлять так, как показано на рис. 7.38.

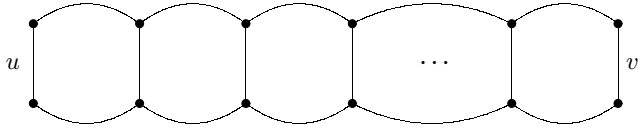



Рис. 7.38. Сильно циклически связанные ребра $u, v$

Определение. Множество всех ребер, сильно циклически связанных с ребром $u \in U$, образует некоторую часть графа $\Gamma(L^b)$, называемую *блоком*, определяемым ребром $u$. Множество вершин $L^b$ этого графа называется *блоковым множеством*, определяемым ребром $u$. Блок является связным графом и может состоять из единственного ребра.

Лемма 7.13.1. *Два ребра сильно циклически связаны тогда и только тогда, когда существует простой цикл, содержащий оба эти ребра. Справедливость данного утверждения следует непосредственно из структуры сильно циклической связанности ребер* (*рис.* 7.38).

Лемма 7.13.2. *Если $P(x, y)$ — простая цепь, связывающая две различные вершины $x, y \in L^b$ блокового множества, то все ребра цепи $P(x, y)$ принадлежат блоку $\Gamma(L^b)$.*

Доказательство. Если предположить, что утверждение леммы не выполняется, то существует участок цепи $P(x, y)$, ребра которого не принадлежат данному блоку $\Gamma(L^b)$. Не уменьшая общности, будем считать, что этим участком является исходная цепь $P(x, y)$. Так как $x, y \in L^b$, то в блоке $\Gamma(L^b)$ существует простая цепь $Q(x, y)$, связывающая $x, y$. Тогда объединение простых цепей $P(x, y \cup Q(x, y))$ составит простой цикл. Согласно лемме 7.13.1 ребра этого цикла будут сильно циклически связаны, а значит, все ребра простой цепи $P(x, y)$ должны лежать в $\Gamma(L^b)$, что противоречит предположению. □

Утверждение 7.13.4. *Из леммы* 7.13.2 *следует, что блок* $\Gamma(L^b)$ *является подграфом. Он циклически сильно замкнут, в том смысле, что если простой цикл* $C$ *имеет хотя бы две вершины, общие с* $L^b$, *то все ребра цепи* $C$ *принадлежат* $\Gamma(L^b)$. *Поэтому два различных блока могут иметь не более одной общей вершины* (*рис.* 7.39, 7.41). *В противном случае блоки должны совпадать.*

Утверждение 7.13.5. *Из определения блокового множества* $L^b$ *следует, что все его вершины являются циклически-реберно связанными при условии, что число ребер в блоке* $\Gamma(L^b)$ *более одного. Отсюда заключаем, что* $L^b \subseteq L$, *где* $L$ — *листовое множество вершин* $L^b$, *и каждый лист* $\Gamma(L)$ *имеет непересекающееся по ребрам разложение на семейство блоков*

$$\Gamma(L) = \bigcup_i \Gamma(L_i^b). \tag{7.4}$$

Определение. В графе $\Gamma(X, U, \Phi)$ вершину $x \in X$ называют *разделяющей вершиной* (разрезающей или точкой сочленения), если имеет место прямое по ребрам разложение (рис. 7.41)

$$\Gamma(L) = \Gamma(X_1) \cup \Gamma(X_2),\ X = X_1 \cup X_2,\ X_1 \cap X_2 = x. \tag{7.5}$$

Утверждение 7.13.6. *Блок* $\Gamma(L^b)$ *не имеет разделяющих вершин, так как все его вершины циклически-реберно связаны.*

На основании разложения 7.4 любого листа $\Gamma(L)$ на непересекающееся по ребрам разложение на блоки $\Gamma(L^b)$ и последнего утверждения составление листа $\Gamma(L)$ из блоков $\Gamma(L^b)$ может быть представлено кактусообразной фигурой, как на рис. 7.39, в которой различные блоки соприкасаются в своих соединяющих вершинах. Соединяющие вершины блоков являются разделяющими вершинами графа. Разложение 7.4 листов на блоки $\Gamma(L_{ij}) = \bigcup_k \Gamma(L_{ijk}^b)$ позволяет теперь уточнить структуру графа (рис. 7.37): каждый лист $\Gamma_{L_{ij}}$ можно представить в виде кактусообразной схемы (рис. 7.39) составляющих его блоков $\Gamma(L_{ijk}^b)$.

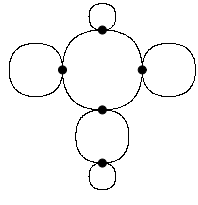


Рис. 7.39

### 7.13.3. Поиск блоков в глубину

Разложение графа на блоки и выделение их имеет важное практическое значение. Иногда недостаточно знать, что граф связен; может интересовать насколько «сильно связен» связный граф.

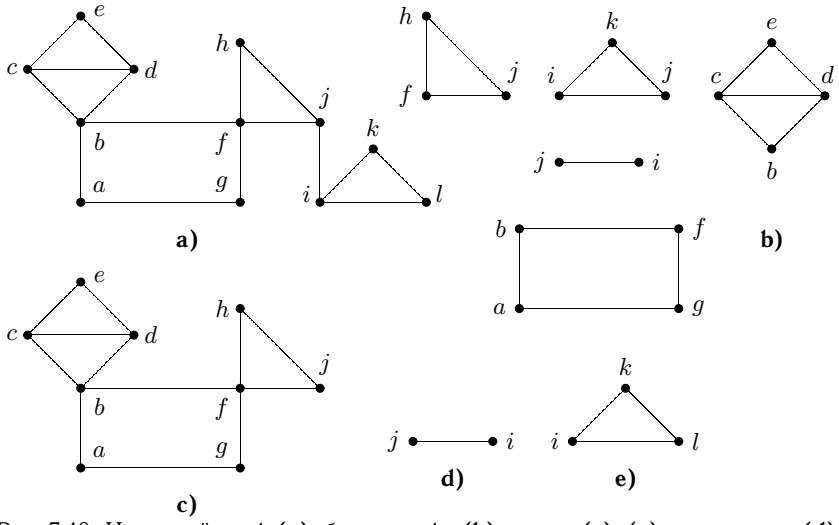



Рис. 7.40. Исходный граф **(a)**; блоки графа **(b)**; листы **(c)**, **(e)** и один мост **(d)**; точки сочленения: *b*, *f*, *j*, *i*

Например, удаляя вершину сочленения двух блоков (рис. 7.41) и инцидентных ей ребер, нарушим связность графа. Связность графа не нарушится, если удалить внутреннюю вершину блока и инцидентные ей ребра. Про такие компоненты графа говорят, что они *двусвязные*. Отыскание точек сочленения и блоков графа важно при изучении надежности коммуникационных и транспортных сетей. Это важно также и при изучении других свойств графа, таких, например, как планарность, так как часто полезно разложить граф на его двусвязные компоненты (блоки) и изучать каждую из них в отдельности.

На рис. 7.40 в качестве примера изображен связный граф и его блоки (двусвязные компоненты). Здесь же показано разложение графа на листы.

Для нахождения блоков и точек сочленения применим известный метод поиска в глубину на неориентированном графе $\Gamma(X, U, \Phi)$. Основу алгоритма выделения блоков и точек их сочленения легче понять, рассмотрев пример на рис. 7.41, где схематически изображен связный граф, состоящий из ше-

сти блоков $\Gamma_i$, $i = 1, 2, \ldots, 6$, и четырех точек сочленения $x_j$, $j = 1, 2, 3, 4$.

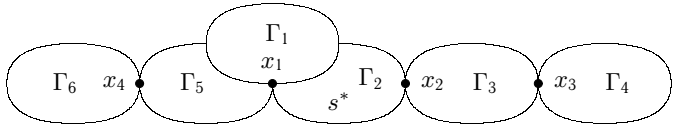



Рис. 7.41. Схематическое изображение графа с шестью блоками и четырьмя точками сочленения

Если начать поиск в глубину, скажем, из вершины $s \in \Gamma_2$, то можем перейти из $\Gamma_2$ в $\Gamma_1$, проходя через вершину $x_1$. Из свойства поиска в глубину, все ребра в $\Gamma_1$ должны быть пройдены до того, как вернемся в $x_1$. Поэтому $\Gamma_1$ состоит в точности из ребер, которые проходятся между входом и выходом из вершины $x_1$. Для других блоков дело обстоит несколько иначе. Так, например, если уйти из $\Gamma_2$ в $\Gamma_5$, а оттуда — в $\Gamma_6$ через $x_4$, то окажемся в $\Gamma_6$, пройдя ребра из $\Gamma_2$, $\Gamma_5$ и $\Gamma_6$. Если хранить ребра в стеке, то во время прохождения назад из $\Gamma_6$ в $\Gamma_5$ через вершину $x_4$ все ребра $\Gamma_6$ будут наверху стека. После их удаления наверху стека останутся ребра $\Gamma_5$ и снова продолжим проход блока $\Gamma_5$. Таким образом, если можно распознавать точки сочленения во время обхода, то применяя поиск в глубину и храня ребра в стеке в той очередности, в которой они проходятся, можно также определить и блоки (двусвязные компоненты). Ребра, находящиеся на верху стека в момент обратного прохода через точку сочленения, составляют блок.

Реализация рассмотренного подхода выделения блоков графа представлена в алгоритме 7.17. В векторе $Mark[x]$ меток вершин $x \in X$ графа фиксируются порядковые номера обхода соответствующих вершин. В этом случае метка вершины сочленения при входе в блок получит наименьший номер из всех остальных вершин блока. Таким образом, для определения точки сочленения выхода из блока достаточно найти вершину этого блока с минимальной меткой. Свойство циклической замкнутости блока позволяет это сделать лишь на основе вектора меток $Mark[x]$, не привлекая дополнительной информации. На рис. 7.42 схематично представлен блок, где вершина $w$ — точка входа в блок и точка сочленения. Вершина $x$ — следующая обследованная вершина блока после $w$.

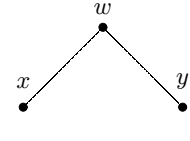



Рис. 7.42

Свойство циклической замкнутости блока позволяет утверждать, что существует по крайней мере еще одна вершина $y$, смежная $w$. Рекурсивный характер алгоритма поиска в глубину на графе позволяет утверждать, что в этом случае ребро блока $(y, w)$ будет пройдено как обратное, и, значит, метка $Mark[w]$ точки сочленения блока будет доступна до выхода из блока по ребру входа $(w, x)$. Поэтому, сохраняя минимальное значение меток обследованных вершин (включая и по обратным ребрам поиска в глубину), будем проверять при рекурсивном выходе из поиска в глубину, совпадает ли это минимальное значение с меткой вершины выхода. Если ответ положительный, то найдена точка сочленения. В алгоритме значения минимальных меток фиксируются в векторе $Virtual[x]$ для каждой вершины $x \in X$, как наименьшее значение из $Mark[y]$, где $y$ — вершины графа, которые достижимы из $x$ при проходе графа в глубину.

Алгоритм 7.17.

$count = 0$;
$Stack = \varnothing$;
for $v \in X$ do $Mark[v] = 0$; { }
for $v \in X$ do if $Mark[v] = 0$ then $Block(v, 0)$;
Procedure $Block(x, w)$
    $count = count + 1$;
    $Mark[x] = count$; { }
    $Virtual[x] = count$; { }
    for $v \in Adj[x]$ do begin
        if $Mark[v] = 0$ then begin
            $Stack = Stack \cup \{(x, v)\}$;
            $Block(v, x)$;
            $Virtual[x] = \min(Virtual[x], Virtual[v])$;
            if $Virtual[v] \geqslant Mark[x]$ begin
                $x$

                $(x, v)$
            end;
        end
        else if $Mark[v] < Mark[x]$ and $v \neq w$ then begin
            $(x, v)$ — обратное ребро
            $Stack = Stack \cup \{(x, v)\}$;
            $Virtual[x] = \min(Virtual[x], Mark[v])$;

```
        end
    end;
end.
```

## § 7.14. Двудольные графы

Определение. Граф $\Gamma(X, U, \Phi)$ называется *двудольным*, если множество его вершин можно разбить на два независимых подмножества $V_1, V_2$ таких, что $X = V_1 \cup V_2$ и $V_1 \cap V_2 = \varnothing$. Такой граф будем обозначать как $\Gamma(V_1 \cup V_2, U, \Phi)$.

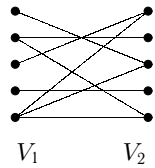


Рис. 7.43

На рис. 7.43 изображен типичный двудольный граф. Двудольные графы играют заметную роль в различных приложениях.

### 7.14.1. Условия существования двудольных графов

Теорема 7.14.1. *Граф* $\Gamma(X, U, \Phi)$ *является двудольным тогда и только тогда, когда любой его простой цикл имеет четную длину.*

Доказательство. ($\Rightarrow$) Ясно, что данное условие для двудольного графа всегда выполняется.

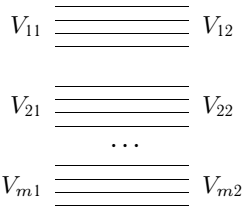


Рис. 7.44

($\Leftarrow$) Разобьем граф $\Gamma$ на компоненты связности $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$. Пусть $\Gamma_k(X_k, U_k, \Phi)$ одна из них и $a \in X_k$ — произвольная вершина. Разобьем множество вершин $X_k$ на непересекающиеся $V_{k_1}$ и $V_{k_2}$, где $V_{k_1}$ — вершины, расстояние до которых от $a$ нечетно; $V_{k_2}$ — вершины, расстояние до которых от $a$ четно, $a \in V_{k_2}$. Множества $V_{k_1}$ и $V_{k_2}$ являются независимыми. Действительно, если предположить, что вершины $b$ и $c$ — смежные и $b, c \in V_{k_1}$ или $b, c \in V_{k_2}$, то существовал бы цикл нечетной длины, соответственно ($a$ — *нечетная длина* — $b$ — *длина единица* — $c$ — *нечетная длина* — $a$) или ($a$ — *четная длина* — $b$ — *длина единица* — $c$ — *четная длина* — $a$), что противоречит условию. Рассмотренное разбиение вершин выполним для каждой компоненты $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$ связности. На рис. 7.44 показано объединение независимых компонент $V_{k_1}$ и $V_{k_2}$ в независимые компоненты $V_1 = \bigcup\limits_{k_1} V_{k1}$ и $V_2 = \bigcup\limits_{k_2} V_{k2}$. □

### 7.14.2. Паросочетания

Определение. *Паросочетанием* в двудольном графе $\Gamma(V_1 \cup V_2, U, \Phi)$ называется независимое подмножество ребер $\pi \subseteq U$, ребра $\pi$ не имеют общих вершин.

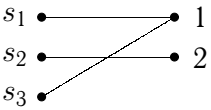


Рис. 7.45

Для графа, изображенного на рис. 7.45, в качестве паросочетания можно взять множество ребер $\pi_1 = \{(s_1, 1), (s_2, 2)\}$, $\pi_2 = \{(s_3, 1), (s_2, 2)\}$ и т. п.

Определение. Паросочетание называется *максимальным*, если любое другое паросочетание содержит меньшее число ребер.

### 7.14.3. Алгоритм определения максимального паросочетания

Пусть $\Gamma(V_1 \cup V_2, U, \Phi)$ — двудольный граф и $\pi$ — произвольное паросочетание. Множество вершин $A_1 \subseteq V_1$ и $A_2 \subseteq V_2$ (рис. 7.46). *Чередующейся цепью* относительно паросочетания $\pi$ называется цепь $C$, в которой ребра поочередно принадлежат $U$ и $\pi$ и которая начинается ребром, не принадлежащим $\pi$. Заметим, что ребра в цепи не повторяются. Пусть $a_1 \in A_1$ — произвольная вершина и $u_1 = (a_1, v_{2\pi})$ — ребро, инцидентное вершинам из $A_1$ и $V_{2\pi}$. Построим чередующуюся цепь $C = (u_1\pi_1 u_2\pi_2 \dots u_{n-1}\pi_{n-1}u_n)$ и допустим, что по такой цепи можно достичь вершины $a_2 \in A_2$. Такую цепь $C$ можно использовать для получения нового паросочетания $\widetilde{\pi}$, которое содержит на одно ребро больше, чем исходное паросочетание $\pi$. Действительно, в $\widetilde{\pi}$ можно включить все ребра $\pi$, удалив $\pi_1, \pi_2, \dots, \pi_{n-1}$ и добавив $u_1, u_2, \dots, u_n$. Полученное таким образом $\widetilde{\pi}$ содержит несмежные ребра, а значит, $\widetilde{\pi}$ — паросочетание. Говорят, что $\widetilde{\pi}$ получено из $\pi$ чередующимся расширением.

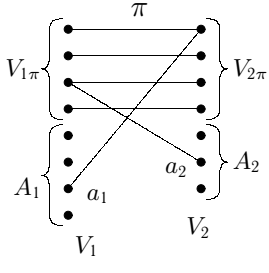


Рис. 7.46

Пример. Дан двудольный граф $\Gamma(V_1 \cup V_2, U, \Phi)$, где $V_1 = \{s_1, s_2, \dots, s_6\}$ и $V_2 = \{1, 2, \dots, 6\}$. Соединение вершин $V_1$ и $V_2$ задано соотношениями: $s_1 \to \{4\}$, $s_2 \to \{1, 6\}$, $s_3 \to \{4, 5, 6\}$, $s_4 \to \{1, 3\}$, $s_5 \to \{2\}$, $s_6 \to \{1, 3\}$. На рис. 7.47 дано его графическое представление. Найти максимальное паросочетание.

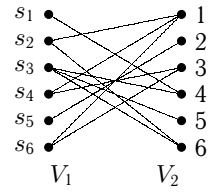


Рис. 7.47

Выберем начальное паросочетание $\pi$ таким образом, что вершину $s_i \in V_1$ соединяем с вершиной $j \in V_2$ — первой незанятой ранее из списка соединений для $s_i$. Тогда начальное паросочетание будет равно $\pi = \{(s_1, 4), (s_2, 1), (s_3, 5), (s_4, 3), (s_5, 2)\}$, $|\pi| = 5$. Вершины $s_6$ и 6 не вошли в паросочетание. Попытаемся увеличить $\pi$, используя алгоритм чередующихся цепей. Обозначим $\xrightarrow{u}$ переход по ребрам графа из $V_1$ в $V_2$ и $\xrightarrow{\pi}$ переход по ребрам паросочетания из $V_2$ в $V_1$. Так, $s_6 \xrightarrow{u} \{1, 3\}$ означает, что из $s_6$ можно перейти в вершины 1 и 3, и $\{1, 3\} \xrightarrow{\pi} \{s_2, s_4\}$ означает, что из вершин 1 и 3 можно достичь по ребрам паросочетания вершин $s_2$ и $s_4$. По алгоритму исходной вершиной цепи является $s_6$. Тогда множество всех чередующихся цепей, начало которых в $s_6$, можно представить так: $s_6 \xrightarrow{u} \{1, 3\} \xrightarrow{\pi} \{s_2, s_4\} \xrightarrow{u} \{1, 3, 6\}$. Переходы следует закончить, если вершина 6 достигнута или подмножество вершин $V_2$, доступных из $s_6$, повторилось в чередующейся цепи. В последнем случае вершина 6 недоступна из $s_6$ и, значит, исходное паросочетание $\pi$ максимальное. В нашем случае вершина 6 оказалась доступной. Для выделения исходной чередующейся цепи из всего множества расширяющихся цепей выполним обратный ход: $6 \xrightarrow{u} s_2 \xrightarrow{\pi} 1 \xrightarrow{u} s_6$. Теперь новое паросочетание строим из старого, исключая из него ребро $(s_2, 1)$ и включая ребра $(6, s_2), (1, s_6)$. Максимальное паросочетание содержит ребра $(s_1, 4)$,$(s_2, 6)$,$(s_3, 5)$,$(s_4, 3)$,$(s_5, 2)$,$(s_6, 1)$ и $|\pi| = 6$.

Теорема 7.14.2 (о максимальном паросочетании). *Пусть* $\Gamma(V_1 \cup V_2, U, \Phi)$ — *двудольный граф. Тогда количество ребер в максимальном паросочетании* $\pi(\Gamma)$ *равно*

$$|\pi(\Gamma)| = |V_1| - \max_{A \subseteq V_1}(\,|A| - |\triangle A|\,), \tag{7.6}$$

*где* $\triangle A = \{\, y \in V_2 \,|\, \exists x \in A \subseteq V_1 \wedge (x, y) \in U\} \subseteq V_2$.

Доказательство. Пусть $\pi(\Gamma)$ — максимальное паросочетание в исходном двудольном графе (рис. 7.48). Паросочетание $\pi$ позволяет рассматривать его как отображение $\pi : V_2 \xrightarrow{\pi} V1$ вершин множества $V_2$ в вершины множества $V_1$ по ребрам паросочетания $\pi$. Аналогично, под $\triangle$ будем понимать отображение

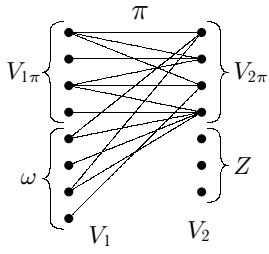



Рис. 7.48. Максимальное паросочетание $\pi$

$\triangle\colon V_1 \xrightarrow{\triangle} V_2$ вершин множества $V_1$ в вершины множества $V_2$ по ребрам $U$ графа. Вершины множеств $\omega$ и $Z$ (рис. 7.48) несмежные, так как паросочетание является максимальным.

Рассмотрим процесс насыщения множества $\omega$. Под $\omega \to {}_{\triangle}\omega_{\pi}$ будем понимать последовательное выполнение двух отображений $\pi(\triangle(\omega)) \subseteq V_1$ множества $\omega$. Далее применим отображения $\triangle$ и $\pi$ к полученному множеству: ${}_{\triangle}\omega_{\pi} \to {}_{\triangle}({}_{\triangle}\omega_{\pi})_{\pi} \subseteq V_1$ и т. д. Ясно, что каждая пара отображений приводит к расширению исходного множества ${}_{\triangle}\omega_{\pi} \subseteq {}_{\triangle}({}_{\triangle}\omega_{\pi})_{\pi}$. Учитывая конечные размеры исходного графа, наступит момент, когда отображаемое множество перестанет расширяться. Таким образом, процесс насыщения отображаемого множества можно представить следующим образом:

$$\omega \to {}_{\triangle}\omega_{\pi} \to {}_{\triangle}({}_{\triangle}\omega_{\pi})_{\pi} \to \ldots \to {}_{\triangle}(\ldots {}_{\triangle}({}_{\triangle}\omega_{\pi})_{\pi}\ldots)_{\pi} = A$$

Для множества $A$ выполняется условие $|A| = |\triangle A|$, в противном случае множество $A$ можно было бы еще расширить. Следствием расширения множества $\triangle\omega$ в процессе его отображения является включение $\triangle\omega \subseteq \triangle A$.

Обозначим множество $\omega \cup A = B$. Покажем, что найденное множество $B$ удовлетворяет условию (7.6) теоремы, т. е. $\pi(\Gamma) = |V_1| - (|B| - |\triangle B|)$. Имеем $|\triangle B| = |\triangle(\omega \cup A)| = |\triangle\omega \cup \triangle A| = |\triangle A|$, следовательно, $|V_1| - (|B| - |\triangle B|) = |V_1| - (|\omega| + |A| - |\triangle A|) = |V_1| - |\omega| = |\pi(\Gamma)|$.

Покажем теперь, что $\forall A \subseteq V_1$ условие (7.6) теоремы записывается в следующем виде: $|\pi(\Gamma)| \leqslant |V_1| - (|A| - |\triangle A|)$. Тогда ранее найденное множество $B$, для которого данное неравенство обращается в равенство, будет доказательством условия (7.6) теоремы. Пусть $A_1 = A \setminus (\omega \cap A)$, тогда $A = (\omega \cap A) \cup A_1$. Отметим, что $\forall A_1 \subseteq V_1 \setminus (\omega \cap V_1)$ выполняется неравенство $|A_1| \leqslant |\triangle A_1|$, так как вершины множества $V_1 \setminus (\omega \cap V_1)$ составляют паросочетание. Тогда верно, что $|A| = |\omega \cap A| + |A_1| \leqslant |\omega| + |A_1| \leqslant |\omega| + |\triangle A_1| \leqslant |\omega| + |\triangle A|$ или $|A| - |\triangle A| \leqslant |\omega|$. Теперь количество ребер в максимальном паросочетании можно оценить: $|\pi(\Gamma)| = |V_1| - |\omega| \leqslant |V_1| - (|A| - |\triangle A|)$. □

Следствие (обоснование справедливости алгоритма определения максимального паросочетания). *Пусть $\widetilde{\pi}$ — произвольное паросочетание, относительно которого ни одна из че-*

*редующихся цепей с началом в вершинах множества $\omega$ не достигает вершин множества $Z$.*

Именно это свойство максимального паросочетания использовалось при доказательстве формулы числа ребер в максимальном паросочетании (7.6). Следовательно, количество ребер и в $\widetilde{\pi}$ определяется по этой же формуле (7.6), а значит, паросочетание $\widetilde{\pi}$ является максимальным.

### 7.14.4. Системы различных представителей

Рассмотрим пять множеств: $S_1 = \{2, 3\}$, $S_2 = \{1, 2, 4\}$, $S_3 = \{1, 2, 5\}$, $S_4 = \{3, 4, 5\}$, $S_5 = \{3, 4, 5\}$. Требуется выбрать такие различные числа $x_1, x_2, x_3, x_4, x_5$, что $x_i \in S_i$, $i = 1, 2, 3, 4, 5$. Для данного примера $x_1 = 2$, $x_2 = 1$, $x_3 = 5$, $x_4 = 3$, $x_5 = 4$. Однако если взять множества $T_1 = \{1, 2\}$, $T_2 = \{1, 2\}$, $T_3 = \{1, 2\}$, $T_4 = \{3, 4, 5\}$, $T_5 = \{3, 4, 5\}$, то такой выбор оказывается невозможным.

Рассмотрим данную задачу в общем случае. Пусть $S = \{1, 2, \ldots, n\}$. $M(S)$ — система подмножеств $S_1, S_2, \ldots, S_n$ множества $S$. Будем говорить, что $M(S)$ имеет систему различных представителей, если для всех $i = 1, 2, \ldots, n$ существуют различные $x_i \in S_i$.

### 7.14.5. Связь системы различных представителей и двудольных графов

Определим двудольный граф $\Gamma(V_1 \cup V_2, U, \Phi)$, соответствующий системе подмножеств. Пусть $M(S) = \{S_1, S_2, \ldots, S_n\}$ — система подмножеств множества $S$. Положим $S_1, S_2, \ldots, S_n$ — вершины $V_1$ графа, которые соединены ребрами со своими элементами — смежными им вершинами из $V_2$ (рис. 7.49).

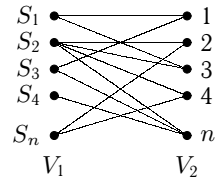


Рис. 7.49

Лемма 7.14.1. *Двудольный граф $\Gamma(V_1 \cup V_2, U, \Phi)$, отвечающий системе подмножеств $M(S) = \{S_1, S_2, \ldots, S_n\}$, имеет максимальное паросочетание из $n$ ребер тогда и только тогда, когда $M(S)$ имеет систему различных представителей.*

Доказательство вытекает из определения построения двудольного графа, отвечающего системе различных представителей).

Теорема 7.14.3 (Ф. Холла о существовании системы различных представителей). *Система $M(S) = \{S_1, S_2, \ldots, S_n\}$ имеет систему различных представителей тогда и только тогда, когда для любой подси-*

*стемы* $\{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\} \subseteq M(S)$ *выполняется неравенство* $\left|\bigcup\limits_{j=1}^{k} S_{i_j}\right| \geqslant k$, *т.е. количество элементов в объединении любых* $k$ *подмножеств должно быть не менее* $k$.

Доказательство. ($\Rightarrow$) Необходимое условие теоремы следует из определения системы различных представителей. Каждое подмножество $S_{i_j}$ содержит элемент $x_{i_j} \in S_{i_j}$, отличный от элементов других подмножеств, а значит, $\left|\bigcup\limits_{j=1}^{k} S_{i_j}\right| \geqslant k$.

($\Leftarrow$) Пусть $\Gamma(V_1 \cup V_2, U, \Phi)$ — двудольный граф, соответствующий системе подмножеств $M(S) = \{S_1, S_2, \ldots, S_n\}$. Покажем, что в данном графе существует максимальное паросочетание, количество ребер в котором равно $n$. Тогда из леммы (7.14.1) будет следовать достаточное условие теоремы. Из теоремы (7.14.2) имеем, что число ребер в максимальном паросочетании равно $|\pi(\Gamma)| = |V_1| - \max\limits_{A \subseteq V_1}(\,|A| - |\triangle A|\,)$, где $\triangle A = \{y \in V_2 \,|\, \exists x \in A \ \wedge (x, y) \in U\} \subseteq V_2$. В рамках принятой интерпретации $A = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$, $|A| = k$ и $\triangle A = \bigcup\limits_{j=1}^{k} S_{i_j}$. По условию теоремы $|\triangle A| \geqslant k$. Таким образом, $\forall A \subseteq V_1$ $|A| - |\triangle A| \leqslant 0$, а значит, $|\pi(\Gamma)| = |V_1| - \max\limits_{A \subseteq V_1}(\,|A| - |\triangle A|\,) \geqslant |V_1|$. Однако $|\pi(\Gamma)| \leqslant |V_1|$, следовательно, $|\pi(\Gamma)| = |V_1| = n$. Достаточное условие доказано. □

### 7.14.6. Задача о назначениях

Существует ряд задач, постановки которых укладываются в рамки задачи о назначениях. Рассмотрим две такие постановки.

Задача. Предположим, что вычислительная сеть объединяет $n$ специализированных ЭВМ. Известно, что наибольшая производительность конкретной ЭВМ сети достигается на определенном классе задач. Это выражается коэффициентом $a_{ij}$ использования $i$-й ЭВМ при решении $j$-го класса задач. Найти оптимальное распределение задач по сети таким образом, чтобы эффективность ее использования была наибольшей.

Задача. Группа лиц может выполнить $n$ видов работ. Эффективность использования $i$-го лица на $j$-й работе определяется мерой ценности $a_{ij}$. Найти оптимальную расстановку людей по видам работ.

Теорема 7.14.4 (алгоритм поиска оптимальной перестановки $\pi$). *Пусть* $A = [a_{ij}]$, $i, j = 1, 2, \ldots, n$, — *матрица целых чисел. Тогда максимум*

$$\max_{\pi} \sum_{i=1}^{n} a_{i\pi_i} \tag{7.7}$$

*по всем перестановкам* $\pi$ *равен минимуму*

$$\min_{u_i + v_j \geqslant a_{ij}} \Big( \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j \Big) \tag{7.8}$$

*по всем числам* $u_i$ *и* $v_j$ *таким, что*

$$u_i + v_j \geqslant a_{ij} \quad \forall i, j = \overline{1, n}. \tag{7.9}$$

*Минимум суммы* (7.8) *достигается на перестановке* $\pi$ *такой, что*

$$u_i + v_{\pi_i} = a_{i\pi_i} \quad \forall i = \overline{1, n}.$$

Доказательство. Пусть $\pi$ — произвольная перестановка. При изменении $i$ от 1 до $n$ величины $\pi_i$ принимают все значения множества $\{1, 2, \ldots, n\}$. По условию $u_i + v_j \geqslant a_{ij}$ $\forall i, j = \overline{1, n}$, а значит, и для $j = \pi_i$ верно $u_i + v_{\pi_i} \geqslant a_{i\pi_i}$. Установим связь сумм (7.7) и (7.8):

$$\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j = \sum_{i=1}^{n} u_i + \sum_{i=1}^{n} v_{\pi_i} = \sum_{i=1}^{n} (u_i + v_{\pi_i}) \geqslant \sum_{i=1}^{n} a_{i\pi_i}.$$

Таким образом, сумма (7.7) для любой перестановки $\pi$ не больше суммы (7.8). Отсюда следует, что теорема будет верна, если мы найдем такие $u_i$ и $v_j$ и перестановку $\pi$, что (7.7) и (7.8) совпадут. В алгоритме определяется перестановка $\pi$ и величины $u_i, v_j$, для которых $u_i + v_{\pi_i} = a_{i\pi_i}$, $i = \overline{1, n}$, что доказывает теорему.

*Шаг 0.* Поиск начальных $u_i$ и $v_j$, удовлетворяющих ограничениям $u_i + v_j \geqslant a_{ij}$ $\forall i, j = \overline{1, n}$. Положим $v_j = 0$ и $u_i = \max\limits_j a_{ij}$ — максимальный элемент в $i$-й строке; условие (7.9) $u_i + v_j = \max\limits_j a_{ij} \geqslant a_{ij}$ выполняется. Для матрицы ценностей на рис. 7.50 справа и снизу указаны начальные значения, соответственно $u_i$ и $v_j$.

*Шаг 1.* Фиксируем все $v_j$, $j = \overline{1, n}$, и уменьшаем значения $u_i$, $i = \overline{1, n}$, до тех пор, пока одно из неравенств $u_i + v_j \geqslant a_{ij}$ не

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | |
|---|---|---|---|---|---|
| $u_1$ | 3 | 4 | 2 | 1 | 4 |
| $u_2$ | 1 | 4 | 3 | 3 | 4 |
| $u_3$ | 6 | 5 | 2 | 5 | 6 |
| $u_4$ | 6 | 4 | 5 | 5 | 6 |
| | 0 | 0 | 0 | 0 | |

Рис. 7.50. Начальная матрица ценностей

обратится в равенство $u_i + v_j = a_{ij}$. Эту процедуру выполняем для каждой строки $i = \overline{1,n}$. В рассматриваемом примере на рис. 7.51 звездочками отмечены совпадения.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | |
|---|---|---|---|---|---|
| $u_1$ | | * | | | 4 |
| $u_2$ | | * | | | 4 |
| $u_3$ | * | | | | 6 |
| $u_4$ | * | | | | 6 |
| | 0 | 0 | 0 | 0 | |

Рис. 7.51. Матрица совпадений

*Шаг 2.* Для каждого $u_i$, $i = \overline{1,n}$, определим множества совпадений

$$S_i = \{\, j \,|\, u_i + v_j = a_{ij}\}, \quad i = \overline{1,n}.$$

Если обратиться к примеру, то $S_1 = \{2\}$, $S_2 = \{2\}$, $S_3 = \{1\}$, $S_4 = \{1\}$. Система множеств $S_i, i = \overline{1,n}$, может иметь систему различных представителей: $j_1 \in S_1$, $j_2 \in S_2$, ..., $j_n \in S_n$, где $j_1, j_2, \ldots, j_n$ — все различные. Тогда на перестановке $\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ j_1 & j_2 & \ldots & j_n \end{pmatrix}$ выполняется соотношение $u_i + v_{j_i} = u_i + v_{\pi_i} = a_{i\pi_i}$, а значит, найденная перестановка $\pi$ является оптимальной.

*Шаг 3.* Предположим, что $S_1, S_2, \ldots, S_n$ не имеют системы различных представителей, как в рассматриваемом примере. Тогда нарушается условие теоремы Ф. Холла (7.14.3), т. е. существуют $S_{i_1}, S_{i_2}, \ldots, S_{i_k}$ такие, что $|S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}| < k$. В нашем примере $|S_1 \cup S_2 \cup S_3 \cup S_4| = 2 < 4$. Перейдем к новым

значениям $u_i^*$ и $v_j^*$, полагая

$$u_{i_1}^* = u_{i_1} - 1, u_{i_2}^* = u_{i_2} - 1, \ldots, u_{i_k}^* = u_{i_k} - 1$$

и

$$v_j^* = v_j + 1, \quad \text{если индекс} \quad j \in S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}.$$

Остальные $u_i$ и $v_j$ остаются без изменений. Обратимся к нашему примеру. На рис. 7.52 показаны новые значения переменных $u_i^*$ и $v_j^*$.

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | |
|---|---|---|---|---|---|
| $u_1$ | 3 | 4 | 2 | 1 | 4 3 |
| $u_2$ | 1 | 4 | 3 | 3 | 4 3 |
| $u_3$ | 6 | 5 | 2 | 5 | 6 5 |
| $u_4$ | 6 | 4 | 5 | 5 | 6 5 |
| | 0 | 0 | 0 | 0 | |
| | 1 | 1 | 0 | 0 | |

Рис. 7.52. Новые значения $u_i$ и $v_j$

При такой замене переменных не нарушается основное условие: $u_i + v_j \geqslant a_{ij} \quad \forall i, j = \overline{1, n}$. Действительно, возможны два случая:

1). $j \in S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}$, тогда $u_i^* + v_j^* = (u_i - 1) + (v_j + 1) = u_i + v_j \geqslant a_{ij}$;

2). $j \notin S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}$, а значит, $v_j^* = v_j$ и $u_i + v_j > a_{ij}$ или $u_i + v_j \geqslant a_{ij} + 1$. Отсюда $u_i^* + v_j^* = (u_i - 1) + v_j \geqslant a_{ij} + 1 - 1 = a_{ij}$.

Обозначим $|S_{i_1} \cup S_{i_2} \cup \ldots \cup S_{i_k}| = t$, тогда новая сумма (7.8)

$$\sum_{i=1}^{n} u_i^* + \sum_{j=1}^{n} v_j^* = \left(\sum_{i=1}^{n} u_i\right) - k + \left(\sum_{j=1}^{n} v_j\right) + t = \left(\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j\right) - (k - t).$$

Сумма уменьшилась на $k - t$. Если теперь перейти на шаг 2, то в случае существования различных представителей для новых значений $u_i^*$ и $v_j^*$, задача будет решена, в противном случае шаги 2 и 3 алгоритма продолжаем. Так как каждое повторение ша-

гов алгоритма приводит к уменьшению суммы (7.8) $\sum\limits_{i=1}^{n} u_i + \sum\limits_{j=1}^{n} v_j$ на $k - t$ единиц, следовательно, процесс закончится, что доказывает теорему. □

Замечание. При решении практических задач изменение переменных $u_i$ и $v_j$ на 1 может затянуть получение ответа. Заметим, что алгоритм допускает изменение переменных $u_i$ и $v_j$ на любую величину, не нарушая при этом основного условия (7.9): $u_i + v_j \geqslant a_{ij} \quad \forall i, j = \overline{1, n}$.

В рассматриваемом примере сумма уменьшилась на 2, так как $|S_1 \cup S_2 \cup S_3 \cup S_4| = t = 2$ и $k = 4$. Необходимо вернуться на шаг 2. Матрица совпадений примет вид, представленный на рисунках 7.52, 7.53. Составим для нее систему множеств совпаде-

| | $v_1$ | $v_2$ | $v_3$ | $v_4$ | |
|---|---|---|---|---|---|
| $u_1$ | | * | | | 4 3 |
| $u_2$ | | * | * | * | 4 3 |
| $u_3$ | * | | | * | 6 5 |
| $u_4$ | * | | * | * | 6 5 |
| | 0 | 0 | 0 | 0 | |
| | 1 | 1 | 0 | 0 | |

Рис. 7.53. Матрица совпадений для новых значений $u_i$ и $v_j$

ний $S_1 = \{2\}$, $S_2 = \{2, 3, 4\}$, $S_3 = \{1, 4\}$, $S_4 = \{1, 3, 4\}$. Проверим наличие системы различных представителей для данных множеств. Обращаясь к интерпретации системы различных представителей в терминах двудольного графа, надо проверить, что максимальное паросочетание равно 4 для графа $\Gamma(V_1 \cup V_2, U, \Phi)$ , где $V_1 = \{S_1, S_2, S_3, S_4\}$, $V_2 = \{1, 2, 3, 4\}$, и связь вершин $S_1 \to \{2\}$, $S_2 \to \{2, 3, 4\}$, $S_3 \to \{1, 4\}$, $S_4 \to \{1, 3, 4\}$. Решая, можно установить, что существуют три таких паросочетания:

$$\pi_1 = \{(S_1, 2), (S_2, 3), (S_3, 1), (S_4, 4)\},$$
$$\pi_2 = \{(S_1, 2), (S_2, 3), (S_3, 4), (S_4, 1)\},$$
$$\pi_3 = \{(S_1, 2), (S_2, 4), (S_3, 1), (S_4, 3)\}.$$

Таким образом, существуют три оптимальных назначения:

$$\pi_1 = \begin{pmatrix} 1\,2\,3\,4 \\ 2\,3\,1\,4 \end{pmatrix},\ \pi_2 = \begin{pmatrix} 1\,2\,3\,4 \\ 2\,3\,4\,1 \end{pmatrix},\ \pi_3 = \begin{pmatrix} 1\,2\,3\,4 \\ 2\,4\,1\,3 \end{pmatrix}.$$

Ценность назначения составляет $a_{12} + a_{23} + a_{31} + a_{44} = 18$.

## § 7.15. Хроматические графы

Пусть $\Gamma(X, U, \Phi)$ — простой граф. Граф $\Gamma$ называется *$k$-раскрашиваемым*, если существует такое разложение множества его вершин $X$ на $k$ непересекающихся подмножеств (компонент) $C_1, C_2, \ldots, C_k$ таких, что

$$X = \bigcup_{i=1}^{k} C_i, \quad C_i \cap C_j = \varnothing \quad \text{при} \quad i \neq j \tag{7.10}$$

и вершины в каждой компоненте $C_i$ независимы, т. е. ребра в $\Gamma$ соединяют вершины только из разных компонент (рис. 7.54). Представление (7.10) называется *$k$-раскраской графа* $\Gamma$. Вершины этого графа можно раскрасить $k$ красками так, чтобы смежные вершины всегда были окрашены в разные цвета. Для этого достаточно вершины каждой компоненты $C_i$ покрасить своей краской. Наименьшее возможное число $k = \chi(\Gamma)$ компонент в разложении (7.10) называется *хроматическим числом* графа.

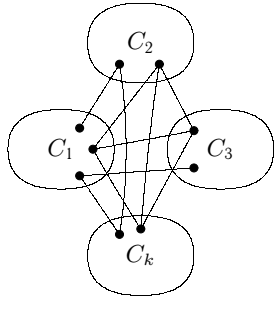


Рис. 7.54

Рассмотрим несколько примеров хроматического разложения графов.

Утверждение 7.15.1. *Полный граф $\Gamma(X, U, \Phi)$ на $|X| = n$ вершинах имеет хроматическое число $\chi(\Gamma) = n$. Здесь каждая компонента $C_i$, $i = 1, 2, \ldots, n$, разложения* (7.10) *состоит из одной вершины, $|C_i| = 1$.*

Утверждение 7.15.2. *Граф $\Gamma(X, U, \Phi)$ содержит максимальный подграф (клику) из $k$ вершин тогда и только когда, когда его хроматическое число $\chi(\Gamma)$ равно $k$.*

Доказательство. ($\Rightarrow$) Пусть $\{x_1, x_2, \ldots, x_k\} \subseteq X$ вершины максимального подграфа. Для разложения (7.10) максимального подграфа требуется $k$ компонент $C_1, C_2, \ldots, C_k$ таких, что $x_i \in C_i$. Покажем, что этого числа компонент достаточно для разложения (7.10) всего графа $\Gamma$. Пусть $y \in X$ — произвольная вершина и $y \notin \{x_1, x_2, \ldots, x_k\}$. Среди $x_i$, $i = \overline{1,k}$, существует такая вершина $x_j \in C_j$, которая несмежна с $y$, в противном случае существовал бы максимальный подграф из $k+1$ вершины. Вершину $y$ включаем в компоненту $C_j$. Следовательно, $\chi(\Gamma) = k$.

($\Leftarrow$) Имеем $\chi(\Gamma) = k$. Предположим, что максимальный подграф в $\Gamma$ содержит $m$ вершин и $m \neq k$. Необходимое условие теоремы в этом случае утверждает: $\chi(\Gamma) = m$, что противоречит условию. $\square$

Утверждение 7.15.3. *Граф $\Gamma(X, U, \Phi)$ является двудольным тогда и только тогда, когда $\chi(\Gamma) = 2$.*

Утверждение 7.15.4. *Хроматическое число дерева равно 2, так как дерево является двудольным графом.*

Теорема 7.15.1. *Для числа $\beta(\Gamma)$ вершинной независимости и хроматического числа $\chi(\Gamma)$ графа $\Gamma(X, U, \Phi)$, $|X| = n$, выполняется соотношение*

$$\beta(\Gamma)\chi(\Gamma) \geqslant n. \tag{7.11}$$

Доказательство. В разложении (7.10) все компоненты $C_i$ являются независимыми. Из определения числа $\beta(\Gamma)$ следует, что $|C_i| \leqslant \beta(\Gamma)$. Суммируя по всем компонентам, получим $n = \sum\limits_{i=1}^{k} |C_i| \leqslant k|\beta(\Gamma)|$, где $k = \chi(\Gamma)$. $\square$

Задача. Для графа $\Gamma$ на рис. 7.55 найти разложение (7.10) и установить, что хроматическое число $\chi(\Gamma) = 3$.

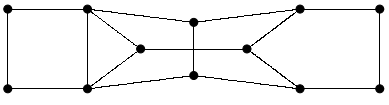


Рис. 7.55

Задача. Пусть $l$ — длина самой длинной простой цепи в графе $\Gamma$. Показать, что $\chi(\Gamma) \leqslant l + 1$.

Решение. Предположим, что $\chi(\Gamma) \geqslant l + 2$. Тогда утверждение 7.15.2 позволяет сделать заключение, что граф $\Gamma$ содержит максимальный подграф (клику) из $l + 2$ вершин или более. Такой подграф содержт простой цикл из $l + 2$ ребер и простую цепь длиной $l + 1$. Это противоречит условию задачи, так как любая простая цепь графа не может превосходить длины $l$. Отсюда следует, что $\chi(\Gamma) \leqslant l + 1$.

## § 7.16. Диаметр, радиус и центры графа

1. Пусть $\Gamma(X, U, \Phi)$ — конечный связный псевдограф. Обозначим через $d(x, y)$ длину минимального маршрута между вершинами $x, y \in X$. Величина $d(\Gamma) = \max\limits_{x,y \in X} d(x, y)$ называется *диаметром графа*.

2. Пусть $x \in X$ — произвольная вершина графа. Величина $r(x) = \max\limits_{y \in X} d(x, y)$ называется *максимальным удалением* в графе $\Gamma$ от вершины $x$.

3. *Радиусом* графа $\Gamma$ называется величина $r(\Gamma) = \min\limits_{x \in X} r(x)$.

4. Любая вершина $z \in X$, для которой $r(z) = r(\Gamma)$, называется *центром* графа.

З а д а ч а. Для графа $\Gamma$ на рис. 7.56 найти диаметр, радиус и все центры.

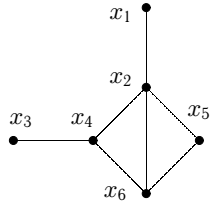


Рис. 7.56

Р е ш е н и е. *Диаметр*

$$d(\Gamma) = \max_{x,y \in X} d(x, y) = 3,$$

$$r(x_1) = 3, \quad r(x_4) = 2,$$
$$r(x_2) = 2, \quad r(x_5) = 3,$$
$$r(x_3) = 3, \quad r(x_6) = 2.$$

Следовательно, радиус графа $r(\Gamma) = \min\limits_{x \in X} r(x) = 2$. Центры графа: $x_2$, $x_4$, $x_6$.

Глава 8

# ВВЕДЕНИЕ В ТЕОРИЮ ГРУПП. ПРИЛОЖЕНИЯ

## § 8.1. Определение группы

Определение. *Группой* называется непустое множество $G$ с бинарной алгебраической операцией $\cdot$ (*будем называть умножением*) такой, что выполняются следующие аксиомы:

1. $\forall a, b \in G\, \exists c \in G\ a \cdot b = c$ — замкнутость операции $\cdot$.
2. $\forall a, b, c \in G\ a \cdot (b \cdot c) = (a \cdot b) \cdot c$ — ассоциативность операции $\cdot$.
3. $\exists! e \in G \forall a \in G\ e \cdot a = a \cdot e = a$ — существование единичного элемента $e$.
4. $\forall a \in G\, \exists! a^{-1} \in G\ a \cdot a^{-1} = a^{-1} \cdot a = e$ — существование обратного элемента $a^{-1}$.

Определение. Группа называется *коммутативной*, если выполняется аксиома коммутативности:

5. $\forall a, b \in G\ a \cdot b = b \cdot a$ — коммутативность операции $\cdot$.

### Примеры групп

1. $G_1 = \{n \,|\, n \in Z\}$ — группа с операцией сложения чисел, где $Z$ — множество целых чисел. Действительно, 0 — единица группы; $n^{-1} = (-n)$ — обратный элемент к $n \in G_1$.
2. $G_2 = \{2n \,|\, n \in Z\}$ — группа с операцией сложения чисел.
3. $G_3 = \{2^n \,|\, n \in Z\}$ — группа с операцией умножения чисел.
4. $G_4$ — множество квадратных матриц порядка $n$, определитель которых не равен нулю, является группой с операцией умножения матриц.
5. $G_5$ — множество ортогональных матриц порядка $n$ является группой с операцией умножения матриц.
6. $G_6 = \{-1, 1\}$ — группа с операцией умножения чисел.
7. $G_7$ — множество рациональных чисел является группой относительно операций сложения и умножения (без нуля).

8. $G_8$ — множество вещественных чисел является группой относительно операций сложения и умножения (без нуля).
9. $G_9 = S_m$ — множество подстановок (перестановок) является группой с операцией умножения подстановок (симметрическая группа $S_m$, см. п. 8.6).

Определение. $H \subseteq G$ называется *подгруппой* группы $G$, если $H$ — группа относительно бинарной операции, определенной в $G$, т. е. для элементов $H$ выполняются аксиомы 1–4. Так, например, являются подгруппами: $G_2 \subset G_1$, $G_5 \subset G_4$, $G_7 \subset G_8$.

Утверждение 8.1.1. *Пусть $M$ — подмножество группы $G$ и $\forall a, b \in M$ выполняется $ab^{-1} \in M$. Показать, что $M$ — подгруппа.*

Данное условие можно рассматривать как характеристическое свойство группы.

Доказательство. Проверим выполнение аксиом группы.
1. *Существование единичного элемента.* Пусть $a \in M$, тогда $aa^{-1} \in M$ или $e \in M$.
2. *Существование обратного элемента.* Пусть $a \in M$ и так как $e \in M$, то $ea^{-1} \in M$ или $a^{-1} \in M$.
3. *Замкнутость.* Пусть $a, b \in M$, тогда и $b^{-1} \in M$. Значит, и $a(b^{-1})^{-1} \in M$ или $ab \in M$. □

## § 8.2. Гомоморфизм групп

Определение. *Гомоморфизмом* группы $G_1$ в $G_2$ называется отображение $f : G_1 \to G_2$, сохраняющее операции: $\forall a, b \in G_1$ $f(a \circ b) = f(a) \bullet f(b)$, где $\circ$ — операция умножения в группе $G_1$; $\bullet$ — операция умножения в группе $G_2$. Если $f$ — взаимно однозначное отображение, то $f$ называется изоморфизмом.

### Свойства гомоморфизма

Свойство 1. *Единичный элемент переходит в единичный.*

Пусть $e_1 \in G_1$ — единица $G_1$, тогда $f(e_1) = e_2 \in G_2$ — единица $G_2$. Действительно, $\forall a \in G_1 f(a) = f(a)f(e_1) = f(e_1)f(a)$, так как $f(a) = f(ae_1) = f(a)f(e_1)$ и $f(a) = f(e_1a) = f(e_1)f(a)$. В группе $\exists! e_2 \in G_2 f(a) = f(a)e_2 = e_2f(a)$. Значит, $f(e_1) = e_2$.

Свойство 2. *Обратный элемент переходит в обратный.*

Пусть $a \in G_1$, тогда $f(a^{-1}) = (f(a))^{-1}$. Действительно, $f(a)f(a^{-1}) = f(a^{-1})f(a) = e_2$, так как $f(a)f(a^{-1}) = f(aa^{-1}) = f(e_1) = f(a^{-1}a) = f(a^{-1})f(a)$, где $f(e_1) = e_2$. В группе $\exists!(f(a))^{-1} \in G_2 f(a)(f(a))^{-1} = (f(a))^{-1}f(a) = e_2$. Следовательно, $f(a^{-1}) = (f(a))^{-1}$.

Определение. *Образом* гомоморфизма $f$ называется подмножество $\operatorname{Im} f = \{a_2 \in G_2 \mid \exists a_1 \in G_1\, a_2 = f(a_1)\} \subseteq G_2$.

Определение. *Ядром* гомоморфизма $f$ называется подмножество $\operatorname{Ker} f = \{a_1 \in G_1 \mid f(a_1) = e_2 \in G_2\} \subseteq G_1$.

Утверждение 8.2.1. $\operatorname{Im} f \subseteq G_2$ — *подгруппа*.

Доказательство. Проверим выполнение аксиом группы.

1. *Замкнутость.* $\forall a_2, b_2 \in \operatorname{Im} f\ \exists a_1, b_1 \in G_1\ f(a_1) = a_2 \wedge f(b_1) = b_2$. Тогда $a_2b_2 = f(a_1)f(b_1) = f(a_1b_1) \in \operatorname{Im} f$.
2. *Существование единичного элемента.* Так как $e_2 = f(e_1) \in \operatorname{Im} f$.
3. *Существование обратного элемента.* $\forall a_2 \in \operatorname{Im} f\ \exists a_1 \in G_1\ f(a_1) = a_2$ . Тогда и $a_2^{-1} = (f(a_1))^{-1} = f(a_1^{-1}) \in \operatorname{Im} f$. □

Утверждение 8.2.2. $\operatorname{Ker} f \subseteq G_1$ — *подгруппа*.

Доказательство. Проверим выполнение аксиом группы.

1. *Замкнутость.* $\forall a_1, b_1 \in \operatorname{Ker} f\ f(a_1b_1) = f(a_1)f(b_1) = e_2e_2 = e_2$. Следовательно, $a_1b_1 \in \operatorname{Ker} f$.
2. *Существование единичного элемента.* Так как $f(e_1) = e_2$, значит, $e_1 \in \operatorname{Ker} f$.
3. *Существование обратного элемента.* $\forall a_1 \in \operatorname{Ker} f\ f(a_1^{-1}) = (f(a_1))^{-1} = e_2^{-1} = e_2$, отсюда $a_1^{-1} \in \operatorname{Ker} f$. □

## § 8.3. Смежные классы

Определение. Пусть $H = \{e, h_1, h_2, \ldots, h_{m-1}\}$ — подгруппа группы $G$. Множество $gH = \{ge, gh_1, gh_2, \ldots, gh_{m-1}\}$, полученное умножением элементов $H$ слева на элемент $g$, называется *левым смежным классом* группы $G$ по подгруппе $H$.

Лемма 8.3.1. *Пусть $H$ — подгруппа группы $G$, тогда* $\forall h \in H \quad hH = H$.

Доказательство. Пусть $H = \{e, h_1, h_2, \ldots, h_{m-1}\}$. $\forall h_k \in H\ hh_k \in H$ — замкнутость подгруппы, значит, $hH \subseteq H$. С другой стороны, $\forall h_k \in H\ h_k = eh_k = (hh^{-1})h_k = h(h^{-1}h_k) \in \in hH$, отсюда $H \subseteq hH$. Таким образом, $\forall h \in H\ hH = H$. □

Лемма 8.3.2. *Пусть $H$ — подгруппа группы $G$. Если $g_1H \cap g_2H \neq \varnothing$, то $g_1H = g_2H$, где $g_1, g_2 \in G$.*

Доказательство. Пусть $t \in g_1H \cap g_2H$, тогда $\exists h_1, h_2 \in \in H$ такие, что $t = g_1h_1 = g_2h_2$. Рассмотрим левый смежный класс $tH = (g_1h_1)H = (g_2h_2)H = g_1(h_1H) = g_2(h_2H)$, отсюда $g_1H = g_2H$. □

Лемма 8.3.3. *Пусть $H$ — подгруппа группы $G$, тогда $\forall g \in G\ |gH| = |H|$.*

Доказательство. Достаточно показать, что все элементы в $gH$ различные. Пусть $\exists h_1 \neq h_2 \in H$ такие, что $gh_1 = gh_2$. Тогда $g^{-1}(gh_1) = g^{-1}(gh_2)$ или $(g^{-1}g)h_1 = (g^{-1}g)h_2$. Отсюда $h_1 = h_2$, что противоречит предположению. □

Лемма 8.3.4. *Группа $G$ распадается на непересекающиеся левые смежные классы по подгруппе $H$, т.е. $G = = g_1H \cup g_2H \cup \ldots \cup g_sH$, где все $g_1, g_2, \ldots, g_s$ — различные; $g_iH \cap g_jH = \varnothing$, если $g_i \neq g_j$.*

Доказательство. Пусть $G = \{g_{k_1}, g_{k_2}, \ldots, g_{k_m}\}$. Ясно, что $G = g_{k_1}H \cup g_{k_2}H \cup \ldots \cup g_{k_m}H$. Воспользовавшись леммой 8.3.2 и удалив совпадающие левые смежные классы из последнего разложения, получим искомое разложение группы $G = g_1H \cup g_2H \cup \ldots \cup g_sH$. □

Теорема 8.3.1 (Лагранжа). *Порядок конечной группы $G$ кратен порядку любой из ее подгрупп $H$ и $|G| = |H||G : H|$, где $|G : H|$ — целое число, называется индексом подгруппы $H$ в группе $G$.*

Доказательство. Лемма 8.3.4 позволяет записать разложение группы $G = g_1H \cup g_2H \cup \ldots \cup g_sH$, где $g_iH \cap g_jH = \varnothing$ при $g_i \neq g_j$. Из леммы 8.3.3 имеем $\forall g_i\ |g_iH| = |H|$, тогда $|G| = = |g_1H \cup g_2H \cup \ldots \cup g_sH| = |g_1H| + |g_2H| + \ldots + |g_sH| = s|H|$. Следовательно, $|G| = |H||G : H|$, где $|G : H| = s$. □

Определение. Пусть $G$ — группа. *Порядком* элемента $g \in G$ $(g \neq e)$ называется наименьшее целое $k$ такое, что $g^k = e$.

Утверждение 8.3.1. *Пусть $G$ — группа и $g \in G$ — произвольный элемент порядка $k$. Тогда множество $H = \{g, g^2, \ldots \ldots, g^k = e\}$ называется циклической подгруппой, а $g$ есть образующий ее элемент.*

Утверждение 8.3.2. *Порядок группы $G$ кратен порядку любого элемента $g \in G$.*

Следствие теоремы 8.3.1 Лагранжа и утверждения 8.3.1.

Утверждение 8.3.3. *Всякая циклическая группа коммутативна* (*абелева*).

Теорема 8.3.2. *Всякая подгруппа циклической группы сама является циклической.*

Доказательство. Пусть $G$ — циклическая группа с образующим элементом $g \in G$ и $H \subseteq G$ — подгруппа. Предположим, что наименьшая положительная степень элемента $g$, содержащаяся в $H$, есть $g^k$. Покажем, что элемент $g^k$ — образующий элемент подгруппы $H$. Допустим, что в $H$ содержится элемент $g^l$, где $l \neq 0$ и $l$ не делится на $k$. Пусть $d = (l, k)$ — наибольший общий делитель, тогда существуют такие целые числа $u$ и $v$, что $k \cdot u + l \cdot v = d$ (см. п. 9.1). Следовательно, подгруппа $H$ в этом случае должна содержать элемент $g^d = (g^k)^u (g^l)^v = g^{k \cdot u + l \cdot v}$. Так как $d < k$, то приходим к противоречию относительно выбора элемента $g^k$. Таким образом, $g^k$ — образующий элемент подгруппы $H \subseteq G$. □

Утверждение 8.3.4. *Число образующих циклической группы $G = \{g^1, g^2, \ldots, g^n = e\}$ равно значению функции Эйлера $\varphi(n)$ — количество чисел из множества $\{1, 2, \ldots, n-1\}$, взаимно простых с $n$* (см. п. 9.7).

Доказательство. Пусть $r \in \{1, 2, \ldots, n-1\}$ и $(r, n) = 1$ — НОД (см. п. 9.1), т. е. $r$ и $n$ — взаимно простые. Если предположить, что наступит $(g^r)^k = g^{rk} = e$ для некоторого $k < n$, то $r \cdot k = n \cdot t$ для некоторого $t$, так как $n$ — порядок элемента $g$. Из $r \cdot k = n \cdot t$ следует, что $n$ делит $k$, так как $(r, n) = 1$. Это противоречит предположению $k < n$. Следовательно, порядок элемента $|g^r| = n$ .

Пусть теперь $(r, n) = s$ и $s > 1$, т. е. $r = s \cdot p$ и $n = s \cdot q$, где $(p, q) = 1$. Тогда $(g^r)^q = (g^{sp})^q = g^{spq} = (g^{sq})^p = (g^n)^p = e^p = e$, а значит, порядок элемента $|g^r| = q < n$. Действительно, порядок $|g^r|$ не меньше $q$, в противном случае имеем $(g^r)^q = (g^r)^k = e$,

где $k < q$, и $r \cdot k = n \cdot t$, так как $n$ — порядок элемента $g$. Как и в первом случае, из $r \cdot k = n \cdot t$, $s \cdot p \cdot k = s \cdot q \cdot t$, $p \cdot k = q \cdot t$ следует, что $q$ делит $k$, так как $(p, q) = 1$. Это противоречит предположению $k < q$. □

Определение. Подгруппа $H$ группы $G$ называется *нормальным делителем*, если правые смежные классы совпадают с левыми: $\forall g \in G \;\; gH = Hg$.

Утверждение 8.3.5. *Множество смежных классов группы $G$ по нормальному делителю $H$ является группой с операцией умножения смежных классов.*

Такая группа называется *факторгруппой* и обозначается $G/H$. Элементами этой группы являются смежные классы разложения группы $G = g_1H \cup g_2H \cup \ldots \cup g_sH$ на непересекающиеся левые смежные классы, т. е. $G/H = \{g_1H, g_2H, \ldots, g_sH\}$.

Доказательство. Проверим выполнение аксиом группы.

1. *Замкнутость.* $\forall g_1H, g_2H \in G/H \; g_1H \cdot g_2H = g_1(Hg_2)H = g_1(g_2H)H = g_1g_2(HH) = (g_1g_2)H \in G/H$. Произведение двух классов — это умножение каждого с каждым элементов указанных классов.
2. *Существование единичного элемента.* Так как $(eH)(gH) = (gH)(eH) = (eg)H = gH$, то $eH = H$ — единица факторгруппы $G/H$.
3. *Существование обратного элемента.* Так как $(gH)(g^{-1}H) = (g^{-1}H)(gH) = (gg^{-1})H = H$, то $g^{-1}H \in G/H$ — обратный элемент к элементу $gH \in G/H$. □

Теорема 8.3.3. *Для любого нормального делителя $H$ группы $G$ отображение $f : G \to G/H$, где $\forall g \in G \;\; f(g) = gH$, является гомоморфизмом, ядро которого $H$, $G/H$ — факторгруппа.*

Доказательство. Проверим свойство гомоморфизма сохранения операций: $f(g_1 \cdot g_2) = (g_1 \cdot g_2)H = (g_1H) \circ (g_2H) = f(g_1) \circ f(g_2)$.

Единицей факторгруппы $G/H$ является $H$, тогда $\operatorname{Ker} f = \{g \in G | f(g) = H\}$. Имеем $\forall h \in H \; f(h) = hH$, отсюда $H \subseteq \operatorname{Ker} f$. С другой стороны, $\forall g \notin H \;\; f(g) = gH \neq H$. В противном случае, если $gH = H$, то существует такое $h \in H$, что $gh = e$ или $g = h^{-1} \in H$, что противоречит предположению $g \notin H$. Таким образом, только $H \subseteq \operatorname{Ker} f$, а значит, $\operatorname{Ker} f = H$. □

Теорема 8.3.4. *Ядро произвольного гомоморфизма есть нормальный делитель.*

Доказательство. Пусть $f : G \to K$, где $G, K$ — группы, $f$ — гомоморфизм. $\operatorname{Ker} f = \{h \in G \mid f(h) = e \in K\}$. Обозначим $H = \operatorname{Ker} f$ — подгруппа. Покажем, что $\forall g \in G \quad gH = Hg$, т. е. $H$ — нормальный делитель. Рассмотрим множество $S = \{g \in G \mid f(g) = k \in K\}$, где $k$ — фиксированный элемент. Покажем, что $S = gH$, где $g \in S$ — произвольный фиксированный элемент. Пусть $g_1 \in S$, тогда $f(g_1 g^{-1}) = f(g_1)f(g^{-1}) = kk^{-1} = e$ и $f(g^{-1}g_1) = f(g^{-1})f(g_1) = k^{-1}k = e$. Отсюда $g_1 g^{-1} \in H$ и $g^{-1}g_1 \in H$ или $g_1 \in Hg$ и $g_1 \in gH$. Таким образом, $S \subseteq gH$ и $S \subseteq Hg$, где $g \in S$. С другой стороны, $\forall h \in H \quad f(gh) = f(g)f(h) = f(g)e = k$. Отсюда $gh \in S$ или $gH \subseteq S$. Так же проверяется, что и $Hg \subseteq S$.

Получили, что $gH = S = Hg$, т. е. $H$ — нормальный делитель. □

## § 8.4. Строение коммутативных (абелевых) групп

Определение. Группа $G$ является *прямым произведением* своих подгрупп $G_1$ и $G_2$, т. е. $G = G_1 \times G_2$, если выполнены следующие условия:

1) пересечение подгрупп $G_1 \cap G_2 = e$;
2) любой элемент $g \in G$ однозначно представим в виде произведения элементов $g = g_1 g_2 \cdot \ldots \cdot g_k$, где $g_i \in G_1 \cup G_2$;
3) если $g_{r_1} \in G_1$ и $g_{r_2} \in G_2$, то $g_{r_1} g_{r_2} = g_{r_2} g_{r_1}$. Коммутативность указанных элементов позволяет записать представление $g = g_1 g_2 \cdot \ldots \cdot g_k \in G$, где $g_i \in G_1 \cup G_2$, в виде $g = g_{s_1} g_{s_2}$, где $g_{s_1} \in G_1$ и $g_{s_2} \in G_2$. Лемма 8.4.1 утверждает, что это представление однозначно.

Лемма 8.4.1. *Пусть $G$ — группа и $G_1$, $G_2$ — ее подгруппы, для которых пересечение $G_1 \cap G_2 = e$. Тогда, если $g_1 g_2 = g_1' g_2'$, то $g_1 = g_1'$ и $g_2 = g_2'$, где $g_1, g_1' \in G_1$ и $g_2, g_2' \in G_2$.*

Доказательство. Действительно, если $g_1 g_2 = g_1' g_2'$, то $g_1^{-1} g_1' = g_2^{-1} g_2'$, так как $G_1 \cap G_2 = e$. Следовательно, $g_1 = g_1'$ и $g_2 = g_2'$. □

Теорема 8.4.1. *Группа $G$ является прямым произведением своих подгрупп $G_1$ и $G_2$ тогда и только тогда, когда выполняются условия:*

*1) пересечение подгрупп* $G_1 \cup G_2 = e$;
*2) любой элемент* $g \in G$ *однозначно представим в виде произведения элементов* $g = g_1 g_2 \cdot \ldots \cdot g_k$, *где* $g_i \in G_1 \cup G_2$;
*3) подгруппы* $G_1$ *и* $G_2$ *являются нормальными делителями.*

Доказательство. ($\Rightarrow$) Очевидно, что если группа $G$ является прямым произведением своих подгрупп $G_1$ и $G_2$, то условия 1–2 выполняются. Проверим справедливость условия 3 для $G_1$. Пусть $g \in G$, тогда $g = g_{s_1} g_{s_2} = g_{s_2} g_{s_1}$, где $g_{s_1} \in G_1$ и $g_{s_2} \in G_2$. Имеем $gG_1 = (g_{s_2} g_{s_1}) G_1 = g_{s_2}(g_{s_1} G_1) = g_{s_2} G_1 = G_1 g_{s_2}$. Подобным образом $G_1 g = G_1(g_{s_1} g_{s_2}) = (G_1 g_{s_1}) g_{s_2} = G_1 g_{s_2}$. Итак, $gG_1 = G_1 g$, а значит, $G_1$ — нормальный делитель.

($\Leftarrow$) Достаточно показать, что если $g_1 \in G_1$ и $g_2 \in G_2$, то $g_1 g_2 = g_2 g_1$. Имеем $g_1 g_2 \in g_1 G_2$ и $g_1 G_2 = G_2 g_1$, тогда $g_1 g_2 \in G_2 g_1$, отсюда $g_1 g_2 \in g_2' g_1$. Аналогично, $g_1 g_2 \in G_1 g_2$ и $G_1 g_2 = g_2 G_1$, тогда $g_1 g_2 \in g_2 G_1$ и, значит, $g_1 g_2 = g_2 g_1'$. Итак, $g_1 g_2 = g_2' g_1 = g_2 g_1'$, а так как $G_1 \cup G_2 = e$, то из леммы 8.4.1 следует, что $g_2' = g_2$ и $g_1' = g_1$. Следовательно, $g_1 g_2 = g_2 g_1$. □

Утверждение 8.4.1. *Пусть* $G$ *— конечная абелева группа порядка* $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, *где* $p_1, p_2, \ldots, p_k$ *— простые различные числа. Множество* $A(p) = \{x \in G \mid |x| = p^\alpha\}$, *где* $\alpha$ *принимает произвольные целые значения, является подгруппой и называется примарной подгруппой группы* $G$, *соответствующей простому числу* $p$.

Доказательство. Достаточно проверить замкнутость $A(p)$ и существование обратного элемента.

1. *Замкнутость.* $\forall x, y \in A(p)$ имеем $x^{p^\alpha} = e$ и $x^{p^\beta} = e$. Тогда $(xy)^{p^{\alpha+\beta}} = x^{p^{\alpha+\beta}} y^{p^{\alpha+\beta}} = (x^{p^\alpha})^{p^\beta} (y^{p^\beta})^{p^\alpha} = e$.

2. *Обратный элемент.* $\forall x \in A(p)$ имеем $x^{p^\alpha} = e$. Рассмотрим $e = (x \cdot x^{-1})^{p^\alpha} = x^{p^\alpha} (x^{-1})^{p^\alpha} = (x^{-1})^{p^\alpha}$. Значит, $x^{-1} \in A(p)$. □

Теорема 8.4.2. *Всякая конечная абелева группа* $G$ *разлагается в прямое произведение своих примарных подгрупп* $A(p_1), A(p_2), \ldots, A(p_k)$, *где* $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$.

Доказательство — индукция по числу простых в разложении порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$. Очевидно, что для $|G| = p^\alpha$ справедливо, так как в этом случае $A(p) = G$. Пусть теперь $|G| = p^\alpha q^\beta$, где $(p, q) = 1$. Покажем, что $G$ представимо в виде $G = A(p) \times A(q)$. Проверим свойства разложения.

1. $A(p) \cap A(q) = e$. Если предположить, что $\exists x \in A(p) \cap A(q)$ и $x \neq e$, то $|x| = p^\alpha$ и $|x| = q^\beta$, тогда и $p^\alpha = q^\beta$, что противоречит условию $(p, q) = 1$.

2. Покажем, что любой элемент $x \in G$ можно представить в виде $x = y \cdot z$, где $y \in A(p)$, $z \in A(q)$. Поскольку порядок $|x|$ делит $|G|$, то $|x| = p^{\lambda_1} q^{\lambda_2}$, где $\lambda_1 \leqslant \alpha$, $\lambda_2 \leqslant \beta$. Так как $(p^{\lambda_1}, q^{\lambda_2}) = 1$ — взаимно простые, то существует представление $p^{\lambda_1} m + q^{\lambda_2} n = 1$ (см. п. 9.1 алгоритм Евклида), где $m, n$ — целые. Тогда $x^{p^{\lambda_1} m + q^{\lambda_2} n} = x$ или $x = y \cdot z$, где $y = x^{q^{\lambda_2} n}$ и $z = x^{p^{\lambda_1} m}$, для которых $y^{p^{\lambda_1}} = \left(x^{p^{\lambda_1} q^{\lambda_2}}\right)^n = e$ и $z^{q^{\lambda_2}} = \left(x^{p^{\lambda_1} q^{\lambda_2}}\right)^m = e$. Проверить, что если $y^N = e$, то порядок $|y|$ делит $N$. Отсюда $y \in A(p)$ и $z \in A(q)$.

Пусть разложение верно для всех $l < k$, т. е. для $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_l^{\alpha_l}$. Рассмотрим группу порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, тогда возможно прямое разложение $G = A(p_1, p_2, \ldots, p_{k-1}) \times A(p_k)$, где $A(p_1, p_2, \ldots, p_{k-1}) = \{x \in G \mid |x| = p_1^{\gamma_1} p_2^{\gamma_2} \cdot \ldots \cdot p_{k-1}^{\gamma_{k-1}}\}$, $\gamma_i \leqslant \alpha_i$, — это доказывается как и для случая $|G| = p^\alpha q^\beta$. Имеем: группа $A(p_k)$ — примарная, группа $A(p_1, p_2, \ldots, p_{k-1})$ по предположению индукции разлагается в прямое произведение своих примарных подгрупп. □

Утверждение 8.4.2. *Порядок конечной примарной группы (подгруппы) $A(p) = \{x \mid |x| = p^\alpha\}$ равен $|A(p)| = p^n$, где $n$ — некоторое положительное целое; $\alpha$ принимает произвольные целые значения.*

Доказательство. Рассмотрим следующее разложение примарной группы. Пусть $A_1 = \{a, a^2, \ldots, a^{p^{\alpha_1}}\}$ — циклическая подгруппа максимального порядка $|A_1| = p^{\alpha_1}$. По теореме Лагранжа (см. теорему 8.3.1) $|A(p)| = |A_1||A(p)/A_1|$, где $A(p)/A_1 = \{x_1 A_1, x_2 A_1, \ldots, x_k A_1\}$ — факторгруппа по подгруппе $A_1$. Ясно, что $\forall x_i A_1 \in A(p)/A_1$ $(x_i A_1)^{p^\alpha} = e$ и, значит, $B(p) = A(p)/A_1$ — примарная группа, которая вновь допускает разложение на смежные классы по циклической подгруппе $A_2 \subseteq B(p)$ максимального порядка, т. е. $|A(p)| = |A_1||A_2||B(p)/A_2|$, где $|A_2| = p^{\alpha_2}$. Исходная примарная группа $A(p)$ — группа конечного порядка и, следовательно, за конечное число $m$ шагов получим разложение $|A(p)| = |A_1||A_2| \cdot \ldots \cdot |A_m|$ или $|A(p)| = p^{\alpha_1} p^{\alpha_2} \cdot \ldots \cdot p^{\alpha_m} = p^n$, где $|A_i| = p^{\alpha_i}$. □

Лемма 8.4.2. *Пусть $A(p)$ — примарная группа (подгруппа), $A \subseteq A(p)$ — циклическая подгруппа максимального*

*порядка* $|A| = p^n$ *с образующим элементом* $a$, $A(p)/A$ — *факторгруппа*, $\overline{y} \in A(p)/A$ — *класс смежности порядка* $|\overline{y}| = p^\alpha$, *т. е.* $\overline{y}^{p^\alpha} = \overline{e} = A$, *тогда существует элемент* $y \in \overline{y}$ *того же порядка* $|y| = p^\alpha$.

Доказательство. Так как $\overline{y}^{p^\alpha} = A$, то для любого $y \in \overline{y}$ выполняется $y^{p^\alpha} \in A$, где $A$ — единица факторгруппы $A(p)/A$. Следовательно, $y^{p^\alpha} = a^c = a^{c_1 p^\beta}$, где $(c_1, p) = 1$ и $\beta \leqslant n$. Положим $z = a^{c_1}$ — образующий элемент группы $A$, тогда $y^{p^\alpha} = z^{p^\beta}$ и $\left(y^{p^\alpha}\right)^{p^{n-\beta}} = \left(z^{p^\beta}\right)^{p^{n-\beta}} = z^{p^n} = e$ или $y^{p^{\alpha+(n-\beta)}} = e$. Так как $z$ — образующий элемент $A$, то $p^{\alpha+n-\beta}$ — порядок $y$. Вследствие максимального порядка подгруппы $|A| = p^n$, порядок $|y| \leqslant p^n$. Отсюда $p^{\alpha+n-\beta} \leqslant p^n$ или $\alpha \leqslant \beta$. Теперь равенство $y^{p^\alpha} = z^{p^\beta}$ можно записать в виде $y^{p^\alpha} = z^{p^\beta} = \left(z^{p^{\beta-\alpha}}\right)^{p^\alpha}$, а так как $z_1 = z^{p^{\beta-\alpha}} \in A$, то $y^{p^\alpha} = z_1^{p^\alpha}$ или $(yz_1^{-1})^{p^\alpha} = e$. Таким образом, элемент $y_1 = yz^{-1} \in \overline{y}$, порядок которого $|y_1| = p^\alpha$. □

Теорема 8.4.3. *Всякая конечная примарная группа* (*подгруппа*) *разложима в прямое произведение своих циклических подгрупп. Если* $A(p) = \{x \in G \mid |x| = p^\alpha\}$ *и* $|A(p)| = p^n$, *то* $A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \ldots \times C(p^{\lambda_m})$, *где* $C(p^{\lambda_i})$ — *циклическая подгруппа порядка* $|C(p^{\lambda_i})| = p^{\lambda_i}$, $\lambda_1 \leqslant \lambda_2 \leqslant \ldots \leqslant \lambda_m$ *и* $\lambda_1 + \lambda_2 + \ldots + \lambda_m = n$.

Доказательство (*индукция по числу* $n$). Для $n = 1$ теорема верна, так как (показать!) группа $A(p)$ порядка $|A| = p$, где $p$ — простое, является циклической и не содержит подгрупп. Предположим, что теорема верна для всех групп меньшего порядка $p^n$. Пусть $A \subseteq A(p)$ — циклическая подгруппа максимального порядка $|A| = p^\alpha$. Рассмотрим факторгруппу $A(p)/A = \{x_1 A, x_2 A, \ldots, x_k A\}$. Данная группа является примарной порядка $|A(p)/A| < p^n$, так как $|A(p)| = |A||A(p)/A|$. Предположение индукции позволяет записать для нее разложение $A(p)/A = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \ldots \times C(p^{\lambda_k})$. Обозначим через $\overline{z}_i \in C(p^{\alpha_i})$ образующие циклических подгрупп. Лемма 8.4.2 утверждает, что существует $y_i \in \overline{z}_i$ и $|y_i| = |\overline{z}_i|$ и можно положить $\overline{z}_i = y_i A$. Из прямого разложения факторгруппы следует, что любой класс смежности $\overline{x} \in A(p)/A$ можно представить в виде $\overline{x} = \overline{z}_1^{t_1} \overline{z}_2^{t_2} \cdot \ldots \cdot \overline{z}_k^{t_k}$, или $\overline{x} = y_1^{t_1} y_2^{t_2} \cdot \ldots \cdot y_k^{t_k} A$, где $0 \leqslant t_i \leqslant p^{\alpha_i}$. Так как $A(p) = x_1 A \cup x_2 A \cup \ldots \cup x_k A$, то произвольный элемент $x \in A(p)$ представим как $x = y_1^{t_1} y_2^{t_2} \cdot \ldots \cdot y_k^{t_k} a^t$. Таким об-

разом, искомое прямое разложение на циклические подгруппы найдено.

Покажем, что представление $A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \ldots \times \times C(p^{\lambda_m})$ единственно. Доказательство единственности — индукция по числу $n$. Для $n = 1$ имеем $A(p) = C(p)$ — свойство выполняется. Пусть верно для всех $k < n$. Покажем, что верно и для групп порядка $p^n$. Предположим существование другого разложения группы $A(p) = C(p^{\beta_1}) \times C(p^{\beta_2}) \times \ldots \times C(p^{\beta_l})$, где $\beta_1 \leqslant \beta_2 \leqslant \ldots \leqslant \beta_l$. Ясно, что порядок $|A(p)| = p^{\lambda_1} p^{\lambda_2} \cdot \ldots \cdot p^{\lambda_m} = p^{\beta_1} p^{\beta_2} \cdot \ldots \cdot p^{\beta_l}$.

Запишем разложение группы в уточненном виде

$$A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \ldots \times C(p^{\lambda_{t_1}}) \times \underbrace{C(p) \times C(p) \times \ldots \times C(p)}_{m-t_1}$$

и

$$A(p) = C(p^{\beta_1}) \times C(p^{\beta_2}) \times \ldots \times C(p^{\beta_{t_2}}) \times \underbrace{C(p) \times C(p) \times \ldots \times C(p)}_{l-t_2}.$$

Рассмотрим группу (подгруппу) $[A(p)]^p = \{x^p \mid x \in A(p)\}$. Для этой группы справедливы разложения:

$$[A(p)]^p = C(p^{\lambda_1 - 1}) \times C(p^{\lambda_2 - 1}) \times \ldots \times C(p^{\lambda_{t_1} - 1}), \qquad (8.1)$$

$$[A(p)]^p = C(p^{\beta_1 - 1}) \times C(p^{\beta_2 - 1}) \times \ldots \times C(p^{\beta_{t_2} - 1}), \qquad (8.2)$$

где $C(p^\alpha) = \{x, x^2, \ldots, x^{p^\alpha} = e\}$ и $[C(p^\alpha)]^p = \{x^p, x^{2p}, \ldots, x^{p^\alpha} = e\}$. Порядок $|[A(p)]^p| < |A(p)|$ и, следовательно, по предположению индукции разложения (8.1) и (8.2) совпадают, т. е. $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$, …, $\alpha_{t_1} = \beta_{t_2}$, $t_1 = t_2$, а значит, и $m - t_1 = l - t_2$ или $m = l$. Единственность разложения доказана. □

Пример. Пусть $G$ — коммутативная группа порядка $|G| = 42$. Так как $42 = 2 \cdot 3 \cdot 7$, то группа разложима в произведение следующих своих циклических подгрупп $G = C(2) \times C(3) \times \times C(7)$.

Пример. Пусть $G$ — коммутативная группа порядка $|G| = 4$. Так как $4 = 2^2$, то группа разложима в произведение следующих своих циклических подгрупп $G = C(4)$ или $G = C(2) \times C(2)$ и в явном виде $G = \{x, x^2, x^3, x^4 = e\}$ — циклическая или $G = \{x, y, xy, e\} = \{x, x^2 = e\} \times \{y, y^2 = e\}$.

## § 8.5. Строение некоммутативных групп

Определение. Пусть $G$ — конечная группа. Подгруппа $H \subseteq G$ называется *p-подгруппой*, если порядок ее $|H| = p^\alpha$.

Определение. $p$-подгруппа называется *силовской*, если порядок ее $p^\alpha$ имеет максимальную степень в разложении порядка группы $G$.

Теорема 8.5.1 (Силова). *Пусть $G$ — конечная группа порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, где $p_i$ — простые числа.*

1. *Для каждого $p_i$ существует силовская подгруппа группы $G$.*
2. *Всякая p-подгруппа группы $G$ содержится в некоторой силовской подгруппе.*
3. *Все силовские подгруппы сопряжены, т.е. если $H, P$ — силовские подгруппы, то существует такое $t \in G$, что $H = tPt^{-1}$.*
4. *Количество силовских p-подгрупп равно $p \cdot k + 1$, где $k$ — некоторое целое.*

Задача. Пусть $G$ — группа порядка $|G| = 28 = 2^2 \cdot 7^1$, тогда существуют силовские подгруппы $H_1$, $|H_1| = 4$ и $H_2$, $|H_2| = 7$.

## § 8.6. Симметрическая группа подстановок

Пусть $S$ — конечное множество из $m$ элементов. Множество всех взаимно однозначных отображений множества $S$ на себя называется *симметрической группой* $S_m$ степени $m$. Без ограничения общности можно считать, что множество $S$ состоит из элементов $\{1, 2, \ldots, m\}$. Каждое такое отображение $\pi : S \to S$ называется *подстановкой* или *перестановкой* и записывается $\pi = \begin{pmatrix} 1 & 2 & \ldots m \\ \pi_1 & \pi_2 & \ldots \pi_m \end{pmatrix}$, где $\pi_i = \pi(i)$ — образ элемента $i = \overline{1, m}$. Произведением подстановок является *композиция отображений* (*операция группы*) $(\pi\sigma)(i) = \sigma(\pi(i))$. Например, для подстановок $\pi = \begin{pmatrix} 1\,2\,3 \\ 3\,2\,1 \end{pmatrix}$ и $\sigma = \begin{pmatrix} 1\,2\,3 \\ 2\,1\,3 \end{pmatrix}$ имеем $\pi\sigma = \begin{pmatrix} 1\,2\,3 \\ 3\,1\,2 \end{pmatrix}$ и $\sigma\pi = \begin{pmatrix} 1\,2\,3 \\ 2\,3\,1 \end{pmatrix}$. Данный пример показывает, что симметрическая группа $S_m$ не является абелевой (некоммутативная) при $m \geqslant 3$. Порядок данной группы $|S_m| = m!$ — количество всех перестановок из $m$ эле-

ментов. Единичная (тождественная) подстановка обозначается $e = \begin{pmatrix} 1\,2\ldots m \\ 1\,2\ldots m \end{pmatrix}$, которая удовлетворяет $\forall \pi \in S_m \ \pi e = e\pi$. Обратной к $\pi = \begin{pmatrix} 1\ \ 2\ \ \ldots m \\ \pi_1\,\pi_2\ldots\pi_m \end{pmatrix}$ является подстановка $\pi^{-1} = \begin{pmatrix} \pi_1\,\pi_2\ldots\pi_m \\ 1\ \ 2\ \ \ldots m \end{pmatrix}$, для которой верно, что $\pi\pi^{-1} = \pi^{-1}\pi = e$.

Утверждение 8.6.1. *Симметрическая группа $S_2$ степени 2 — абелева.*

Определение. Подстановка $\pi$, перемещающая элементы $i_1, i_2, \ldots, i_k$ так, что $\pi(i_1) = i_2, \pi(i_2) = i_3, \ldots, \pi(i_k) = i_1$ и оставляющая на месте остальные элементы, называется *циклом длины* $k$ и обозначается $(i_1, i_2, \ldots, i_k)$.

*Равносильное определение*. Подстановка называется *циклической*, если каждый из ее действительно перемещаемых элементов $i_1, i_2, \ldots, i_k$ можно перевести в любой другой (из действительно перемещаемых элементов), если подстановку применить достаточное число раз. Например, $\pi(i_1) = i_2$, $\pi^2(i_1) = \pi(\pi(i_1)) = i_3$, ..., $\pi^{k-1}(i_1) = i_k$, $\pi^k(i_1) = i_1$. Теперь цикл можно записать: $(i_1, i_2, \ldots, i_k) = (\pi(i_1), \pi^2(i_1), \ldots, \pi^{k-1}(i_1))$.

Пример. $\begin{pmatrix} 3\,2\,1\,4\,5\,6\,7 \\ 2\,1\,4\,5\,6\,7\,3 \end{pmatrix} = (3, 2, 1, 4, 5, 6, 7)$ — цикл длины семь.

Определение. Два цикла называются *независимыми*, если они не содержат общих действительно перемещаемых элементов. Например, $(1, 2, 3, 5, 9)$ и $(7, 8)$ — независимые циклы.

Теорема 8.6.1. *Каждую нетождественную подстановку $S_m$ можно разложить единственным образом в произведение независимых циклов.*

Доказательство. Пусть $\pi \in S_m$ и $i, j \in S$. Элементы $i$ и $j$ назовем *эквивалентными* $i \sim j$, если $j = \pi^k(i)$ для некоторого целого числа $k$. Введенное отношение есть отношение эквивалентности на множестве $S$. Оно разбивает множество $S$ на непересекающиеся классы эквивалентности по этому отношению $S = S_1 \cup S_2 \cup \ldots \cup S_r$ (см. п. 7.4). Каждый элемент $i \in S$ принадлежит одному и только одному классу $S_t$, причем множество $S_t$ состоит из образов элемента $i$ при действии степеней подстановки $\pi$: $S_t = \{i, \pi(i), \pi^2(i), \ldots, \pi^{k_t-1}(i)\}$, где $k_t$ — количество

элементов в $S_t$. Множества $S_t$ еще называют $\mathrm{sign}(\pi)$-орбитами. Выберем в каждом классе $S_t$ по одному представителю $i_t$ и поставим ему в соответствие цикл $\pi_t = (i_t, \pi(i_t), \pi^2(i_t), \ldots, \pi^{k_t-1}(i_t))$. Так как любой элемент, не принадлежащий $S_t$, остается на месте при действии степеней $\pi_t$, то перестановка $\pi$ есть произведение независимых циклов $\pi = \pi_1\pi_2 \cdot \ldots \cdot \pi_r$. □

Замечание 1. Если цикл $\pi_t = (i_t)$ имеет длину 1, то он действует как тождественная подстановка. Такие циклы в записи $\pi = \pi_1\pi_2 \cdot \ldots \cdot \pi_r$ можно опускать.

Замечание 2. Независимые циклы в записи $\pi = \pi_1\pi_2 \ldots \pi_r$ можно произвольным образом переставлять между собой. Так, $\pi = \pi_{k_1}\pi_{k_2} \cdot \ldots \cdot \pi_{k_r}$, где $\{\pi_{k_1}, \pi_{k_2}, \ldots, \pi_{k_r}\} = \{\pi_1, \pi_2, \ldots, \pi_r\}$.

Пример. $\begin{pmatrix} 1\,2\,3\,4\,5\,6 \\ 3\,2\,4\,5\,1\,6 \end{pmatrix} = (1345)(2)(6) = (1345)$,

$\begin{pmatrix} 1\,2\,3\,4\,5\,6 \\ 3\,2\,1\,5\,4\,6 \end{pmatrix} = (13)(2)(45)(6) = (13)(45) = (45)(13)$.

Определение. *Декрементом $d$ подстановки* называется разность между числом действительно перемещаемых элементов и числом независимых циклов, на которые она раскладывается. Подстановка называется *четной*, если $d$ — четное число, и подстановка *нечетная*, если $d$ — нечетное. Введем функцию

$$\mathrm{sign}(\pi) = \begin{cases} +1, & \text{если } \pi \text{ — четная подстановка,} \\ -1, & \text{если } \pi \text{ — нечетная подстановка,} \end{cases}$$

где $\pi \in S_m$, тогда $\mathrm{sign}(\pi) = (-1)^d$.

Например, для подстановки $\begin{pmatrix} 1\,2\,3\,4\,5\,6 \\ 3\,1\,2\,5\,4\,6 \end{pmatrix} = (132)(45)$ декремент равен $d = 5 - 2 = 3$, следовательно, подстановка нечетная.

Определение. Цикл длины 2 называется *транспозицией* $\tau = (\alpha\beta)$. Для транспозиции декремент $d = 2 - 1 = 1$ — нечетное число.

Теорема 8.6.2. *При умножении подстановки $\pi \in S_m$ на транспозицию $\tau = (\alpha\beta)$ она меняет свою четность.*

Доказательство. Пусть $\pi = (i_1 i_2 \ldots i_r)\ (j_1 j_2 \ldots j_s) \times \times \ldots \times (l_1 l_2 \ldots l_p)$ — разложение подстановки в произведение независимых циклов. Умножим ее на транспозицию $\tau = (\alpha\beta)$.

Рассмотрим все возможные случаи:
1) $\alpha \notin \pi$, $\beta \notin \pi$;
2) $\alpha \in \pi$, $\beta \notin \pi$;
3) $\alpha \notin \pi$, $\beta \in \pi$;
4) $\alpha \in \pi$, $\beta \in \pi$, $\alpha$ и $\beta$ принадлежат одному и тому же циклу;
5) $\alpha \in \pi$, $\beta \in \pi$, $\alpha$ и $\beta$ принадлежат разным циклам.

Пусть $k$ — число действительно перемещаемых элементов $\pi$; $l$ — число независимых циклов в разложении $\pi$. Декремент подстановки $d(\pi) = k - l$.

1. Пусть $\alpha \notin \pi$, $\beta \notin \pi$, тогда $\pi\tau = (i_1 i_2 \dots i_r)\ (j_1 j_2 \dots j_s) \times \dots \times \times (l_1 l_2 \dots l_p)(\alpha\beta)$. Декремент $d(\pi\tau) = (k+2) - (l+1) = k - - l + 1$.
2. Пусть $\alpha \in \pi$, $\beta \notin \pi$. Будем считать, что $i_1 = \alpha$. В этом случае $\pi\tau = (\alpha i_2 \dots i_r)(j_1 j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots l_p)(\alpha\beta) = = (\alpha i_2 \dots i_r \beta)(j_1 j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = = (k+1) - l = k - l + 1$.
3. Случай $\alpha \notin \pi$, $\beta \in \pi$ рассматривается подобно $\alpha \in \pi$, $\beta \notin \pi$.
4. Пусть $\alpha \in \pi$, $\beta \in \pi$, $\alpha$ и $\beta$ принадлежат одному и тому же циклу. Тогда $\pi\tau = (\alpha i_2 \dots i_m \beta i_{m+1} \dots i_r)(j_1 j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots \dots l_p)(\alpha\beta) = (\alpha i_2 \dots i_m)(\beta i_{m+1} \dots i_r)(j_1 j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = k - (l+1) = k - l - 1$.
5. Пусть $\alpha \in \pi$, $\beta \in \pi$, $\alpha$ и $\beta$ принадлежат разным циклам. Тогда $\pi\tau = (\alpha i_2 \dots i_r)(\beta j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots l_p)(\alpha\beta) = = (\alpha i_2 \dots i_r \beta j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = k - - (l-1) = k - l + 1$.

Таким образом, во всех случаях $\operatorname{sign}(\pi\tau) = (-1)^{k-l\pm1} = = (-1)^{\pm1}(-1)^{k-l} = -\operatorname{sign}(\pi)$ четность подстановки $\pi\tau$ меняется. □

Теорема 8.6.3. *Каждая подстановка $\pi \in S_m$ разлагается в произведение транспозиций не единственным образом, однако четность числа транспозиций постоянна и совпадает с четностью самой подстановки.*

Доказательство. $\pi = (i_1 i_2 \dots i_r)\ (j_1 j_2 \dots j_s) \cdot \dots \cdot (l_1 l_2 \dots \dots l_p)$ — разложение подстановки в произведение независимых циклов. Каждый цикл разлагается в произведение транспозиций $(i_1 i_2 \dots i_r) = (i_1 i_2)(i_1 i_3) \cdot \dots \cdot (i_1 i_r)$. Таким способом можно разложить в произведение транспозиций все циклы подстановки $\pi = \tau_1 \tau_2 \dots \tau_n$, где $\tau_i$ — транспозиции. Теорема 8.6.2 позволяет записать $\operatorname{sign}(\pi) = \operatorname{sign}(\tau_1 \tau_2 \cdot \dots \cdot \tau_n) = \operatorname{sign}((\tau_1 \tau_2 \cdot \dots \cdot \tau_{n-1})\tau_n) = = \operatorname{sign}(\tau_1 \tau_2 \cdot \dots \cdot \tau_{n-1}) \cdot (-1) = \operatorname{sign}(\tau_1 \tau_2 \cdot \dots \cdot \tau_{n-2}) \cdot (-1) \cdot (-1) = = \dots = \operatorname{sign}(\tau_1) \cdot (-1)^{n-1} = (-1)^n$.

Таким образом, четность подстановки $\operatorname{sign}\pi = (-1)^n$ совпадает с четностью числа транспозиций $n$ в разложении $\pi = \tau_1\tau_2 \times \times \ldots \cdot \tau_n$.

□

Следствие 1. $\operatorname{sign}(\sigma\tau) = \operatorname{sign}(\sigma)\operatorname{sign}(\tau)$, *где* $\sigma\tau$ — *произвольные подстановки*.

Доказательство. Пусть $\sigma = s_1s_2 \cdot \ldots \cdot s_r$ и $\tau = t_1t_2 \times \times \ldots \times t_l$ — разложения в произведение транспозиций $s_i, t_j$. Тогда $\sigma\tau = s_1s_2 \cdot \ldots \cdot s_rt_1t_2 \cdot \ldots \cdot t_l$ — разложение в произведение транспозиций. Из теоремы $\operatorname{sign}(\sigma\tau) = (-1)^{r+l} = (-1)^r(-1)^l = = \operatorname{sign}(\sigma)\operatorname{sign}(\tau)$. □

Следствие 2. $\operatorname{sign}(\sigma) = \operatorname{sign}(\sigma^{-1})$, *где* $\sigma^{-1}$ — *обратная подстановка*.

Доказательство. Пусть $\sigma = s_1s_2 \cdot \ldots \cdot s_r$, тогда $\sigma^{-1} = = s_rs_{r-1} \cdot \ldots \cdot s_1$, так как $\sigma\sigma^{-1} = (s_1s_2 \cdot \ldots \cdot s_r)(s_rs_{r-1} \cdot \ldots \cdot s_1) = = (s_1(s_2(\ldots(s_rs_r)\ldots)s_2)s_1) = e$ и $(s_is_i) = e$ — тождественная подстановка, где $s_i$ — транспозиции. Первое следствие позволяет записать $\operatorname{sign}(\sigma\sigma^{-1}) = \operatorname{sign}(\sigma)\operatorname{sign}(\sigma^{-1})$. С другой стороны, $\sigma\sigma^{-1} = e$ — тождественная подстановка и $\operatorname{sign}(e) = 1$. Тогда $\operatorname{sign}(\sigma)\operatorname{sign}(\sigma^{-1}) = 1$ и, следовательно, $\operatorname{sign}(\sigma) = \operatorname{sign}(\sigma^{-1})$. □

Теорема 8.6.4. *Число четных подстановок* $A_m \subseteq S_m$ *равно числу нечетных*.

Доказательство. Достаточно показать, что $|A_m| = = m!/2$, так как $|S_m| = m!$. Для этого установим взаимно однозначное соответствие между четными и нечетными подстановками.

Пусть $\tau$ — произвольная фиксированная транспозиция. Рассмотрим отображение $\varphi : S_m \to S_m$, где $\forall\pi \in S_m\ \varphi(\pi) = \pi\tau$. Пусть $a \in A_m$ — произвольная четная подстановка, тогда $\varphi(a) = = a\tau$ — нечетная подстановка.

*Свойство* 1. Для $\varphi$ верно, что $\forall a_1 \neq a_2 \in A_m\ \varphi(a_1) \neq \varphi(a_2)$, в противном случае $a_1\tau = a_2\tau$. Откуда, умножая справа последнее равенство на $\tau$, получим $a_1(\tau\tau) = a_2(\tau\tau)$, а так как $(\tau\tau) = e$, то $a_1 = a_2$, что противоречит выбору $a_1 \neq a_2$.

*Свойство* 2. Для любой нечетной подстановки $b$ существует прообраз $b\tau \in A_m$ четной подстановки, так как $\varphi(b\tau) = (b\tau)\tau = = b(\tau\tau) = b$.

Свойства 1,2 позволяют утверждать, что отображение является взаимно однозначным. □

Утверждение 8.6.2. *$A_m$ — подгруппа симметрической группы $S_m$.*

Теорема 8.6.5 (Кэли). *Всякая конечная группа $G$ порядка $m$ изоморфна некоторой подгруппе симметрической группы $S_m$.*

Доказательство. Для любого элемента $a \in G$ рассмотрим отображение $L_a : G \to G$, состоящее в умножении всех элементов $G = \{g_1, g_2, \ldots, g_m\}$ слева на $a$: $L_a(g_i) = ag_i$. Свойство группы $aG = G$, позволяет утверждать, что $L_a$ — взаимно однозначное отображение (подстановка). Обратным к $L_a$ будет отображение $L_a^{-1} = L_{a^{-1}}$, единичным отображением является $L_e$. Вследствие ассоциативности умножения в группе $G$ имеем замкнутость: $(L_aL_b)(g) = L_b(L_a(g)) = b(a(g)) = (ab)g = L_{ab}(g)$, т. е. $L_aL_b = L_{ab}$. Отсюда следует, что множество $H = \{L_{g_1}, L_{g_2}, \ldots$ $\ldots, L_{g_m}\}$ образует подгруппу в множестве всех взаимно однозначных отображений $G$ в себя, т. е. в симметрической группе $S_m$. Тогда отображение $\varphi : G \to H \subseteq S_m$ такое, что $\forall a \in$ $\in G \;\; \varphi(a) = L_a$ есть изоморфизм, поскольку $\varphi$ — взаимно однозначное и выполняется свойство $\varphi(a, b) = L_{ab} = L_aL_b = \varphi(a)\varphi(b)$ сохранения операций. □

## § 8.7. Действие групп на множестве

Определение. Говорят, что *задано действие группы $G$* на множестве $S = \{1, 2, \ldots, m\}$, если определен гомоморфизм $T$ группы $G$ в симметрическую группу $S_m$ подстановок: $T : G \to$ $\to S_m$. Свойство сохранения операций для гомоморфизма:

$$\forall g_1, g_2 \in G \; T(g_1g_2) = T(g_1)T(g_2), \text{ где } T(g_i) = \begin{pmatrix} 1 & 2 & \ldots m \\ \alpha_1 & \alpha_2 & \ldots \alpha_m \end{pmatrix} \in S_m.$$

Далее полагаем, что $g_i = \begin{pmatrix} 1 & 2 & \ldots m \\ \alpha_1 & \alpha_2 & \ldots \alpha_m \end{pmatrix}$.

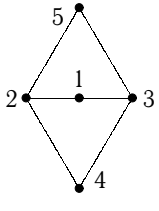


Рис. 8.1

Замечание. Чаще всего бывает так, что действие $G$ на $S$ возникает естественным образом, как группа симметрий структуры, определенной на $S$.

Пример. Пусть $S = \{1, 2, 3, 4, 5\}$ — вершины графа на рис. 8.1. Найти $G$ — группу самосовмещений данного графа.

Р е ш е н и е. Исходное множество элементов $S$ является связанным или структурой. Группа $G$, действующая на $S$, — группа самосовмещений: $G = \{e, a, b, ab\}$, где

$e = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix}$ — тождественное преобразование;

$a = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,5\,4 \end{pmatrix} = (45)$ — поворот вокруг горизонтальной оси;

$b = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,4\,5 \end{pmatrix} = (23)$ — поворот вокруг вертикальной оси;

$ab = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,5\,4 \end{pmatrix} = (23)(45)$ — поворот вокруг горизонтальной оси и вертикальной.

Т а б л и ц а 8.1

| | $e$ | $a$ | $b$ | $ab$ |
|---|---|---|---|---|
| $e$ | $e$ | $a$ | $b$ | $ab$ |
| $a$ | $a$ | $e$ | $ab$ | $b$ |
| $b$ | $b$ | $ab$ | $e$ | $a$ |
| $ab$ | $ab$ | $b$ | $a$ | $e$ |

В табл. 8.1 содержатся все возможные произведения элементов рассматриваемой группы. Из табл. 8.1 видно, что $G$ — коммутативная группа и $G = \{e, a\} \times \{e, b\}$.

О п р е д е л е н и е. Элементы $s_1, s_2 \in S$ называются *g-эквивалентными* и записывают $s_1 \sim s_2$, если $\exists g \in G$, который, действуя на множестве $S$, переводит $s_1$ в $s_2$, т. е. $g = \begin{pmatrix} 1 & 2 & \dots s_1 \dots m \\ \alpha_1 & \alpha_2 & \dots s_2 \dots \alpha_m \end{pmatrix}$ или, более короткая запись: $gs_1 = s_2$.

У т в е р ж д е н и е 8.7.1. *Определенная g-эквивалентность на множестве $S$ является отношением эквивалентности.*

Д о к а з а т е л ь с т в о. Проверим свойства отношения эквивалентности.

1. $\forall s_1 \in S \quad s_1 \sim s_1$. Действительно, $es_1 = s_1$, где $e \in G$ — единичный элемент.
2. $\forall s_1, s_2 \in S \quad s_1 \sim s_2 \to s_2 \sim s_1$. Имеем $s_1 \sim s_2$, тогда $\exists g \in G\, gs_1 = s_2$ и $s_2 = g^{-1}s_1$, а значит, $s_2 \sim s_1$.
3. $\forall s_1, s_2, s_3 \in S \quad s_1 \sim s_2 \wedge s_2 \sim s_3 \to s_1 \sim s_3$. Имеем, что $\exists t, g \in G\, ts_1 = s_2 \wedge gs_2 \sim s_3$, отсюда $(tg)s_1 = g(ts_1) = gs_2 = s_3$. Следовательно, $s_1 \sim s_3$. □

Утверждение 8.7.2. *Группа* $G = \{g_1, g_2, \ldots, g_k\}$, *действуя на множестве* $S$, *порождает его разбиение на непересекающиеся подмножества — классы эквивалентности* (*рис.* 8.2): $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_G}$, *где* $\forall s_1, s_2 \in \widehat{S}_\alpha$ $\exists g \in G \quad gs_1 = s_2$ *и* $\forall s_1 \in \widehat{S}_\alpha \quad \forall s_2 \in \widehat{S}_\beta$ $\forall g \in G \quad gs_1 \neq s_2$, $N_G$ *— количество классов эквивалентности. Пусть* $s_1 \in \widehat{S}_\alpha$, *тогда класс эквивалентности* $\widehat{S}_\alpha$ *составят все различные элементы множества* $\widehat{S}_\alpha = \{g_1 s_1, g_2 s_1, \ldots, g_k s_1\}$.

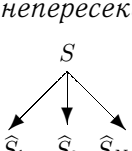


Рис. 8.2

Определение. Множество $Z(s_1) = \{g \in G \mid gs_1 = s_1 \in S\}$ называется *стабилизатором* $s_1 \in S$. Элементы стабилизатора оставляют $s_1$ на месте.

Пример. Продолжим рассмотрение примера на рис. 8.1. $S = \{1, 2, 3, 4, 5\}$ — вершины графа. На $S$ действует группа самосовмещений $G = \{e, a, b, ab\}$, где

$$e = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix},\ a = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,5\,4 \end{pmatrix},\ b = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,4\,5 \end{pmatrix},\ ab = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,5\,4 \end{pmatrix}.$$

Найдем все классы эквивалентности:

$$\widehat{S}_1 = \{e(1), a(1), b(1), ab(1)\} = \{1, 1, 1, 1\} = \{1\},$$
$$\widehat{S}_2 = \{e(2), a(2), b(2), ab(2)\} = \{2, 2, 3, 3\} = \{2, 3\},$$
$$\widehat{S}_3 = \{e(4), a(4), b(4), ab(4)\} = \{4, 5, 4, 5\} = \{4, 5\}.$$

$S = \widehat{S}_1 \cup \widehat{S}_2 \cup \widehat{S}_3$ — всего три класса эквивалентности. Определим стабилизаторы для вершин $\{1, 2, 3, 4, 5\}$ графа:

$$Z(1) = \{e, a, b, ab\},\ Z(2) = Z(3) = \{e, a\},\ Z(4) = Z(5) = \{e, b\}.$$

Утверждение 8.7.3. $Z(s_1) \subseteq G$ *— подгруппа группы* $G$.

Доказательство. Проверим свойства группы.

1. *Замкнутость.* $\forall g_1, g_2 \in Z(s_1)\ gs_1 = s_1, gs_2 = s_2$, тогда и $(g_1 g_2)s_1 = g_2(g_1 s_1) = g_2 s_1 = s_1$, следовательно, $g_1 g_2 \in Z(s_1)$.
2. *Единичный элемент* $e \in Z(s_1)$, так как $es_1 = s_1$.
3. *Обратный элемент.* Пусть $g \in Z(s_1)$, тогда $gs_1 = s_1$, отсюда $s_1 = g^{-1}s_1$, следовательно, $g^{-1} \in Z(s_1)$. □

Утверждение 8.7.4. $\forall s_1, s_2 \in \widehat{S}_\alpha \quad |Z(s_1)| = |Z(s_2)|$ *и* $\exists t \in G\ Z(s_1) = tZ(s_2)t^{-1}$. *В этом случае говорят, что подгруппы* $Z(s_1)$, $Z(s_2)$ *сопряжены.*

Доказательство. Имеем $s_1, s_2 \in \widehat{S}_\alpha$, следовательно, $\exists t \in G\ ts_1 = s_2 \quad \forall g \in Z(s_2) \quad (tg)s_1 = g(ts_1) = gs_2 = s_2 = ts_1$

или $(tg)s_1 = ts_1$. Отсюда $(tgt^{-1})s_1 = t^{-1}((tg)s_1) = t^{-1}(ts_1) = (tt^{-1})s_1 = s_1$, т. е. $tgt^{-1} \in Z(s_1)$. Получили, что $\forall g \in Z(s_2)$ $tgt^{-1} \in Z(s_1)$. Заметим, что $\forall g_1 \neq g_2 \in Z(s_2)$ $tg_1t^{-1} \neq tg_2t^{-1}$, значит, $|Z(s_2)| = |tZ(s_2)t^{-1}| \leqslant |Z(s_1)|$ или $|Z(s_2)| \leqslant |Z(s_1)|$. Подобным образом доказывается в обратную сторону: $|Z(s_2)| \geqslant |Z(s_1)|$, следовательно, $|Z(s_1)| = |Z(s_2)|$. Показали, что $\exists t \in G\ \forall g \in Z(s_2)\ tgt^{-1} \in Z(s_1)$ и $|Z(s_1)| = |Z(s_2)| = |tZ(s_2)t^{-1}|$, отсюда $Z(s_1) = tZ(s_2)t^{-1}$. □

Утверждение 8.7.5. $|\widehat{S}_\alpha| = \dfrac{|G|}{|Z(s_1)|}$, *где* $s_1 \in \widehat{S}_\alpha$.

Доказательство. Пусть $G = \{g_1, g_2, \ldots, g_k\}$. Из *утверждения* 8.7.2 следует, что $\widehat{S}_\alpha = \{g_1s_1, g_2s_1, \ldots, g_ks_1\}$, однако среди выписанных элементов множества $\widehat{S}_\alpha$ могут встречаться одинаковые. Назовем $g_{i_1}, g_{i_2} \in G$ эквивалентными $g_{i_1} \sim g_{i_2}$, если они действуют на элемент $s_1$ одинаковым образом, т. е. $g_{i_1}s_1 = g_{i_2}s_1$. Данное отношение является отношением эквивалентности. Введенная эквивалентность разбивает группу $G$ на классы, количество которых равно числу различных элементов среди выписанных: $\widehat{S}_\alpha = \{g_1s_1, g_2s_1, \ldots, g_ks_1\}$, т. е. равно $|\widehat{S}_\alpha|$.

Пусть $g_{i_1}s_1 = g_{i_2}s_1$ $(g_{i_1} \sim g_{i_2})$ или $(g_{i_1}g_{i_2}^{-1})s_1 = g_{i_2}^{-1}(g_{i_1}s_1) = g_{i_2}^{-1}(g_{i_2}s_1) = (g_{i_2}^{-1}g_{i_2})s_1 = s_1$, следовательно, $g_{i_1}g_{i_2}^{-1} \in Z(s_1)$ или $g_{i_1} \in Z(s_1)g_{i_2}$. Верно и обратное, если $g_{i_1} \in Z(s_1)g_{i_2}$, то $g_{i_1}g_{i_2}^{-1} \in Z(s_1)$ или $(g_{i_1}g_{i_2}^{-1})s_1 = s_1$. Отсюда $g_{i_1}s_1 = g_{i_2}s_1$ или $g_{i_1} \sim g_{i_2}$. Таким образом, $g_{i_1} \sim g_{i_2}$ тогда и только тогда, когда $g_{i_1}, g_{i_2}$ лежат в одном правом классе смежности по стабилизатору $Z(s_1)(g_{i_1} \in Z(s_1)g_{i_2})$, а значит, и число элементов в $\widehat{S}_\alpha$ равно количеству правых смежных классов в $G$ по подгруппе $Z(s_1)$. Согласно теореме 8.3.1 Лагранжа, $|\widehat{S}_\alpha| = \dfrac{|G|}{|Z(s_1)|} = |G : Z(s_1)|$. □

Лемма 8.7.1 (Бернсайда). *Число классов эквивалентности* $\widehat{S}_\alpha$, *на которые распадается множество* $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_G}$ *под действием группы* $G$, *равно* $N_G = \dfrac{1}{|G|}\sum\limits_{g \in G} |\psi(g)|$, *где* $\psi(g) = \{s \in S \,|\, gs = s\}$.

Доказательство. Подсчитаем двумя различными способами множество всех таких пар элементов, что $\forall g \in G\ \forall s \in S$ $\{(g, s) \,|\, gs = s\}$. Это можно выполнить следующим образом:

$\sum\limits_{s\in S}|Z(s)| = \sum\limits_{g\in G}|\psi(g)|$. Первая сумма $\sum\limits_{s\in S}|Z(s)| = \sum\limits_{\alpha=1}^{N_G}\sum\limits_{s\in\widehat{S}_\alpha}|Z(s)|$. Так как $\forall s_1, s_2 \in \widehat{S}_\alpha\ |Z(s_1)| = |Z(s_2)|$, то $\sum\limits_{s\in\widehat{S}_\alpha}|Z(s)| = |\widehat{S}_\alpha| \times$ $\times\, |Z(s_\alpha)| = \dfrac{|G|}{|Z(s_\alpha)|}|Z(s_\alpha)| = |G|$, где $s_\alpha \in \widehat{S}_\alpha$. Таким образом, $\sum\limits_{s\in S}|Z(s)| = \sum\limits_{\alpha=1}^{N_G}\sum\limits_{s\in\widehat{S}_\alpha}|Z(s)| = \sum\limits_{i=1}^{N_G}|G| = N_G\cdot|G|$. Получили, что $N_G\cdot|G| = \sum\limits_{g\in G}|\psi(g)|$, отсюда $N_G = \dfrac{1}{|G|}\sum\limits_{g\in G}|\psi(g)|$. □

Пример. Продолжим рассмотрение примера на рис. 8.1. $S = \{1,2,3,4,5\}$ — вершины графа. На $S$ действует группа самосовмещений $G = \{e, a, b, ab\}$, где $e = \begin{pmatrix}1\,2\,3\,4\,5\\1\,2\,3\,4\,5\end{pmatrix}$, $a = \begin{pmatrix}1\,2\,3\,4\,5\\1\,2\,3\,5\,4\end{pmatrix}$, $b = \begin{pmatrix}1\,2\,3\,4\,5\\1\,3\,2\,4\,5\end{pmatrix}$, $ab = \begin{pmatrix}1\,2\,3\,4\,5\\1\,3\,2\,5\,4\end{pmatrix}$.

Полным перебором установили, что под действием $G$ множество $S$ распадается на три класса эквивалентности: $S = \widehat{S}_1 \cup \widehat{S}_2 \cup$ $\cup\, \widehat{S}_3$, где $\widehat{S}_1 = \{1\}$, $\widehat{S}_2 = \{2,3\}$, $\widehat{S}_3 = \{4,5\}$.

Установим данный факт, применяя лемму Бернсайда 8.7.1:

$$N_G = \frac{1}{|G|}\sum_{g\in G}|\psi(g)| \text{ — число классов эквивалентности,}$$

$$\begin{aligned}\psi(e) &= \{s\in S \mid es = s\} = \{1,2,3,4,5\},\\ \psi(a) &= \{s\in S \mid as = s\} = \{1,2,3\},\\ \psi(b) &= \{s\in S \mid bs = s\} = \{1,4,5\},\\ \psi(ab) &= \{s\in S \mid (ab)s = s\} = \{1\}.\end{aligned}$$

Отсюда следует, что число классов эквивалентности

$$N_G = \frac{1}{|G|}(|\psi(e)| + |\psi(a)| + |\psi(b)| + |\psi(ab)|) = \frac{1}{4}(5+3+3+1) = 3.$$

Замечание. Рассмотрению более содержательных задач, при решении которых возможно применение изложенной выше техники подсчета классов эквивалентности, предварим изложение теории перечисления Пойа. Это позволит нам с более формальных позиций подойти к пониманию самой техники подсчета и к применению ее для решения задач.

## § 8.8. Цикловой индекс группы

Пусть группа $G$ действует на множестве $S = \{1, 2, \ldots, m\}$ и $T : G \to S_m$ — гомоморфизм в симметрическую группу $S_m$. Рассмотрим разложение $g \in G$ на независимые циклы

$$g = \begin{pmatrix} 1 & 2 & \ldots m \\ \alpha_1 & \alpha_2 & \ldots \alpha_m \end{pmatrix} = \underbrace{(\beta_1)(\beta_2) \cdot \ldots \cdot (\beta_{k_1})}_{k_1} \underbrace{(\beta_{i_1})(\beta_{i_2}) \cdot \ldots \cdot}_{k_2} \underbrace{(\beta_{j_1})(\beta_{j_2}) \cdot \ldots \cdot (\beta_{j_m})}_{k_m},$$

где $k_i$ — количество циклов длины $i$; $\quad i = 1, 2, \ldots, m$.

Набор $(k_1, k_2, \ldots, k_m)$ называется *характеристикой элемента* $g \in G$, где $1k_1 + 2k_2 + \ldots + mk_m = m$.

Определение. *Цикловым индексом* $Z(G, x_1, x_2, \ldots, x_m)$ группы $G$, действующей на множестве $S$, называется полином от переменных $x_1, x_2, \ldots, x_m$, определяемый формулой

$$Z(G, x_1, x_2, \ldots, x_m) = \frac{1}{|G|} \sum_{g \in G} x_1^{k_1} x_2^{k_2} \cdot \ldots \cdot x_m^{k_m},$$

где $(k_1, k_2, \ldots, k_m)$ — характеристика элемента $g \in G$.

Пример. Продолжим рассмотрение примера на рис. 8.1. $S = \{1, 2, 3, 4, 5\}$ — вершины графа. На $S$ действует группа самосовмещений $G = \{e, a, b, ab\}$, где $e = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,4\,5 \end{pmatrix}$, $a = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,2\,3\,5\,4 \end{pmatrix}$, $b = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,4\,5 \end{pmatrix}$, $ab = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 1\,3\,2\,5\,4 \end{pmatrix}$.

Найдем цикловой индекс группы $G$, для этого выполним разложение на независимые циклы подстановок элементов $G$ и установим их характеристики:

$$\begin{aligned} e &= (1)(2)(3)(4)(5),\ (k_1, k_2, \ldots, k_m) = (5, 0, 0, 0, 0); \\ a &= (1)(2)(3)(45),\ (k_1, k_2, \ldots, k_m) = (3, 1, 0, 0, 0); \\ b &= (1)(23)(4)(5),\ (k_1, k_2, \ldots, k_m) = (3, 1, 0, 0, 0); \\ ab &= (1)(23)(45),\ (k_1, k_2, \ldots, k_m) = (1, 2, 0, 0, 0); \end{aligned}$$

подставляя в цикловой индекс $Z$, получим

$$Z(G, x_1, x_2, \ldots, x_m) = \frac{1}{4}(x_1^5 + x_1^3 x_2^1 + x_1^3 x_2^1 + x_1^1 x_2^2) = \frac{1}{4}(x_1^5 + 2x_1^3 x_2^1 + x_1^1 x_2^2).$$

## § 8.9. Теория перечисления Пойа

Введем следующие обозначения.

$D = \{d_1, d_2, \ldots, d_m\}$ — конечное множество.
$G = \{g_1, g_2, g_3, \ldots\}$ — конечная группа, действующая на $D$.
$R = \{r_1, r_2, r_3, \ldots\}$ — конечное множество качественных признаков (цвета).
$S = \{f \mid f : D \to R\}$ — множество функций; каждая функция $f$ определяет качественные признаки элементов $D$.
$\Omega = \{\omega_1, \omega_2, \omega_3 \ldots\}$ — множество весов.
$\omega : R \to \Omega$ — весовая функция, назначает веса $\omega(r) \in \Omega$ признакам $r \in R$.
$R(\omega) = \sum\limits_{r \in R} \omega(r)$ — сумма весов элементов $r \in R$ или
$R(\omega) = \sum\limits_{\omega \in \Omega} C_\omega \omega$, где $C_\omega$ — число элементов $r \in R$ с весом $\omega \in \Omega$, множество $\{C_\omega\}$ — перечень $R$ относительно весовой функции $\omega(r)$.
$R(\omega^2) = \sum\limits_{r \in R} \omega^2(r) = \sum\limits_{\omega \in \Omega} C_\omega \omega^2$.
$R(\omega^i) = \sum\limits_{r \in R} \omega^i(r) = \sum\limits_{\omega \in \Omega} C_\omega \omega^i$.

Рассмотрим введенные понятия на следующем примере, к которому ниже не раз будем возвращаться. Пусть $D = \{d_1, d_2, d_3\}$ — вершины правильного треугольника (рис. 8.3). Треугольник нанизан на вертикальную ось, вокруг которой он свободно вращается. $R = \{\bullet, \circ\}$ — множество из двух красок. Найти количество различных раскрасок вершин треугольника. На $D$ действует группа самосовмещений $G = \{e, a, a^2\}$, где $e = \begin{pmatrix} 1\,2\,3 \\ 1\,2\,3 \end{pmatrix}$ — тождественное совмещение, $a = \begin{pmatrix} 1\,2\,3 \\ 2\,3\,1 \end{pmatrix}$ — поворот вокруг оси на 120°, $a^2 = \begin{pmatrix} 1\,2\,3 \\ 3\,1\,2 \end{pmatrix}$ — поворот на 240°.

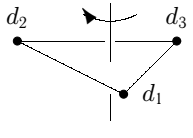


Рис. 8.3

$\Omega = \{x, y, x^{-1}, y^{-1}$ *и их произведения*$\}$ — множество весов. $S = \{f \mid f : D \to R\}$ — множество раскрасок. В треугольнике три вершины и каждую допускается окрашивать в любой из двух цветов $R = \{\bullet, \circ\}$, следовательно, всего функций $2^3$. Такое количество раскрасок будет, если треугольник сделать неподвижным (рис. 8.3). Если допустить вращение, то различные раскраски неподвижного треугольника становятся одинаковыми для

вращающегося треугольника. Приведем пример трех раскрасок: $f_1, f_2, f_3$ (рис. 8.4).

| $f_1 : D \to R$ | $f_2 : D \to R$ | $f_3 : D \to R$ |
|---|---|---|
| $f_1(d_1) = \bullet$ | $f_2(d_1) = \bullet$ | $f_3(d_1) = \circ$ |
| $f_1(d_2) = \bullet$ | $f_2(d_2) = \circ$ | $f_3(d_2) = \bullet$ |
| $f_1(d_3) = \circ$ | $f_2(d_3) = \circ$ | $f_3(d_3) = \circ$ |

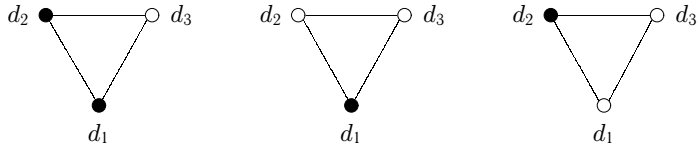



Рис. 8.4. Раскраски треугольника

Очевидно, что для решаемой задачи раскраски $f_2$ и $f_3$ совпадают. Назначим краскам $R = \{\bullet, \circ\}$ веса: $\omega(\bullet) = x$ и $\omega(\circ) = y$.

Замечание. Отметим, что назначенные веса элементов $\omega(\bullet) = x$ и $\omega(\circ) = y$ позволяют сравнивать признаки количественно, забывая, в какой-то степени, о качественном их содержании.

Определение. Для каждой функции $f \in S$ определим *вес*

$$W(f) = \prod_{d \in D} \omega(f(d)).$$

В нашем примере
$$W(f_1) = \omega(\bullet)\omega(\bullet)\omega(\circ) = xxy = x^2y,$$
$$W(f_2) = \omega(\bullet)\omega(\circ)\omega(\circ) = xyy = xy^2,$$
$$W(f_3) = \omega(\circ)\omega(\bullet)\omega(\circ) = yxy = xy^2.$$

Определение. Группа $G$, действуя на $D$, *индуцирует* (наводит, создает, определяет) свое действие на множество функций $S$. Положим $\forall f \in S \forall g \in G\ gf = f(g(d))$ — это рассматривается $\forall d \in D$.

В нашем примере рассмотрим действие элемента $a^2 \in G$ на $f_2 \in S$:

$$f_2(a^2(d_1)) = f_2(d_3) = \circ \quad \text{и} \quad f_3(d_1) = \circ,$$
$$f_2(a^2(d_2)) = f_2(d_1) = \bullet \quad \text{и} \quad f_3(d_2) = \bullet,$$
$$f_2(a^2(d_3)) = f_2(d_2) = \circ \quad \text{и} \quad f_3(d_3) = \circ.$$

Таким образом, элемент группы $a^2$ переводит $f_2$ в $f_3$ или, в принятых обозначениях, $a^2 f_2 = f_3$.

Определение. Положим $f_1 \sim f_2$, если $\exists g \in G \ \forall d \in D$ $f_1(gd) = f_2(d)$. Такие $f_1$ и $f_2$ определяют *одинаковые раскраски* элементов $d \in D$.

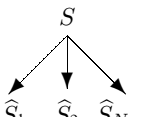


Рис. 8.5

Введенная эквивалентность функций есть *отношение эквивалентности*, которое порождает разбиение множества элементов $f \in F$ на непересекающиеся классы эквивалентности (рис. 8.5): $S = \widehat{S}_1 \cup \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_G}$, где $\forall f_1, f_2 \in \widehat{S}_\alpha \, \exists g \in G \, gf_1 = f_2$ и $\forall f_1 \in \widehat{S}_\alpha \ \forall f_2 \in \widehat{S}_\beta \ \forall g \in G \ \ gf_1 \neq f_2$, $N_G$ — количество классов эквивалентности. Каждый класс эквивалентности определяет отдельную раскраску элементов $d \in D$. Таким образом, количество различных раскрасок равно $N_G$. Для определения $N_G$ воспользуемся леммой 8.7.1 Бернсайда: $N_G = \dfrac{1}{|G|} \sum\limits_{g \in G} |\psi(g)|$, в данном случае $\psi(g) = \{f \in S | \, gf = f\}$ или $f(g(d)) = f(d) \ \forall d \in D$.

Вернемся к нашему примеру на рис. 8.3 и найдем для него число $N_G$. Здесь $\psi(e) = \{f \in S | \, ef = f\}$ удовлетворяют все $f \in S$, отсюда $\psi(e) = 2^3 = 8$. $\psi(a) = \{f \in S | \, af = f\}$ — это такие раскраски вершин, которые допускают совмещение с эквивалентной из раскрасок вращением треугольника на 120°. Это возможно, если все вершины либо белые, либо черные. Итак, $|\psi(a)| = 2$. Подобным образом устанавливается, что и $|\psi(a^2)| = 2$. Следовательно, $N_G = (8 + 2 + 2)/3 = 4$.

Утверждение 8.9.1. *Если $f_1 \sim f_2$, то $W(f_1) = W(f_2)$, т. е. эквивалентные функции имею одинаковые веса.*

Доказательство. $D = \{d_1, d_2, \ldots\}$ и $D = \{gd_1, gd_2, \ldots\}$ — верно $\forall g \in G$, так как $G$ действует на $D$ и $g = \begin{pmatrix} d_1 & d_2 \ldots \\ gd_1 & gd_2 \ldots \end{pmatrix}$ — это подстановка. Теперь $W(f_1) = \prod\limits_{d \in D} \omega(f_1(d)) = \prod\limits_{d \in D} \omega(f_1(gd))$. Имеем $f_1 \sim f_2$, тогда $\exists g \in G \forall d \in D \ \omega(f_1(gd)) = \omega(f_2(d))$. Отсюда $W(f_1) = \prod\limits_{d \in D} \omega(f_1(gd)) = \prod\limits_{d \in D} \omega(f_2(d)) = W(f_2)$. □

Определение. Последнее утверждение позволяет определить *вес* каждого класса эквивалентности $\widehat{S}_i$ в разложении

$S = \widehat{S}_1 \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_G}$ как $W(\widehat{S}_i) = W(f)$, где $f \in \widehat{S}_i$. Определение корректно, так как каждый класс $\widehat{S}_i$ состоит из эквивалентных функций $f$, веса которых совпадают.

Теорема 8.9.1 (Пойа). $\sum\limits_{\omega\in\Omega} C_\omega\omega = Z(G, R(\omega^1), R(\omega^2), \ldots$ $\ldots, R(\omega^m))$, *где* $C_\omega$ *— число классов эквивалентности* $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_G}$ *с весом* $\omega \in \Omega$, $Z(G, x_1^{k_1}, x_2^{k_2}, \ldots, x_m^{k_m})$ *— цикловой индекс группы* $G$, *действующей на множестве* $D = \{d_1, d_2, \ldots, d_m\}$. *Отметим, что* $\sum\limits_{\omega\in\Omega} C_\omega = N_G$.

Доказательство. Пусть $\triangle$ — разбиение множества $D$ на непересекающиеся подмножества:

$$D = \underbrace{D_{11} + D_{12} + \ldots + D_{1k_1}}_{k_1} + \underbrace{D_{21} + D_{22} + \ldots + D_{2k_2}}_{k_2} + \ldots + \underbrace{D_{m1} + D_{m2} + \ldots + D_{mk_m}}_{k_m},$$

где $|D_{ij}| = i$, $|D| = m$, $1k_1 + 2k_2 + \ldots + mk_m = m$.

Определение. Говорят, что $f \in S$ *подчинена разбиению* $\triangle$ множества $D$ и записывают $f \in \triangle$, если $f$ постоянна на каждом подмножестве $D_{ij}$ из разбиения: $\forall d \in D_{ij} f(d) = r_{ij}$, где $r_{ij} \in R$.

Функция $f$, подчиненная разбиению $\triangle$, взаимно однозначно определяется набором

$$(\underbrace{r_{11}r_{12}\ldots r_{1k_1}}_{k_1}\ \underbrace{r_{21}r_{22}\ldots r_{2k_2}}_{k_2}\ \ldots\ \underbrace{r_{m1}r_{m2}\ldots r_{mk_m}}_{k_m}),$$

где $r_{ij} \in R$.

Все такие наборы составляют множество

$$S = \underbrace{S_{11} \times S_{12} \times \ldots \times S_{1k_1}}_{k_1} \times \underbrace{S_{21} \times S_{22} \times \ldots \times S_{2k_2}}_{k_2} \times \ldots \times \underbrace{S_{m1} \times S_{m2} \times \ldots \times S_{mk_m}}_{k_m},$$

где $S_{ij} = R$, $i = \overline{1,m}$, $j = \overline{1,k_i}$. Полагая веса элементов $s_{ij} \in S_{ij}$ равными $\omega(s_{ij}) = [\omega(r_{ij})]^i$ и вес $s = (s_{11}s_{12}\ldots s_{1k_1}\ s_{21}s_{22}\ldots s_{2k_2}\ldots$ $\ldots\ s_{m1}s_{m2}\ldots s_{mk_m}) \in S$ равным произведению весов $\omega(s) =$

$= \prod_{i=1}^{m} \prod_{j=1}^{k_i} \omega(s_{ij}) = \prod_{i=1}^{m} \prod_{j=1}^{k_i} [\omega(r_{ij})]^i$, имеем $\omega(s) = W(f)$, где $f \in \triangle$ — подчинена разбиению.

Для множества $S$ выполнены все условия правила обобщенного произведения, тогда сумма весов элементов данного множества, а значит, и сумма весов всех функций $f \in \triangle$ составит

$$W(S) = \prod_{i=1}^{m} \prod_{j=1}^{k_i} W(S_{ij}) = \prod_{i=1}^{m} [W(S_{ij})]^{k_i},$$

так как

$$W(S_{ij_1}) = W(S_{ij_2}),$$
$$W(S_{ij}) = \sum_{s \in S_{ij}} \omega(s) = \sum_{r \in R} [\omega(r)]^i = R(\omega^i).$$

Следовательно,

$$W(S) = \prod_{i=1}^{m} [W(S_{ij})]^{k_i} = \prod_{i=1}^{m} [R(\omega^i)]^{k_i} =$$
$$= [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \cdot \ldots \cdot [R(\omega^m)]^{k_m}$$

или

$$\sum_{f \in \triangle} W(f) = [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \cdot \ldots \cdot [R(\omega^m)]^{k_m}.$$

Пусть $g \in G$ и

$$g = \underbrace{\beta_1 \beta_2 \ldots \beta_{k_1}}_{k_1} \underbrace{\beta_{i_1} \beta_{i_1} \ldots}_{k_2} \ldots \underbrace{\ldots \beta_{j_1} \beta_{j_2} \ldots \beta_{j_m}}_{k_m}$$

— разложение на независимые циклы, где $(k_1 k_2 \ldots k_m)$ — характеристика $g$ и $1k_1 + 2k_2 + \ldots + mk_m = m$. Запишем разложение на независимые циклы в виде

$$g = \underbrace{D_{11} D_{12} \ldots D_{1k_1}}_{k_1} \underbrace{D_{21} D_{22} \ldots D_{2k_2}}_{k_2} \ldots \underbrace{D_{m1} D_{m2} \ldots D_{mk_m}}_{k_m},$$

где $D_{ij}$ — цикл длины $i$, $|D_{ij}| = i$.

Обозначим разбиение множества $D$ на $D_{ij}$ через $\triangle g$. Элементы $d \in D_{ij}$ одного цикла при умножении на $g$ переходят последовательно по циклу в элементы того же цикла. Указанное свойство позволяет заключить, что $\forall d_1, d_2 \in D_{ij}\ \exists k\ g^k d_1 = d_2$.

Пусть $g = (1,2,3,4,5)(6,7,8)$, тогда $g^2 = (13524)(687)$.

Определение. Функция $f \in S$ называется *неподвижной* относительно $g \in G$, если $\forall d \in D\ f(gd) = f(d)$.

Утверждение 8.9.2. *Функция $f \in S$ неподвижна относительно $g \in G$ тогда и только тогда, когда $f \in \triangle g$ подчинена разбиению $\triangle g$.*

Доказательство. ($\Rightarrow$) Пусть $d_1, d_2 \in D$ — принадлежат одному циклу. Следовательно, $\exists k\ g^k d_1 = d_2$. Тогда $f(d_2) = f(g^k d_1) = f(g^{k-1}(gd_1)) = f(g^{k-1}d_1) = \ldots = f(gd_1) = f(d_1)$, где каждый переход обусловлен неподвижностью $f : \forall d \in D\ f(gd) = f(d)$. Таким образом, $\forall d_1, d_2 \in D_{ij}\ f(d_1) = f(d_2)$, т. е. $f \in \triangle g$.

($\Leftarrow$) Пусть $d \in D_{ij}$, тогда и $gd \in D_{ij}$ — свойство циклов. Имеем $f \in \triangle g$ или $\forall d_1, d_2 \in D_{ij}\ f(d_1) = f(d_2)$, следовательно, и $f(gd) = f(d)$, тогда $f$ — неподвижная относительно $g \in G$. $\square$

Запишем сумму весов $f$, подчиненных разбиению $\triangle g$, в следующем виде $\sum\limits_{f \in \triangle g} \sum\limits_{\omega \in \Omega} a_{\triangle g, \omega}\omega$, где $a_{\triangle g,\omega}$ — количество функций, подчиненных разбиению $\triangle g$ с весом $\omega \in \Omega$.

Составим множество всех функций, неподвижных относительно $g \in G$, с весом $\omega \in \Omega$: $\psi_\omega(g) = \{f \,|\, \forall d \in D\, f(gd) = f(d) \wedge W(f) = \omega\}$. Утверждение 8.9.2 позволяет записать $a_{\triangle g,\omega} = |\psi_\omega(g)|$. Величина $C_\omega$ — число классов эквивалентности с весом $\omega \in \Omega$, на которые распадается множество функций $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \ldots \cup \widehat{S}_{N_g}$ под действием группы $G$. По лемме Бернсайда 8.7.1 $C_\omega = \dfrac{1}{|G|} \sum\limits_{g \in G} |\psi_\omega(g)| = \dfrac{1}{|G|} \sum\limits_{g \in G} a_{\triangle g,\omega}$. Умножив последнее равенство на $\omega$ и просуммировав по всем весам из $\Omega$, получим, что $\sum\limits_{\omega \in \Omega} C_\omega \omega = \dfrac{1}{|G|} \sum\limits_{g \in G} \sum\limits_{\omega \in \Omega} a_{\triangle g,\omega}\omega$.

Сумма $\sum\limits_{\omega \in \Omega} a_{\triangle g,\omega}\omega = \sum\limits_{f \in \triangle g} W(f)$ — сумме весов функций, подчиненных разбиению $\triangle g$. Так как

$$\sum_{f \in \triangle g} W(f) = [R(\omega^1)]^{k_1}[R(\omega^2)]^{k_2} \cdot \ldots \cdot [R(\omega^m)]^{k_m},$$

то

$$\sum_{\omega \in \Omega} C_\omega \omega = \frac{1}{|G|} \sum_{g \in G} [R(\omega^1)]^{k_1}[R(\omega^2)]^{k_2} \cdot \ldots \cdot [R(\omega^m)]^{k_m}.$$

$\square$

## § 8.10. Цикловая структура групп подстановок

Рассмотрим несколько общих примеров, которые в какой-то степени дают представление о возможностях изложенной выше техники подсчета классов эквивалентности с использованием теории перечисления Пойа.

### 8.10.1. Цикловой индекс группы, действующей на себе

Пусть группа $G$ действует на множестве $S = G$ следующим образом: $\forall g \in G\ \forall s \in S\ g(s) = gs$. Если порядок элемента $|g| = d$, то все элементы $gs, g^2s, \ldots, g^ds = s$ различные. Группа $G$ распадается под действием $g$ на циклы одинаковой длины. Количество циклов равно $n/d$, где $n = |G|$ — порядок группы. Цикловой индекс группы примет вид

$$Z(G, x_1, x_2, \ldots, x_n) = \frac{1}{|G|} \sum_{d/n} \gamma(d) x_d^{n/d}, \tag{8.3}$$

где $\gamma(d)$ — количество элементов $g \in G$ с порядком $|g| = d$. Суммирование выполняется по всем делителям $d$ числа $n$.

### 8.10.2. Цикловой индекс циклической группы

Пусть $S = \{0, 1, \ldots, n-1\}$ — множество вершин правильного $n$-угольника (рис. 8.6) на плоскости. Группа $G$ — циклическая группа самосовмещений. Образующий элемент группы $a \in G$ — вращение (отображение) $S$ вокруг центра на угол $2\pi/n$.

Следовательно, $G = \{a^n = e, a, a^2, \ldots, a^{n-1}\}$, где $e$ — тождественная подстановка. Если обозначить вершины многоугольника $0, 1, \ldots, n-1$ элементами группы $\{a^n = e, a, a^2, \ldots, a^{n-1}\}$, то вращение будет эквивалентно умножению вершины на соответствующий элемент группы. Например, совмещение при вращении на угол $2\pi/n$ равносильно умножению новых меток $\{a^n = e, a, a^2, \ldots, a^{n-1}\}$ вершин $n$-угольника на элемент $a$; действительно, $a \cdot e \to a$, $a \cdot a \to a$, $a \cdot a^2 \to a^3, \ldots a \cdot a^{n-1} \to a^n$.

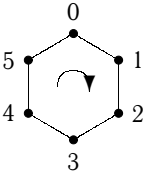


Рис. 8.6

Таким образом, можно считать, что циклическая группа $G$ действует на себе. Согласно (8.3) цикловой индекс данной группы примет вид

$$Z(G, x_1, x_2, \ldots, x_n) = \frac{1}{|G|} \sum_{d/n} \gamma(d)\, x_d^{n/d} = \frac{1}{|G|} \sum_{d/n} \varphi(d)\, x_d^{n/d}, \tag{8.4}$$

где $\gamma(d) = \varphi(d)$ — функция Эйлера. Действительно, если порядок элемента равен $d$ (элемент суммируется в $\gamma(d)$), то он является образующим элементом циклической подгруппы $H \subseteq G$ порядка $|H| = d$. Показать, что если порядки элементов циклической группы совпадают: $|a^k| = |a^m| = d$, то они являются образующими одной и той же подгруппы $H \subseteq G$. Число образующих в такой подгруппе $\varphi(d)$.

Найдем количество возможных раскрасок $N_G$ двумя цветами вершин данного (рис. 8.6) $n$-угольника. Пусть цвета — $R = \{\bullet, \circ\}$. Воспользуемся теоремой 8.9.1 Пойа. Для определения количества раскрасок положим веса цветов равными $\omega(\bullet) = 1$ и $\omega(\circ) = 1$. Тогда $R(\omega^k) = \omega^k(\bullet) + \omega^k(\circ) = 2$,

$$N_G = \sum_{\omega\in\Omega} C_\omega = Z(G, R(\omega^1), R(\omega^2), \ldots, R(\omega^n)) =$$

$$= \frac{1}{n}\sum_{d/n} \varphi(d)[R(\omega^d)]^{n/d} = \frac{1}{n}\sum_{d/n} \varphi(d)2^{n/d}.$$

Для треугольника ($n = 3$) число раскрасок

$$N_G = \frac{1}{n}\sum_{d/n} \varphi(d)2^{n/d} = \frac{1}{3}(\varphi(1)2^3 + \varphi(3)2^1) = \frac{1}{3}(1\cdot 2^3 + 2\cdot 2^1) = 4.$$

Для шестиугольника ($n = 6$) число раскрасок

$$N_G = \frac{1}{n}\sum_{d/n} \varphi(d)2^{n/d} = \frac{1}{6}(\varphi(1)2^6 + \varphi(2)2^3 + \varphi(3)2^2 + \varphi(6)2^1) =$$

$$= \frac{1}{6}(1\cdot 2^6 + 1\cdot 2^3 + 2\cdot 2^2 + 2\cdot 2^1) = 14.$$

### 8.10.3. Цикловой индекс симметрической группы

Пусть $S = \{1, 2, \ldots, n\}$ — произвольное множество, на котором действует группа всех подстановок. $S_n$ — симметрическая группа, $|S_n| = n!$. Рассмотрим произвольную подстановку $\pi \in S_n$ с характеристикой $(k_1, k_2, \ldots, k_n)$, где $1k_1 + 2k_2 + \ldots + nk_n = n$. Подсчитаем количество подстановок с данной характеристикой. Для этого запишем $\pi$ в цикловом представлении, помещая знаки $(-)$ на местах, которые должны занять $n$ элементов. Так, для характеристики $(3, 2)$ следует записать $(-)(-)(-)(--)(--)$. Далее пробелы можно заполнять $n$ элементами множества $S$ в любом порядке. Это дает $n!$ подстановок с характеристикой $(k_1, k_2, \ldots, k_n)$, которые, однако, не все различные. Каждый из

циклов $k_r$ может начинаться с любого своего элемента, не изменяя исходной подстановки и уменьшая общее число различных подстановок в $r$ раз. Если имеется $k_r$ таких циклов, то их можно переставлять $k_r!$ способами, что также не будет приводить к новым подстановкам. Все эти варианты суть различные обозначения одной и той же подстановки с характеристикой $(k_1, k_2, \ldots, k_n)$. Поэтому различных подстановок с указанной характеристикой будет $\dfrac{n!}{k_1!\,1^{k_1} k_2!\,2^{k_2} \cdot \ldots \cdot k_n!\,n^{k_n}}$. Тогда цикловой индекс симметрической группы можно записать в следующем виде:

$$Z(G, x_1, x_2, \ldots, x_n) =$$
$$= \frac{1}{n!}\Bigg( \sum_{1k_1+2k_2+\ldots+nk_n=n} \frac{n!}{k_1!\,1^{k_1} k_2!\,2^{k_2} \cdot \ldots \cdot k_n! n^{k_n}}\, x_1^{k_1} x_2^{k_2} \cdot \ldots \cdot x_n^{k_n} \Bigg) =$$
$$= \sum_{1k_1+2k_2+\ldots+nk_n=n} \frac{1}{k_1!\,k_2! \cdot \ldots \cdot k_n!} \left(\frac{x_1}{1}\right)^{k_1} \left(\frac{x_2}{2}\right)^{k_2} \cdot \ldots \cdot \left(\frac{x_n}{n}\right)^{k_n}.$$

Рассмотрим пример определения числа $N_G$ раскрасок двумя цветами $R = \{\bullet, \circ\}$ трех усов $S = \{1, 2, 3\}$ антенны (рис. 8.7). Усы свободно вращаются в пространстве. В этом случае на множестве $S = \{1, 2, 3\}$ действует симметрическая группа подстановок $S_3$. Найдем все характеристики $(k_1, k_2, k_3)$ разложения числа $3 = 1k_1 + 2k_2 + 3k_3$. Ими будут: $(3, 00)$, $(1, 1, 0)$ и $(0, 0, 1)$. Цикловой индекс примет вид

$$Z(G, x_1, x_2, x_3) = \sum_{1k_1+2k_2+3k_3=3} \frac{1}{k_1!k_2!k_3!} \left(\frac{x_1}{1}\right)^{k_1} \left(\frac{x_2}{2}\right)^{k_2} \left(\frac{x_3}{3}\right)^{k_3} =$$
$$= \frac{1}{3!}\left(\frac{x_1}{1}\right)^3 + \frac{1}{1!1!}\left(\frac{x_1}{1}\right)^1 \left(\frac{x_2}{2}\right)^1 + \frac{1}{1!}\left(\frac{x_3}{3}\right)^1.$$

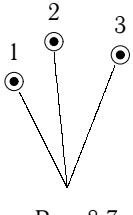


Рис. 8.7

Для определения количества раскрасок положим веса цветов равными $\omega(\bullet) = 1$ и $\omega(\circ) = 1$, тогда $R(\omega^k) = \omega^k(\bullet) + \omega^k(\circ) = 2$. Следовательно,

$$N_G = \sum_{\omega \in \Omega} C_\omega = Z(G, R(\omega^1), R(\omega^2), \ldots, R(\omega^n)) =$$
$$= \frac{1}{3!}\left(\frac{2}{1}\right)^3 + \frac{1}{1!1!}\left(\frac{2}{1}\right)^1 \left(\frac{2}{2}\right)^1 + \frac{1}{1!}\left(\frac{2}{3}\right)^1 = 4.$$

Глава 9

# ЭЛЕМЕНТЫ ТЕОРИИ ЧИСЕЛ

Теория чисел занимается изучением свойств целых чисел. Целыми называются числа $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$. Для любых $a, b \in Z$ сумма $a + b$, разность $a - b$ и произведение $a \cdot b$ являются целыми числами. Но частное $a/b$ от деления $a$ на $b$ (если $b$ не равно нулю) может быть как целым, так и не целым. В случае, когда частное $a/b$ является целым, то обозначают $a = bq$, где $q$ — целое число, $b$ называют делителем числа $a$ и записывают так: $b/a$. В общем случае единственным является представление $a = bq + r$, $0 \leqslant r < b$, где $r$ — называют остатком от деления.

## § 9.1. Наибольший общий делитель

В дальнейшем будем рассматривать лишь положительные делители чисел. Всякое целое, делящее одновременно целые $a, b, \dots, c$, называется их *общим делителем*. Наибольший из общих делителей называется *наибольшим общим делителем* (НОД) и обозначается $(a, b, \dots, c)$. Если $(a, b, \dots, c) = 1$, то $a, b, \dots$ $\dots, c$ называются *взаимно простыми*. Например, $(10, 15) = 5$, $(8, 21) = 1$.

### Свойства наибольшего общего делителя

1. Если $a = bq$, то $(a, b) = b$.
2. Если $a = bq + r$, тогда общие делители чисел $a$ и $b$ суть те же, что и общие делители чисел $b$ и $r$; в частности, $(a, b) = (b, r)$.
3. Для определения наибольшего общего делителя применяется *алгоритм Евклида*. Он состоит в нижеследующем. Пусть $a$ и $b$ — положительные целые числа и $a > b$. Составим ряд равенств:

$$\begin{aligned}
a &= bq_1 + r_2, & 0 < r_2 < b,\\
b &= r_2q_2 + r_3, & 0 < r_3 < r_2,\\
r_2 &= r_3q_3 + r_4, & 0 < r_4 < r_3,\\
&\dots & \dots\\
r_{n-2} &= r_{n-1}q_{n-1} + r_n, & 0 < r_n < r_{n-1},\\
r_{n-1} &= r_nq_n,
\end{aligned} \tag{9.1}$$

заканчивающийся, когда получается некоторое $r_{n+1} = 0$. Последнее неизбежно случится, так как ряд $b, r_2, r_3, \ldots$ убывающих целых чисел не может содержать более чем $b$ положительных.

4. Из формул (9.1) алгоритма Евклида следует, что

$$(a, b) = (b, r_2) = (r_2, r_3) = \ldots = (r_{n-1}, r_n) = r_n.$$

Наибольший общий делитель равен последнему не равному нулю остатку алгоритма Евклида.

5. Из формул (9.1) алгоритма Евклида следует также, что существуют целые $t_1$ и $t_2$, что $t_1 a + t_2 b = r_n$. В частности, если $(a, b) = 1$, то $t_1 a + t_2 b = 1$.

Пример. Найдем $(525, 231)$.
$525 = 231 \cdot 2 + 63,$
$231 = 63 \cdot 3 + 42,$
$63 = 42 \cdot 1 + 21,$
$42 = 21 \cdot 2.$
Последний остаток есть 21, значит, НОД$= (525, 231) = 21$.

6. Пусть $m$ — любое положительное целое, тогда $(am, bm) = (a, b)m$.
7. Если $(a, b) = 1$, то $(ac, b) = (c, b)$.
8. Если $(a, b) = 1$ и $ac$ делится на $b$, то $c$ делится на $b$.

## § 9.2. Наименьшее общее кратное

Всякое целое, кратное всех данных чисел, называется их *общим кратным*. Наименьшее положительное общее кратное называется *наименьшим общим кратным* (НОК). Наименьшее общее кратное двух чисел $a$ и $b$ равно их произведению, деленному на их общий наибольший делитель, т. е. $\dfrac{a \cdot b}{(a, b)}$.

## § 9.3. Простые числа

1. Число 1 имеет только один положительный делитель, именно 1. В этом отношении число 1 в ряде натуральных чисел стоит совершенно особо. Всякое целое, большее 1, имеет не менее двух делителей, именно 1 и самого себя; если других делителей нет, то число называется *простым*. Целое, большее 1, имеющее кроме 1 и самого себя другие положительные делители, называют *составным*.

2. Наименьший отличный от единицы делитель целого, большего единицы, есть число простое. В противном случае, можно было бы выбрать делитель еще меньше.
3. Наименьший отличный от единицы делитель (он будет простым) составного числа $a$ не превосходит $\sqrt{a}$. Действительно, пусть $q$ — именно этот делитель, тогда $a = qb$ и $b \geqslant q$ ( $q$ — наименьший делитель), откуда $a \geqslant q^2$ или $q \leqslant \sqrt{a}$.
4. *Простых чисел бесконечно много.* Это следует из того, что каковы бы ни были различные простые числа $p_1, p_2, \ldots, p_k$, можно получить новое простое, среди них не находящееся. Таковым будет простой делитель суммы $p_1 p_2 \cdot \ldots \cdot p_k + 1$, который, деля всю сумму, не может совпадать ни с одним из простых $p_1, p_2, \ldots, p_k$.
5. *Решето Эратосфена* для составления таблицы простых чисел.

   Данный способ состоит в следующем.

   Выписываем числа: 1, 2, 3, ~~4~~, 5, 6, 7, ~~8~~, 9, ~~10~~, 11, ... , $N$.

Первое, большее 1, число этого ряда есть 2. Оно делится только на себя и на 1 и, следовательно, оно простое. Вычеркиваем из ряда (как составные) все числа, кратные 2, кроме самого себя. Первое, следующее за 2 невычеркнутое число, есть 3 — оно также будет простым. С ним, как и с числом 2, проделываем ту же процедуру и т. д. Если указанным способом уже вычеркнуты все числа, кратные простым, меньшим $p$ ($\sqrt{a} < p$), то все невычеркнутые, меньшие $p^2$, будут простые.

*Составление таблицы простых чисел, не превосходящих $N$, закончено, как только вычеркнуты все составные кратные простых, не превосходящих $\sqrt{N}$.*

В алгоритме 9.1 реализовано решето Эратосфена для нечетных чисел $2N + 1$ с *предпросеиванием* для двойки; другими словами, начинаем только с нечетных чисел, отсеивая кратные 3, 5, 7, 11 и т. д. Вектор $X$ является двоичным набором индикаторов простых нечетных чисел. Так, элемент вектора $X_k = 1$, если соответствующее ему число $2k + 1$ — простое; в противном случае, когда $X_k = 0$, число $2k + 1$ — непростое.

Алгоритм 9.1.

$X = \{1, 1, \ldots, 1\}$;
for $k = 3$ to $\sqrt{2N+1}$ by 2 do begin
  $\{X_{(k-1)/2} \qquad k\}$
  $\{ \quad k\}$

```
    if X_{(k-1)/2} = 1 then
        for i = k^2 to 2N + 1 by 2k do X_{(k-1)/2} = 0;
end;
{ }
for k = 1 to N do if X_k = 1 then  2k + 1.
```

Рабочая программа на языке Pascal реализации решета Эратосфена генерации простых чисел приводится в алгоритме 8.2. Алгоритм 8.3 — это пример реализации алгоритма решета Эратосфена на языке C.

Алгоритм 9.2. Pascal

```
Program PgmPrimary; {   }
uses CRT;
Const
  N=45; {                    }
Type
  Vector=array[1..N] of Boolean;
Var
  f :Text; {   }
  X :Vector; {   }

Procedure Primary; { }
Var
  i,k,M,nm :Integer;
begin
  nm:=2*N+1;
  for k:=1 to N do X[k]:=TRUE;
  k:=3;
  M:=trunc(sqrt(nm));
  while k<=M do begin
    if X[(k-1) div 2] then begin
      i:=k*k; {     }
      while i<=nm do begin
        X[(i-1) div 2]:=FALSE;
        i:=i+2*k;
      end;
    end;
    k:=k+2; {     }
```

```
  end;
  {          }
  i:=0;
  for k:=1 to N do
    if X[k] then begin
      i:=i+1;
      Write(f,2*k+1,' ');
    end;
  WriteLn(f);
  WriteLn(f,'Количество простых нечетных чисел '+
      'в диапазоне [3,',2*N+1,'] равно ',i);
end;
begin
  Assign(f,'Primary.out'); {    }
  Rewrite(f);
  Primary;
  Close(f);
end.
```

Алгоритм 9.3. C

```
#include<stdio.h>
#include <math.h>
#define N 45
  char X[N+1];

int main(void){
  FILE *f;
  int i,k,M,nm;
  f=fopen("primary.out","wt");
  nm=2*N+1;
  M=sqrt(nm);
  for( k=1; k<=N; k++ ) X[k]=1;
  for( k=3; k<=M; k+=2 )
    if( X[(k-1)/2] ) for( i=k*k; i<=nm; i+=2*k ) X[(i-1)/2]=0;

  i=0;
  for( k=1; k<=N; k++ ) if( X[k] ){ i++; fprintf(f,"%d ",2*k+1); }
```

```
  fprintf(f,"\nКоличество простых нечетных чисел
          в диапазоне [3,%d] равно %d",2*N+1,i);
  fclose(f);
  return 0;
}
```

Исходными данными для программ алгоритмов 9.2 и 9.3 является верхняя граница $2N+1$ диапазона $[3, 2N+1]$ поиска простых чисел. Значение $2N+1$ этой границы устанавливается явным образом в программе посредством присваивания соответствующего значения переменной $N$. Нижняя граница диапазона всегда принимается равной 3 — первое нечетное простое число.

Результаты расчетов по программам алгоритмов 9.2 и 9.3 сохраняются в выходном файле Primary.out со следующей структурой:

3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89.
Количество простых нечетных чисел в диапазоне [3, 91] равно 23.

В первой строке приводятся все вычисленные простые нечетные числа из диапазона $[3, 2N+1]$, границы которого распечатываются во второй строке результирующего файла. Во второй же строке указывается и общее количество найденных простых чисел.

## § 9.4. Сравнения, свойства сравнений

Пусть $m$ — целое положительное число, которое назовем *модулем*. Будем говорить, что целые числа $a$ и $b$ сравнимы по модулю $m$, если $a - b = t \cdot m$ для некоторого целого $t$, т. е. равны остатки от деления $a$ и $b$ на $m$. Сравнение чисел $a$ и $b$ по модулю $m$ будем записывать $a \equiv b\,(mod\ m)$. Например, $77 \equiv 5\,(mod\ 8)$, $102 \equiv 0\,(mod\ 3)$.

### Свойства сравнений

1. $a \equiv a\,(mod\ m)$ — свойство рефлексивности.
2. $a \equiv b\,(mod\ m) \Rightarrow b \equiv a\,(mod\ m)$ — свойство симметричности.
3. $a \equiv b\,(mod\ m) \wedge b \equiv c\,(mod\ m) \Rightarrow a \equiv c\,(mod\ m)$ — свойство транзитивности.
4. $a \equiv b\,(mod\ m) \Rightarrow (a, m) = (b, m)$.
5. $a \equiv b\,(mod\ m) \wedge c \equiv d\,(mod\ m) \Rightarrow a + c \equiv b + d\,(mod\ m)$.
6. $a \equiv b\,(mod\ m) \wedge c \equiv d\,(mod\ m) \Rightarrow a \cdot c \equiv b \cdot d\,(mod\ m)$.
7. $a = a_1 d,\quad b = b_1 d,\quad (d, m) = 1, a \equiv b\,(mod\ \ m) \Rightarrow a_1 \equiv \equiv b_1\,(mod\ m)$.

8. $a \equiv b\,(mod\ m),\ m_1 \backslash m\ \ (m_1$ делит $m) \Rightarrow a \equiv b\,(mod\ m_1)$.
9. $a \equiv b\,(mod\ m), d \backslash a, d \backslash b, d \backslash m\ \Rightarrow a_1 \equiv b_1\,(mod\ m_1)$

## § 9.5. Полная система вычетов

Свойства 1–3 сравнений показывают, что операция сравнения целых чисел по модулю $m$ является *отношением эквивалентности*. Множество всех целых чисел $Z$ разбивается на *классы эквивалентности* (рис. 9.1), которые называются *вычетами по модулю* $m$. Числа $r, s \in Z$ лежат в одном классе $k_i$, т. е. $r, s \in \{k_i\}$ тогда и только тогда, когда $r \equiv s\,(mod\ m)$ имеют одинаковые остатки от деления на $m$. Числа одного и того же класса имеют с модулем $m$ один и тот же наибольший общий делитель, так как из $r, s \in \{k_i\}$ следует, что $(r, m) = (s, m)$ (см. п. 9.4). Особенно важны классы, для которых этот делитель равен единице, т. е. классы, содержащие числа, взаимно простые с модулем.

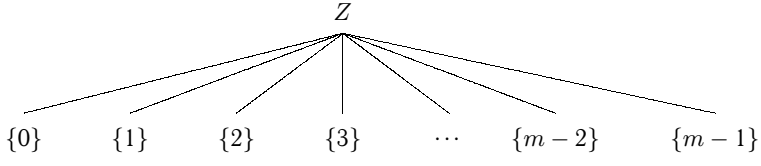



Рис. 9.1. Полная система вычетов

Определение. Множество классов вычетов $\{0\}, \{1\}, \ldots, \{m-1\}$ по модулю $m$ называется *полной системой вычетов*. Полную систему вычетов можно получить следующим образом. Пусть $Z$ — аддитивная (операция сложения) группа целых чисел, $mZ$ — подгруппа всех чисел, кратных $m$. Тогда факторгруппа $Z/mZ$ — *аддитивная группа вычетов по модулю* $m$ (полная система вычетов).

## § 9.6. Приведенная система вычетов

Определение. Пусть $m$ — целое положительное число. Множество классов из полной системы вычетов $\{0\}, \{1\}, \ldots, \{m-1\}$, взаимно простых с $m$, называется *приведенной системой вычетов*, которую будем обозначать как $M_\pi$ или $M_\pi(m)$. Приведенную систему вычетов, следовательно, можно составить из чисел полной системы вычетов, взаимно простых с модулем.

Обыкновенно $M_\pi$ выделяют из системы наименьших неотрицательных вычетов: 0, 1, 2, ..., $m-1$.

Утверждение 9.6.1. *$M_\pi$ является группой с операцией умножения.*

Доказательство. Проверим все свойства (аксиомы) группы.

1. *Замкнутость.* Пусть произвольные $a, b \in M_\pi$. Покажем, что $ab \in M_\pi$. По условию $(a, m) = 1$ и $(b, m) = 1$, тогда $(ab, m) = 1$.
2. *Ассоциативность* операции умножения чисел выполняется.
3. *Единичный элемент* — 1.
4. *Существование обратного элемента.*

Возьмем произвольный элемент $a \in M_\pi$. Покажем совпадение множеств $aM_\pi = M_\pi$. Ясно, что $aM_\pi \subseteq M_\pi$, так как для $M_\pi$ выполняется свойство замкнутости. Для доказательства $aM_\pi = M_\pi$ теперь достаточно показать, что все элементы множества $aM_\pi$ различны. Предположим, что существуют $b_1 \neq b_2 \in M_\pi$, для которых $ab_1 - ab_2 \equiv 0\,(mod\ m)$. Отсюда $a(b_1 - b_2) \equiv 0\,(mod\ m)$. А так как $(a, m) = 1$ — взаимно простые, то $b_1 - b_2 \equiv 0\,(mod\ m)$ или $b_1 \equiv b_2\,(mod\ m)$, что противоречит принадлежности их разным классам.

Следовательно, все элементы множества $aM_\pi$ различны, значит, $\exists b \in M_\pi$, что $a \cdot b \equiv 1\,(mod\ m)$. Элемент $b$ является обратным к $a$, и по доказательству он единственный такой элемент (*существование обратного элемента $a^{-1}$ доказано*). Таким образом, получили, что $M_\pi$ является группой по умножению. *Порядок этой группы равен количеству чисел меньших $m$ и взаимно простых с ним.* □

## § 9.7. Функция Эйлера

Определение. *Функция Эйлера* $\varphi(m)$ определяется для всех целых положительных $m$ и равна количеству чисел ряда

$$1,\ 2,\ \ldots,\ m-1,$$

взаимно простых с $m$, где число 1 полагается взаимно простым с любым из чисел и $\varphi(1) = 1$.

Пример. $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 2$, $\varphi(5) = 4$, $\varphi(6) = 2$, $\varphi(7) = 6$, $\varphi(8) = 4$, $\varphi(9) = 6$, $\varphi(10) = 4$, $\varphi(11) = 10$.

Замечание. Отметим, что порядок группы $M_\pi(m)$ приведенной системы вычетов по модулю $m$ равен $|M_\pi(m)| = \varphi(m)$.

### Свойства функции Эйлера

Свойство 1. *Если* $(m_1, m_2) = 1$, *то*
$$\varphi(m_1 \cdot m_2) = \varphi(m_1)\varphi(m_2).$$

Доказательство (1 способ). Пусть $C(m_1m_2)$ — циклическая группа порядка $|C(m_1m_2)| = m_1m_2$, число образующих ее равно $\varphi(m_1m_2)$ (см. утверждение 8.3.4). Так как $(m_1, m_2) = 1$, то допустимо разложение (см. п. 8.4) группы $C(m_1m_2)$ в прямое произведение своих циклических подгрупп $C(m_1m_2) = C(m_1) \times C(m_2)$ и, следовательно, число образующих группы $C(m_1m_2)$ равно $\varphi(m_1)\varphi(m_2)$. □

Доказательство (2 способ). Достаточно показать, что
$$|M_\pi(m_1m_2)| = |M_\pi(m_1)| \cdot |M_\pi(m_2)|.$$

1. Заметим, что из $(m_1, m_2) = 1$ следует существование целых $a$ и $b$, для которых выполняется $am_2 + bm_1 = 1$ (алгоритм Евклида см. п. 9.1). Приведем значения целых $a$ и $b$ к значениям $a \in \{1, 2, \ldots, m_1\}$ и $b \in \{1, 2, \ldots, m_2\}$, для которых $am_2 + bm_1 \equiv 1\,(mod\ m_1m_2)$. Пусть $c \in \{1, 2, \ldots, m_1m_2\}$, тогда верно $cam_2 + cbm_1 \equiv c\,(mod\ m_1m_2)$, где значения $ca$ и $cb$ приведены к значениям $a \in \{1, 2, \ldots, m_1\}$ и $b \in \{1, 2, \ldots, m_2\}$. Таким образом, произвольное число $c \in \{1, 2, \ldots, m_1m_2\}$ можно записать в виде $am_2 + bm_1 \equiv c\,(mod\ m_1m_2)$, где $a \in \{1, 2, \ldots, m_1\}$ и $b \in \{1, 2, \ldots, m_2\}$. Данное представление является *однозначным*, так как число возможных пар $(a, b)$ равно $m_1m_2$ и таково же количество представляемых чисел $c \in \{1, 2, \ldots, m_1m_2\}$. Далее рассмотрим все представления $am_2 + bm_1$ для всех $a \in \{1, 2, \ldots$ $\ldots, m_1\}$ и $b \in \{1, 2, \ldots, m_2\}$.

2. Пусть $a \in M_\pi(m_1)$ и $b \in M_\pi(m_2)$, тогда $(a, m_1) = 1$ и $(b, m_2) = 1$ и $(m_1, m_2) = 1$. Покажем, что $(am_2 + bm_1, m_1m_2) =$ $= 1$. Выражение $(am_2 + bm_1, m_1m_2) = 1$ эквивалентно $(am_2 +$ $+ bm_1, m_1) = 1$ и $(am_2 + bm_1, m_2) = 1$. В первом случае — $(am_2 + bm_1, m_1) = (am_2, m_1) = (a, m_1) = 1$ и во втором — $(am_2 + bm_1, m_2) = (bm_1, m_2) = (b, m_2) = 1$. Таких пар $(a, b)$, а значит, и чисел $am_2 + bm_1$, взаимно простых с $m_1m_2$, равно $\varphi(m_1)\varphi(m_2)$. Покажем, что других чисел взаимно простых с $m_1m_2$ среди $am_2 + bm_1$ нет.

3. Остались не рассмотренными числа $am_2 + bm_1$, где $a \notin M_\pi(m_1)$ или $b \notin M_\pi(m_2)$, для них $(am_2 + bm_1, m_1m_2) \neq 1$, т. е. такие числа не являются взаимно простыми с $m_1m_2$.

4. Из пунктов 1), 2), 3) следует, что $\varphi(m_1m_2) = \varphi(m_1)\varphi(m_2)$. □

Свойство 2. $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^\alpha\left(1 - \frac{1}{p}\right)$, *где* $p$ — *простое число и* $\alpha > 0$.

Доказательство. Числа вида $k{\cdot}p$, где $k \in \{1, 2, \ldots, p^{\alpha-1}\}$ — это все числа, не взаимно простые с $p^\alpha$ среди $1, 2, \ldots, p^\alpha$, а значит, остальные являются взаимно простыми с $p^\alpha$. Отсюда $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$. □

Свойство 3. *Пусть разложение числа* $m$ *на простые сомножители имеет вид* $m = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$, *где* $p_i$ — *простые числа, тогда* $\varphi(m) = \prod\limits_{i=1}^{k} \varphi(p_i^{\alpha_i}) = \prod\limits_{i=1}^{k} p_i^{\alpha_i}\left(1 - \frac{1}{p_i}\right) =$

$$= m \prod_{i=1}^{k}\left(1 - \frac{1}{p_i}\right).$$

Свойство 4. $\sum\limits_{d\backslash n} \varphi(d) = n$, *где* $d$ — *различные делители числа* $n$.

Доказательство (1 способ). Пусть $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k}$ — разложение числа на простые сомножители. Правило обобщенного произведения (см. п. 4.4) позволяет записать:

$$\sum_{d\backslash n} \varphi(d) = \sum_{(r_1 r_2 \ldots r_k)} \varphi(p_1^{r_1} p_2^{r_2} \cdot \ldots \cdot p_k^{r_k}) =$$

$$= \sum_{(r_1 r_2 \ldots r_k)} \varphi(p_1^{r_1})\varphi(p_2^{r_2}) \cdot \ldots \cdot \varphi(p_k^{r_k}) =$$

$$= \prod_{i=1}^{k} \sum_{r_i=0}^{\alpha_i} \varphi(p_i^{r_i}) = \prod_{i=1}^{k} \sum_{d\backslash p_i^{\alpha_i}} \varphi(d).$$

Сумма

$$\sum_{d\backslash p^\alpha} \varphi(d) = \varphi(p^0) + \varphi(p^1) + \ldots + \varphi(p^\alpha)$$

$$1 + (p-1) + (p^2 - p) + \ldots + (p^\alpha - p^{\alpha-1}) = p^\alpha.$$

Отсюда искомая сумма

$$\sum_{d\backslash n} \varphi(d) = \prod_{i=1}^{k} \sum_{d\backslash p_i^{\alpha_i}} \varphi(d) = \prod_{i=1}^{k} p_i^{\alpha_i} = n.$$

Доказательство (2 способ). Пусть $C(n)$ — циклическая группа порядка $|C(n)| = n$. Для всякого делителя $d$ числа $n$ существует единственная подгруппа $C(d) \subseteq C(n)$ порядка $|C(d)| = d$. Образующие группы $C(d)$ составляют множество $S_d = \{x \in C(n) \,|\, |x| = d\}$. Из утверждения 8.3.4 следует, что число образующих группы $C(d)$ равно $\varphi(d) = |S_d|$. Так как $S_d \cap S_{d'} = \varnothing$ для $d \neq d'$, то $\bigcup\limits_{d\backslash n} S_d = C(n)$ или $\sum\limits_{d\backslash n} \varphi(d) = n$. □

Теорема 9.7.1 (Эйлера). *$x^{\varphi(m)} \equiv 1\,(mod\ m)$, где $m > 0$ и $(x, m) = 1$.*

Доказательство. Приведенная система вычетов $M_\pi$ по модулю $m$ является группой, порядок которой $|M_\pi| = \varphi(m)$. Пусть $x$ — произвольное, такое, что $(x, m) = 1$ и $x \equiv r\,(mod\ m)$, где $r \in M_\pi$. Из свойств сравнений следует, что наибольший общий делитель $(r, m) = 1$. Теорема 8.3.1 Лагранжа утверждает, что порядок элемента группы кратен порядку этой группы. Пусть $k$ — порядок элемента $r$, т.е. $r^k \equiv 1\,(mod\ m)$. Отсюда $\varphi(m) = k \cdot d$, где $d$ — положительное целое. Тогда $r^{\varphi(m)} \equiv r^{kd} \equiv 1\,(mod\ m)$. Из $x \equiv r\,(mod\ m)$ следует, что $x^{\varphi(m)} \equiv r^{\varphi(m)}\,(mod\ m)$, а значит, и $x^{\varphi(m)} \equiv 1\,(mod\ m)$. □

Теорема 9.7.2 (Ферма). *$x^p \equiv x\,(mod\ p)$, где $p$ — простое число; $x$ — произвольное целое положительное число.*

Доказательство. Пусть $x \equiv 0\,(mod\ p)$, тогда и $x^p \equiv 0\,(mod\ p)$. Пусть теперь $x \equiv r\,(mod\ p)$, где $r \in \{1, 2, \ldots, p-1\}$ и, значит, $(r, p) = 1$. Из теоремы 9.7.1 Эйлера следует, что $x^{\varphi(p)} \equiv 1\,(mod\ p)$, где $\varphi(p) = p - 1$. Отсюда $x^{p-1} \equiv 1\,(mod\ p)$ и $x^p \equiv x\,(mod\ p)$. □

Теорема 9.7.3 (Вильсона). *$(p-1)! + 1 \equiv 0\,(mod\ p)$, где $p$ — простое число.*

Доказательство. Пусть $M_\pi = \{1, 2, \ldots, p-1\}$ — приведенная система вычетов по модулю $p$. $M_\pi$ является группой. Для любого $x \in \{1, 2, \ldots, p-1\}$ существует единственный обратный элемент $y \in \{1, 2, \ldots, p-1\}$ такой, что $x \cdot y \equiv 1\,(mod\ p)$.

Заметим, что $x \cdot x \equiv 1\,(mod\ p)$ выполняется только для двух элементов: $x = 1$ и $x = p - 1$. Действительно, $x^2 - 1 = (x-1) \times (x+1) \equiv 0\,(mod\ p)$ равносильно $x - 1 \equiv 0\,(mod\ p)$ или $x + 1 \equiv 0\,(mod\ p)$. Отсюда $x = 1$ или $x = p - 1$. Таким образом, обратными элементами к себе являются только $x = 1$

и $x = p - 1$. Для любого из оставшихся элементов группы $x \in \{2, 3, \ldots, p-2\}$ существует единственный обратный $y \in \{2, 3, \ldots, p-2\}$ такой, что $xy \equiv 1\,(mod\ p)$ и $x \neq y$. Тогда верно $2 \cdot 3 \cdot \ldots \cdot (p-2) \equiv 1\,(mod\ p)$. Умножив последнее сравнение на $1 \cdot (p-1)$, получим

$$1 \cdot 2 \cdot \ldots \cdot (p-1) \equiv -1\,(mod\ p) \text{ или } (p-1! + 1 \equiv 0\,(mod\ p)). \square$$

З а д а ч а. Пусть $p$ — простое и $h_1, h_2, \ldots, h_n$ — целые числа. Доказать, что $(h_1 + h_2 + \ldots + h_n)^p \equiv h_1^p + h_2^p + \ldots + h_n^p\,(mod\ p)$.

Р е ш е н и е. Согласно теореме Ферма 9.7.2 $h_i^p \equiv h_i\,(mod\ p)$, $i = \overline{1, n}$. Свойства операции сравнения (см. п. 9.4) позволяют записать данные $n$ сравнений $h_i^p \equiv h_i\,(mod\ p)$ в виде их суммы: $h_1^p + h_2^p + \ldots + h_n^p \equiv h_1 + h_2 + \ldots + h_n\,(mod\ p)$. С другой стороны, верно и $(h_1 + h_2 + \ldots + h_n)^p \equiv h_1 + h_2 + \ldots + h_n\,(mod\ p)$, что непосредственно вытекает из теоремы Ферма.

## § 9.8. Функция Мёбиуса. Формула обращения Мёбиуса

О п р е д е л е н и е. *Функция Мёбиуса* $\mu(n)$ определяется для всех целых положительных $n$ и равна

$$\mu(n) = \begin{cases} 1, & \text{если } n = 1, \\ 0, & \text{если } n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k} \text{ и } \exists \alpha_i > 1, \\ (-1)^k, & \text{если } n = p_1 p_2 \ldots p_k, \end{cases}$$

где $n = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ — разложение на простые сомножители, $p_i$ — простые числа, $\alpha_i$ — кратность $p_i$ в разложении.

П р и м е р.

$$\begin{array}{llllll} \mu(1) = 1, & \mu(2) = -1, & \mu(3) = -1, & \mu(4) = 0, & \mu(5) = -1, & \\ \mu(6) = 1, & \mu(7) = -1, & \mu(8) = 0, & \mu(9) = 0, & \mu(10) = 1, & \\ \mu(11) = -1, & \mu(12) = 0, & \mu(13) = -1, & \mu(14) = 1, & \mu(15) = 1, & \\ \mu(16) = 0, & \mu(17) = -1, & \mu(18) = 0, & \mu(19) = -1, & \mu(20) = 0, & \\ \mu(21) = 1. & & & & & \end{array}$$

Л е м м а 9.8.1. $\sum\limits_{d \backslash n} \mu(d) = \begin{cases} 0, & \textit{если } n > 1, \\ 1, & \textit{если } n = 1, \end{cases}$

*где суммирование идет по всем делителям $d$ числа $n$.*

Доказательство. Если $n = 1$, то $\sum\limits_{d\backslash 1} \mu(d) = \mu(1) = 1$. Пусть теперь $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \ldots \cdot p_k^{\alpha_k} \neq 1$ — разложение на простые сомножители. Тогда

$$\sum_{d\backslash n} \mu(d) = \sum_{d\backslash n,\, \mu(d)\neq 0} \mu(d) = \sum_{r=0}^{k} C_k^r (-1)^r = (1-1)^k = 0.$$

Все делители $d$, для которых $\mu(d) \neq 0$, имеют вид $p_{i_1} p_{i_2} \cdot \ldots \cdot p_{i_r}$ и $\mu(p_{i_1} p_{i_2} \cdot \ldots \cdot p_{i_r}) = (-1)^r$. Количество таких делителей $p_{i_1} p_{i_2} \cdot \ldots \cdot p_{i_r}$, выбираемых из $p_1 p_2 \cdot \ldots \cdot p_k$, равно числу сочетаний $C_k^r$. □

Теорема 9.8.1. *Формула обращения Мёбиуса*:

$$\text{если } f(n) = \sum_{d\backslash n} g(d), \text{ то } g(n) = \sum_{d\backslash n} \mu(d) f\left(\frac{n}{d}\right),$$

*где* $f(n)$, $g\left(\dfrac{n}{d}\right)$ — *функции, определенные для всех целых положительных* $n$.

Доказательство. Выполним подстановку $f\left(\dfrac{n}{d}\right)$ в сумме $\sum\limits_{d\backslash n} \mu(d) f\left(\dfrac{n}{d}\right) = \sum\limits_{d\backslash n} \mu(d) \Big( \sum\limits_{\delta\backslash \frac{n}{d}} g(\delta) \Big)$. Заметим, что здесь число $n$ неявно рассматривается в виде произведения $n = d \cdot \delta \cdot r$, где делители $d$ и $\delta$ принимают все допустимые значения независимо друг от друга и порядок суммирования не влияет на значение суммы, т. е.

$$\sum_{d\backslash n} \mu(d) \Big( \sum_{\delta\backslash \frac{n}{d}} g(\delta) \Big) = \sum_{\delta\backslash n} g(\delta) \Big( \sum_{d\backslash \frac{n}{\delta}} \mu(d) \Big),$$

где $\sum\limits_{d\backslash \frac{n}{\delta}} \mu(d) = \begin{cases} 0, & \text{если } \dfrac{n}{\delta} \neq 1, \\ 1, & \text{если } \delta = n, \end{cases}$ — вследствие леммы 9.8.1. Тогда

$$\sum_{\delta\backslash n} g(\delta) \Big( \sum_{d\backslash \frac{n}{\delta}} \mu(d) \Big) = g(n) \sum_{d\backslash \frac{n}{n}} \mu(d) = g(n) \sum_{d\backslash 1} \mu(d) =$$

$$= g(n)\mu(1) = g(n).$$

□

З а д а ч а. Установить связь функции Эйлера $\varphi(n)$ и функции Мёбиуса $\mu(n)$.

Р е ш е н и е. $\sum\limits_{d\backslash n}\varphi(d)=n$ — свойство 4 функции Эйлера (см. п. 9.7). К данной сумме применим формулу обращения Мёбиуса:

$$\varphi(n)=\sum_{d\backslash n}\mu(d)\frac{n}{d}=n\sum_{d\backslash n}\frac{\mu(d)}{d}\ \text{и, наконец,}\ \frac{\varphi(n)}{n}=\sum_{d\backslash n}\frac{\mu(d)}{d}.$$

Для $n=6$ имеем $\varphi(6)=2$, все делители $d\in\{1,\,2,\,3,\,6\}$, $\mu(1)=1$, $\mu(2)=-1$, $\mu(3)=-1$, $\mu(6)=1$ и, наконец, выражение $\dfrac{\varphi(n)}{n}=\sum\limits_{d\backslash n}\dfrac{\mu(d)}{d}$ примет вид $\dfrac{2}{6}=\dfrac{1}{1}+\dfrac{-1}{2}+\dfrac{-1}{3}+\dfrac{1}{6}$. Пусть $n=$ $=p_1^{\alpha_1}p_2^{\alpha_2}\dots p_k^{\alpha_k}$ — разложение на простые множители. Так как $\varphi(n)=n\prod\limits_{i=1}^{k}\left(1-\dfrac{1}{p_i}\right)$ — свойство 3 функции Эйлера (см. п. 9.7), то

$$n\prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right)=n\sum_{d\backslash n}\frac{\mu(d)}{d}\ \text{или}\ \prod_{i=1}^{k}\left(1-\frac{1}{p_i}\right)=\sum_{d\backslash n}\frac{\mu(d)}{d}.$$

## Задачи и упражнения

### Системы счисления

**1.** Выполнить перевод чисел из одной системы счисления в другую. Найти $x_2, y_8, z_{16}$:

1). $354_{10} = x_2 = y_8 = z_{16}$.
2). $801_{10} = x_2 = y_8 = z_{16}$.
3). $832_{10} = x_2 = y_8 = z_{16}$.
4). $132_{10} = x_2 = y_8 = z_{16}$.
5). $900_{10} = x_2 = y_8 = z_{16}$.
6). $354_{10} = x_2 = y_8 = z_{16}$.
7). $137_{10} = x_2 = y_8 = z_{16}$.

1a). $0.131_{10} = x_2 = y_8 = z_{16}$.
2a). $0.018_{10} = x_2 = y_8 = z_{16}$.
3a). $0.018_{10} = x_2 = y_8 = z_{16}$.
4a). $0.780_{10} = x_2 = y_8 = z_{16}$.
5a). $0.411_{10} = x_2 = y_8 = z_{16}$.
6a). $0.873_{10} = x_2 = y_8 = z_{16}$.
7a). $0.731_{10} = x_2 = y_8 = z_{16}$.

**2.** Выполнить перевод чисел из одной системы счисления в другую. Найти $x_{10}$ и $y$:

1). $3231.3_4 = x_{10}$.
2). $10121.1_3 = x_{10}$.
3). $632.7_8 = x_{10}$.
4). $244.2_5 = x_{10}$.
5). $101.21_4 = x_{10}$.
6). $342.6_7 = x_{10}$.
7). $181.2_9 = x_{10}$.

1a). $534.6_{10} = y_7$.
2a). $394.2_{10} = y_3$.
3a). $811.4_{10} = y_4$.
4a). $379.1_{10} = y_6$.
5a). $227.7_{10} = y_5$.
6a). $557.8_{10} = y_7$.
7a). $238.9_{10} = y_8$.

### Введение в математическую логику

**3.** Проверить, какие из функций являются тождественно истинными, условно истинными, тождественно ложными:

1). $f = (x \to y) \to ((x \vee z) \to (y \vee z))$.
2). $f = ((x \oplus y) \sim z) \cdot (x \to y \cdot z)$.
3). $f = ((x \vee \overline{y}) \downarrow (x \oplus \overline{y})) \oplus (\overline{x \to \overline{y}} \to (x \vee y))$.
4). $f = ((x \vee \overline{y}) \cdot z \to ((x \sim z) \oplus y)) \cdot x \cdot y \cdot z$.
5). $f = (x \nrightarrow y) \nrightarrow ((x \vee z) \nrightarrow (y \vee z))$.
6). $f = ((x \oplus y) \sim z) \cdot (x \nrightarrow y \vee z)$.
7). $f = ((x \vee \overline{y}) \mid (x \oplus \overline{y})) \oplus (x \to \overline{y} \nrightarrow (x \vee y))$.

**4.** Для следующих булевых функций составить таблицы истинности, построить их СДНФ, СКНФ и полиномы Жегалкина:

1). $f(x, y, z) = \neg(yx \vee z) \vee x \oplus y | x$.
2). $f(x, y, z) = x \oplus xz | \neg(xy) \oplus y$.

3). $f(x,y,z) = (x \cdot \neg y) \vee y \to (x \sim y) \oplus z$.
4). $f(x,y,z) = x \sim y \sim z \oplus \neg(xy)|x$.
5). $f(x,y,z) = \neg(xy) \vee \neg(z) \downarrow x|z$.
6). $f(x,y,z) = \neg(\neg(x)y\neg(z)) \oplus xy \sim x|y$.
7). $f(x,y,z) = x \sim xy \oplus (x \downarrow z) \vee y$.

**5.** Минимизировать следующие булевы функции трех переменных двумя способами: (1) геометрическим на трехмерном кубе и (2) аналитическим методом Куайна:

1). $\overline{x}\,\overline{y}\,\overline{z} \vee x\,y\,\overline{z} \vee x\,\overline{y}\,z \vee \overline{x}\,\overline{y}\,z \vee \overline{x}\,y\,z \vee x\,y\,z$.
2). $x\,\overline{y}\,\overline{z} \vee \overline{x}\,\overline{y}\,\overline{z} \vee \overline{x}\,y\,\overline{z} \vee x\,y\,\overline{z} \vee x\,\overline{y}\,z \vee \overline{x}\,\overline{y}\,z$.
3). $x\,y\,\overline{z} \vee x\,y\,z \vee \overline{x}\,y\,z \vee \overline{x}\,\overline{y}\,\overline{z} \vee x\,\overline{y}\,\overline{z}$.
4). $\overline{x}\,y\,\overline{z} \vee x\,y\,z \vee \overline{x}\,y\,z \vee \overline{x}\,\overline{y}\,z \vee x\,\overline{y}\,z \vee x\,\overline{y}\,\overline{z}$.
5). $x\,y\,\overline{z} \vee \overline{x}\,y\,\overline{z} \vee \overline{x}\,y\,z \vee x\,y\,z \vee x\,\overline{y}\,z \vee \overline{x}\,\overline{y}\,\overline{z}$.
6). $\overline{x}\,\overline{y}\,\overline{z} \vee x\,y\,z \vee \overline{x}\,y\,z \vee \overline{x}\,\overline{y}\,z \vee x\,\overline{y}\,z \vee x\,\overline{y}\,\overline{z} \vee x\,y\,\overline{z}$.
7). $\overline{x}\,y\,\overline{z} \vee \overline{x}\,\overline{y}\,\overline{z} \vee x\,y\,z \vee \overline{x}\,y\,z \vee x\,\overline{y}\,z \vee x\,\overline{y}\,\overline{z}$.

**6.** Минимизировать следующие булевы функции четырех переменных двумя способами: (1) геометрическим на карте Карно и (2) аналитическим методом Куайна:

1). $\overline{x}\,y\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,\overline{w} \vee x\,\overline{y}\,z\,w \vee \overline{x}\,y\,z\,w \vee \overline{x}\,\overline{y}\,z\,w \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee \overline{x}\,y\,\overline{z}\,\overline{w} \vee \overline{x}\,\overline{y}\,\overline{z}\,\overline{w}$.
2). $x\,\overline{y}\,z\,\overline{w} \vee \overline{x}\,y\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,\overline{w} \vee \overline{x}\,y\,z\,w \vee x\,y\,\overline{z}\,w \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee x\,y\,\overline{z}\,\overline{w} \vee \overline{x}\,\overline{y}\,\overline{z}\,\overline{w}$.
3). $x\,y\,z\,w \vee x\,\overline{y}\,z\,\overline{w} \vee x\,\overline{y}\,z\,w \vee \overline{x}\,\overline{y}\,z\,w \vee x\,\overline{y}\,\overline{z}\,w \vee \overline{x}\,y\,\overline{z}\,w \vee \overline{x}\,\overline{y}\,z\,w \vee \overline{x}\,\overline{y}\,\overline{z}\,\overline{w}$.
4). $x\,y\,z\,\overline{w} \vee \overline{x}\,y\,z\,\overline{w} \vee x\,\overline{y}\,z\,w \vee \overline{x}\,\overline{y}\,z\,w \vee x\,\overline{y}\,\overline{z}\,w \vee \overline{x}\,\overline{y}\,\overline{z}\,w \vee x\,y\,\overline{z}\,\overline{w} \vee \overline{x}\,y\,\overline{z}\,\overline{w}$.
5). $\overline{x}\,\overline{y}\,z\,\overline{w} \vee \overline{x}\,y\,z\,\overline{w} \vee x\,y\,z\,w \vee x\,\overline{y}\,z\,w \vee x\,\overline{y}\,\overline{z}\,w \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,w \vee \overline{x}\,y\,\overline{z}\,\overline{w}$.
6). $\overline{x}\,y\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,w \vee \overline{x}\,\overline{y}\,z\,\overline{w} \vee x\,y\,z\,w \vee x\,\overline{y}\,\overline{z}\,w \vee x\,y\,\overline{z}\,w \vee \overline{x}\,y\,\overline{z}\,w \vee x\,y\,\overline{z}\,\overline{w}$.
7). $\overline{x}\,\overline{y}\,z\,\overline{w} \vee \overline{x}\,y\,z\,w \vee \overline{x}\,\overline{y}\,z\,w \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee x\,\overline{y}\,\overline{z}\,w \vee x\,y\,\overline{z}\,w \vee \overline{x}\,y\,\overline{z}\,w \vee \overline{x}\,\overline{y}\,\overline{z}\,w$.
8). $x\,\overline{y}\,z\,\overline{w} \vee x\,y\,z\,\overline{w} \vee \overline{x}\,y\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,\overline{w} \vee x\,\overline{y}\,\overline{z}\,\overline{w} \vee x\,y\,\overline{z}\,\overline{w} \vee \overline{x}\,y\,\overline{z}\,\overline{w} \vee \overline{x}\,\overline{y}\,\overline{z}\,\overline{w}$.
9). $\overline{x}\,\overline{y}\,\overline{z} \vee x\,y\,z \;\vee \overline{x}\,\overline{z}\,\overline{w} \vee y\,z\,w \;\vee x\,z\,\overline{w} \vee x\,\overline{y}\,\overline{w} \vee \overline{x}\,\overline{y}\,z\,w \;\vee \overline{x}\,y\,z\,\overline{w}$.
10). $y\,\overline{z}\,\overline{w} \vee x\,y\,\overline{z} \vee \overline{y}\,\overline{z}\,w \;\vee x\,\overline{y}\,w \;\vee \overline{x}\,y\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,\overline{z}\,w$.

**7.** Проверить, что наборы булевых функций $\sigma$ — полные. Для каждого такого набора $\sigma$ получить функции: $\overline{x}, x \cdot y, x \vee y$. Для найденных выражений выполнить проверку — составить таблицы истинности:

1). $\sigma = \{\, 0, x \to y \,\}$.
2). $\sigma = \{\, 1, x \nrightarrow y \,\}$.
3). $\sigma = \{\, \overline{x}, x \to y \,\}$.
4). $\sigma = \{\, \overline{x}, x \nrightarrow y \,\}$.
5). $\sigma = \{\, x \to y, x \nrightarrow y \,\}$.
6). $\sigma = \{\, x \sim y, xy, 0 \,\}$.
7). $\sigma = \{\, x \sim y, x \vee y, 0 \,\}$.
8). $\sigma = \{\, x \oplus y, xy, 1 \,\}$.
9). $\sigma = \{\, x \oplus y, x \vee y, 1 \,\}$.

**8.** Проверить, являются ли полными следующие наборы $\sigma$ булевых функций?

1). $\sigma = \{ xy \oplus z,\ (x \sim y) \oplus z) \}$.
2). $\sigma = \{ x \to y,\ \neg(x \oplus y \oplus z) \}$.
3). $\sigma = \{ x \to y,\ x \to \overline{y}\, z \}$.
4). $\sigma = \{ 0, \overline{x}, f_1, f_2, f_3 \}$, где $f_1$, $f_2$, $f_3$ — попарно различные булевы функции 2-х переменных.
5). $\sigma = \{ x \to y,\ (1100001100111100) \}$, где (1100001100111100) — таблица истинности функции.
6). $\sigma = \{ (1011),\ (1111110011000000) \}$, где (1011), (1111110011000000) — таблицы истинности функций.
7). $\sigma = \{ x \nrightarrow y,\ x \nrightarrow \overline{y}\, z \}$.

**9.** Построить для $f$ полином Жегалкина:
1). $f = \overline{x}y \vee \overline{y}z \vee \overline{z}w \vee \overline{w}x$.
2). $f = (x \to y)(y \to x) \to z$.
3). $f = xy \vee yz \vee zw \vee wx$.
4). $f = (x \to y)(y \to x) \sim z$.
5). $f = xy \to yz \to zw \to wx$.
6). $f = \overline{x}y \oplus \overline{y}z \oplus \overline{z}w \oplus \overline{w}x$.
7). $f = xy \nrightarrow yz \nrightarrow zw \nrightarrow wx$.

**10.** Какие из перечисленных функций являются монотонными:
1). $f = xy(x \oplus y)$.
2). $f = xy \oplus yz \oplus zx \oplus z$.
3). $f = x \to (x \to y)$.
4). $f = x \to (y \to x)$.
5). $f = x \nrightarrow (x \nrightarrow y)$.
6). $f = x \nrightarrow (y \nrightarrow x)$.
7). $f = xy \vee yz \vee zx \to z$.

**11.** Проверить, являются ли функции самодвойственными:
1). $f = \overline{x} \vee y \vee \overline{z}$.
2). $f = (x \to y) \to \overline{x \cdot z} \to (y \to z)$.
3). $f = x \to (y \to (z \to (w \to 0)))$.
4). $f = x \nrightarrow (y \nrightarrow (z \nrightarrow (w \nrightarrow 1)))$.
5). $f = x_1 \oplus x_2 \oplus \ldots \oplus x_n \oplus 1$.
6). $f = \overline{x}_1 \oplus \overline{x}_2 \oplus \ldots \oplus \overline{x}_{2n} \oplus 1$.
7). $f = x_1 \oplus x_2 \oplus \ldots \oplus x_{2n+1} \oplus \sigma$, где $\sigma \in \{0, 1\}$.

**12.** Булева функция $f(x, y)$ удовлетворяет соотношению $f(f(x, f(y, z)), f(f(x, y), f(x, z))) = 1$.
Доказать справедливость следующих соотношений:
1). $f(x, x) = 1$.
2). $f(x, f(y, x)) = 1$.
3). $f(f(f(x, y), f(x, z)), f(x, f(y, z))) = 1$.
4). $f(f(x, y), f(f(x, f(y, z)), f(x, z))) = 1$.
5). $f(f(x, f(y, z)), f(y, f(x, z))) = 1$.

**13.** Каково число функций $f(x_1, x_2, \ldots, x_n)$, принимающих значение 1 менее, чем на $k$ наборах?

**14.** Пусть $f(x_1, x_2, \ldots, x_n)$ и $g(y_1, y_2, \ldots, y_m)$ — булевы функции, переменные которых $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_m$ не пересекаются. Известно, что СДНФ функции $f(x_1, x_2, \ldots, x_n)$ содержит $r$ слагаемых, а СДНФ функции $g(y_1, y_2, \ldots, y_m)$ содержит $s$ слагаемых. Найти длину СДНФ функций:

1). $f(x_1, x_2, \ldots, x_n) \wedge g(y_1, y_2, \ldots, y_m)$.
2). $f(x_1, x_2, \ldots, x_n) \vee g(y_1, y_2, \ldots, y_m)$.
3). $f(x_1, x_2, \ldots, x_n) \oplus g(y_1, y_2, \ldots, y_m)$.
4). $f(x_1, x_2, \ldots, x_n) \to g(y_1, y_2, \ldots, y_m)$.
5). $f(x_1, x_2, \ldots, x_n) \nrightarrow g(y_1, y_2, \ldots, y_m)$.
6). $f(x_1, x_2, \ldots, x_n) \downarrow g(y_1, y_2, \ldots, y_m)$.
7). $f(x_1, x_2, \ldots, x_n) \mid g(y_1, y_2, \ldots, y_m)$.
8). $f(x_1, x_2, \ldots, x_n) \sim g(y_1, y_2, \ldots, y_m)$.

**15.** Найти число слагаемых в СДНФ булевых функций или (тоже самое) число наборов значений переменных $x_1$, $x_2$, ..., $x_n$, на которых функции обращаются в 1:

1). $f(x_1, x_2, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus x_n$.
2). $f(x_1, x_2, \ldots, x_n) = (x_1 \vee x_2 \vee \ldots \vee x_n) \cdot (\overline{x}_1 \vee \overline{x}_2 \vee \ldots \vee \overline{x}_n)$.
3). $f(x_1, x_2, \ldots, x_n) = (x_1 \oplus x_2 \oplus x_3) \cdot (\overline{x}_1 \oplus \overline{x}_2 \oplus \overline{x}_3) \oplus \oplus x_4 \oplus \ldots \oplus x_n$.
5). $f(x_1, x_2, \ldots, x_n) = x_1 x_2 \ldots x_k \oplus x_{k+1} x_{k+2} \ldots x_n$.
6). $f(x_1, x_2, \ldots, x_n) = 1 \oplus x_1 \oplus x_1 x_2 \oplus \ldots \oplus x_1 x_2 \ldots x_n$.
7). $f(x_1, x_2, \ldots, x_n) = x_1 \nrightarrow (x_2 \nrightarrow (\ldots (x_{n-1} \nrightarrow x_n)))$.

**16.** Построить вывод формул в системе аксиом Клини:

1). $\neg C \supset \neg D$, $\neg B \supset \neg C$, $D \vdash B$.
2). $\neg B \supset \neg A \vdash B \supset A$.
3). $\neg A$, $A \vdash B$.
4). $A \supset B, B \supset C \vdash A \supset C$.
5). $A \supset (B \supset C) \vdash B \supset (A \supset C)$.
6). $A \supset (B \supset C) \vdash A \wedge B \supset C$.
7). $A \wedge B \supset C \vdash A \supset (B \supset C)$.
8). $A \supset B \vdash \neg B \supset \neg A$

Доказать формулы в системе аксиом Клини

9). $\vdash (A \supset (\neg B \supset C)) \supset (\neg B \supset (A \supset C))$.
10). $\vdash (B \supset C) \supset (A \supset (A \wedge B \supset C))$.
11). $\vdash ((A \supset B) \wedge \neg B) \supset \neg A$.
12). $\vdash \neg A \supset (A \supset B)$.
13). $\vdash ((\neg X \supset Y) \wedge (\neg X \supset \neg Y)) \supset X$.
14). $\vdash (B \supset C) \supset ((A \supset B) \supset (A \supset C))$.
15). $\vdash A \supset A$.

**17.** Доказать справедливость формул:

1. $\forall x\, P(x) \;\wedge\; \forall x\, Q(x) \;=\; \forall x\,(\,P(x) \wedge Q(x)\,)$.
2. $\exists x\, P(x) \;\vee\; \exists x\, Q(x) \;=\; \exists x\,(\,P(x) \vee Q(x)\,)$.
3. $\forall x\, P(x) \;\vee\; \forall x\, Q(x) \;\neq\; \forall x\,(\,P(x) \vee Q(x)\,)$.
4. $\forall x\, P(x) \;\vee\; \forall x\, Q(x) \;\supset\; \forall x\,(\,P(x) \vee Q(x)\,)$.
5. $\exists x\, P(x) \;\wedge\; \exists x\, Q(x) \;\neq\; \exists x\,(\,P(x) \wedge Q(x)\,)$.
6. $\exists x\, P(x) \;\wedge\; \exists x\, Q(x) \;\subset\; \exists x\,(\,P(x) \wedge Q(x)\,)$.
7. $\forall x\, P(x) \;\vee\; C \;=\; \forall x\,(\,P(x) \vee C\,)$, $C$ – const.
8. $\exists x\, P(x) \;\wedge\; C \;=\; \exists x\,(\,P(x) \wedge C\,)$, $C$ – const.

**18.** Привести к предваренной нормальной форме (ПНФ) следующие предикаты:

1). $\exists x\ \exists y\ \ P(x,y) \supset \neg\exists x \forall y\ \ Q(x,y)$.
2). $\neg\forall x\ \exists y\ \ P(x,y) \supset (\neg\exists x \forall y\ \ Q(x,y) \supset \neg\exists x\ \ R(x))$.
3). $\neg\forall x\ \forall y\ \ P(x,y) \vee (\neg\exists x \forall y\ \ P(x,y) \supset \forall x\ \ R(x))$.
4). $\forall x\ \neg\exists y\ \ P(x,y) \supset (\exists x\ \neg\forall y\ \ P(x,y) \supset \forall x\ \neg\exists y \forall z\, S(x,y,z))$.
5). $\forall x\ \ A(x) \supset (\neg\exists x\ \ B(x) \supset (\forall x\ \ C(x) \supset \neg\forall x\ \ D(x)))$.
6). $(\neg\forall x\ \exists y\ \ P(x,y) \supset \neg\exists x \forall y\ \ Q(x,y)) \supset \neg\exists x\ \ R(x)$.
7). $\neg\forall x\ \ R(x) \supset (\neg\exists x\ \neg\forall y\ \ P(x,y)) \supset \exists x\ \exists y \neg\exists z\, S(x,y,z))$.

**19.** Задача (*Кислера*). Браун, Джонс, и Смит обвиняются в подделке сведений о подлежащих налоговому обложению доходах. Они дают под присягой следующие показания:

*Браун* : Джонс виновен, а Смит невиновен.
*Джонс*: Если Браун виновен, то виновен и Смит.
*Смит* : Я невиновен, но хотя бы один из них двоих виновен.

Обозначим через Б, Д и С высказывания: «Браун виновен», «Джонс виновен», «Смит виновен». Выразите показания каждого из обвиняемых логической формулой. Постройте таблицы истинности трех полученных формул. Затем ответьте на следующие вопросы.

1. Совместны ли показания всех троих заподозренных (т. е. могут ли они быть верны одновременно)?
2. Показания одного из обвиняемых следуют из показаний другого; о чьих показаниях идет речь?
3. Если все трое невиновны, то кто совершил лжесвидетельство?
4. Предполагая, что показания всех обвиняемых верны, укажите, кто невиновен, а кто виновен?
5. Если невиновный говорит истину, а виновный лжет, то кто невиновен, а кто виновен?

**20.** Задача (*Клини*). Переведите каждое из следующих рассуждений в логическую символику *высказываний* и проанализируйте полученные высказывания на предмет их тождественной истинности.

1. Любой друг Коли — друг Васи. Петя — не друг Васи, значит, Петя — не друг Коли.
2. Я заплатил бы за ремонт телевизора, только если бы он стал работать. Он же не работает. Поэтому я платить не буду.
3. Если он принадлежит нашей компании, то он храбр и на него можно положиться. Он не принадлежит нашей компании. Значит, он не храбр и на него нельзя положиться.
4. Если подозреваемый совершил кражу, то либо она была тщательно подготовлена, либо он имел соучастника. Если бы кража была подготовлена тщательно, то, если бы был соучастник, украдено было бы гораздо больше. Значит, подозреваемый невиновен.
5. Если он автор этого слуха, то он глуп или беспринципен. Он не глуп и не лишен принципов, значит, не он автор этого слуха.
6. Если наступит мир, то возникнет депрессия, разве что страна проведет программу перевооружения либо осуществит грандиозную программу внутренних капиталовложений в области образования, охраны окружающей среды, борьбы с бедностью и т. д. Невозможно договориться о целях такой грандиозной программы внутренних капиталовложений. Значит, если наступит мир и не будет депрессии, то непременно будет осуществляться программа перевооружения.
7. Намеченная атака удастся, только если захватить противника врасплох или же если позиции его плохо защищены. Захватить его врасплох можно только, если он беспечен. Он не будет беспечен, если его позиции плохо защищены. Значит, атака не удастся.
8. Если мы не будем продолжать политику сохранения цен, то мы потеряем голоса фермеров. Если же мы будем продолжать эту политику, то продолжится перепроизводство, разве что мы прибегнем к контролю над производством. Без голосов фермеров нас не переизберут. Значит, если нас переизберут и мы не прибегнем к контролю над производством, то будет продолжаться перепроизводство.
9. В бюджете возникнет дефицит, если не повысят пошлины. Если в бюджете имеется дефицит, то государственные расходы на общественные нужды сократятся. Значит, если повы-

сят пошлины, то государственные расходы на общественные нужды не сократятся.

### Комбинаторные схемы

**21.** Доказать комбинаторными рассуждениями (т. е. используя только определение числа сочетаний) тождества:

а) $C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$.

б) $C_n^k = C_{n-1}^k + C_{n-2}^{k-1} + \ldots + C_{n-(k+1)}^0$.

в) $C_{2n}^{n-1} = C_{2n-1}^n + C_{2n-1}^n + \ldots + C_{2n-n}^n$.

г) $C_{n+m+1}^m = C_{n+m}^m + C_{n+m-1}^{m-1} + \ldots + C_n^0$.

**22.** Доказать тождества:

а) $\sum\limits_{k=0}^{n} C_n^k = 2^n$.

б) $\sum\limits_{k=0}^{n} kC_n^k = n\,2^{n-1}$.

в) $\sum\limits_{k=0}^{n} k^2 C_n^k = n(n+1)\,2^{n-2}$.

г) $\sum\limits_{k=0}^{n} C_n^k (m-1)^{n-k} = m^n$.

д) $\sum\limits_{k=0}^{n} (-1)^k C_n^k = 0$.

**23.** Доказать тождество $\sum\limits_{i_n=1}^{m} \sum\limits_{i_{n-1}=1}^{i_n} \sum\limits_{i_{n-2}=1}^{i_{n-1}} \cdot \ldots \cdot \sum\limits_{i_1=1}^{i_2} \sum\limits_{i_0=1}^{i_1} 1 = C_{n+m}^{n+1}$.

**24.** Доказать, что $\sum\limits_{k=1}^{n} \dfrac{C_{n-1}^{k-1}}{C_{2n-1}^k} = \dfrac{2}{n+1}$, $n \geqslant 1$.

**25.** Доказать, что следующие числа: $\dfrac{(2n)!}{2^n}$, $\dfrac{(3n)!}{2^n 3^n}$, $\dfrac{(n^2)!}{n^n}$, $\dfrac{(2n)!}{n!\,(n+1)!}$ являются целыми?

**26.** Доказать формулу бинома Ньютона $(a+b)^n = \sum\limits_{k=0}^{n} C_n^k a^k b^{n-k}$.

**27.** Найти число подмножеств множества $M = \{a_1, a_2, \ldots, a_n\}$.

**28.** Доказать, что $\sum\limits_k C_n^{2k} = \sum\limits_k C_n^{2k+1} = 2^{n-1}$.

**29.** Доказать, что $\sum\limits_{r=0}^{k} C_m^r C_n^{k-r} = C_{n+m}^k$ и $\sum\limits_{r=0}^{n} \left(C_n^k\right)^2 = C_{2n}^n$ (теорема сложения).

**30.** Доказать равенство $\sum\limits_{k=0}^{n} \sum\limits_{r=0}^{n-k} C_n^k C_{n-k}^r = 3^n$.

**31.** Доказать равенство $\sum\limits_{k=0}^{n} \frac{1}{k+1} C_n^k = \frac{2^{n+1}-1}{n+1}$.

**32.** Доказать равенство $\sum\limits_{k=0}^{n} (-1)^k \frac{1}{k+1} C_n^k = \frac{1}{n+1}$.

**33.** Найти сумму $\sum\limits_{k=0}^{n} \left(k^2 - 1 + \frac{1}{k+1}\right) C_n^k$.

**34.** Доказать тождество $\sum \frac{n!}{n_1!\, n_2! \cdot \ldots \cdot n_k!} = k^n$, где суммирование распространяется на все упорядоченные разбиения $n$ на $k$ слагаемых: $n = n_1 + n_2 + \ldots + n_k$, $n \geqslant 0$, — целые числа.

**35.** Из города $A$ в город $B$ ведут семь дорог, а из города $B$ в город $C$ — три дороги. Сколько возможных маршрутов ведут из $A$ в $C$ через город $B$?

**36.** Сколькими способами число $11^n$ можно представить в виде трех сомножителей (представления, отличающиеся порядком сомножителей, считаются различными; $11^0$ — сомножитель)?

**37.** Сколькими способами можно указать на шахматной доске $2n \times 2n$ два квадрата — белый и черный?

**38.** Сколькими способами можно указать на шахматной доске $2n \times 2n$ белый и черный квадраты, не лежащие на одной горизонтали и вертикали?

**39.** Какое количество матриц из $n$ строк и $m$ столбцов с элементами из множества $\{0, 1\}$ можно составить?

**40.** Сколькими способами можно составить трехцветный флаг, если имеется материал 5 различных цветов? Та же задача, если одна из полос должна быть красной?

**41.** Надо послать 6 срочных писем. Сколькими способами это можно сделать, если любое письмо можно передать любым из 3 курьеров?

**42.** У одного студента 7 книг, у другого 9 различных книг. Сколькими способами они могут обменять одну книгу одного на одну книгу другого?

**43.** Сколько различных словарей надо издать, чтобы можно было переводить с любого из данных $n$ языков на любой другой язык этого же множества?

**44.** В правление избрано $m$ человек. Из них надо выбрать председателя, заместителя председателя, секретаря и казначея. Сколькими способами можно это сделать?

**45.** У мамы 5 яблок, 7 груш и 3 апельсина. Каждый день в течение 15 дней подряд она выдает сыну по одному фрукту. Сколькими способами это может быть сделано?

**46.** У мамы $m$ яблок и $n$ груш. Каждый день в течение $m+n$ дней подряд она выдает сыну по одному фрукту. Сколькими способами это может быть сделано?

**47.** Найти число векторов $a=(a_1, a_2, \ldots, a_n)$, координаты которых удовлетворяют условию $a_i \in \{0,1\},\ i=1,2,\ldots,n,\ \sum\limits_{i=1}^{n} a_i = r$.

**48.** У англичан принято давать детям несколько имен. Сколькими способами можно назвать ребенка, если ему дают не более трех имен, а общее число имен равно $m$?

**49.** Сколькими способами можно расставить белые фигуры: 2 коня, 2 слона, 2 ладьи, ферзя и короля на первой линии шахматной доски?

**50.** Сколькими способами можно расставить $k$ ладей на шахматной доске размером $n \times m$ так, чтобы они не угрожали друг другу, т. е. так, чтобы никакие две из них не стояли на одной вертикали или горизонтали?

**51.** Сколькими способами можно посадить $n$ мужчин и $n$ женщин за круглый стол так, чтобы никакие два лица одного пола не сидели рядом? Та же задача, но стол может вращаться и способы, переходящие при вращении друг в друга, считаются одинаковыми?

**52.** На школьном вечере присутствуют 12 девушек и 15 юношей. Сколькими способами можно выбрать из них 4 пары?

**53.** Пусть $n$ $(n \geqslant 2)$ человек садятся за круглый вращающийся стол. Два размещения будем считать совпадающими, если каждый человек имеет одних и тех же соседей в обоих случаях. Сколько существует способов сесть за стол?

**54.** Хор состоит из 10 участников. Сколькими способами можно в течение трех дней выбирать по 6 участников, так, чтобы каждый день были различные составы хора?

**55.** Сколькими способами можно распределить $3n$ различных предметов между тремя людьми так, чтобы каждый получил $n$ предметов?

**56.** Имеется $n$ абонентов. Сколькими способами можно одновременно соединить три пары?

**57.** Сколькими способами можно составить три пары из $n$ шахматистов?

**58.** Рассматриваются всевозможные разбиения $2n$ элементов на пары, причем разбиения, отличающиеся друг от друга порядком элементов внутри пар и порядком расположения пар, считаются совпадающими. Определить число таких разбиений.

**59.** Доказать, что нечетное число предметов можно выбрать из $n$ предметов $2^{n-1}$ способами.

**60.** Сколькими способами можно посадить рядом 3 англичан, 3 французов и 3 немцев так, чтобы никакие три соотечественника не сидели рядом?

**61.** В колоде 52 карты. В скольких случаях при выборе из колоды 10 карт среди них окажутся: а) ровно один туз; б) хотя бы один туз; в) не менее двух тузов; г) ровно два туза?

**62.** Сколькими способами можно выбрать 6 карт из колоды, содержащей 52 карты, так, чтобы среди них были карты каждой масти?

**63.** На железнодорожной станции имеется $m$ светофоров. Сколько может быть дано различных сигналов, если каждый светофор имеет три состояния: красный, желтый и зеленый?

**64.** Имеется 17 пар различных предметов. Найти полное число выборок из этих предметов. Каждая пара может участвовать в выборке, предоставляя любой из двух ее элементов, или не участвовать. Выборки считаются различными, если отличаются друг от друга своим составом; порядок предметов в выборке не учитывается.

**65.** Найти число способов раскладки $n$ различных шаров по $m$ различным корзинам?

**66.** Найти число способов раскладки $n$ одинаковых шаров по $m$ различным корзинам?

**67.** Сколькими способами можно разместить $n$ одинаковых шаров по $m$ различным корзинам при следующих условиях:

а) пустых корзин нет;

b) во второй корзине $k$ шаров;

c) в первых $k$ корзинах соответственно $a_1, a_2, \ldots, a_k$ шаров?

**68.** Сколькими способами можно разместить $n_1$ красных, $n_2$ желтых и $n_3$ зеленых шаров по $m$ различным корзинам?

**69.** Сколькими способами 3 человека могут разделить между собой 6 одинаковых яблок, 1 апельсин, 1 сливу, 1 лимон, 1 грушу, 1 айву и 1 финик?

**70.** Поезду, в котором находится $n$ пассажиров, предстоит сделать $m$ остановок. Сколькими способами могут распределиться пассажиры между этими остановками?

**71.** Сколькими способами можно раскрасить квадрат, разделенный на четыре части, пятью цветами:

а) допуская окрашивание разных частей в один цвет;

б) если различные части окрашиваются разными цветами?

**72.** Сколькими способами можно выбрать 5 номеров из 36?

**73.** В скольких случаях при игре в «Спортлото» 5 номеров из 36 будут правильно выбраны: а) ровно 3 номера; б) не менее 3 номеров?

**74.** Сколько существует различных комбинаций из 30 монет достоинством 1,2 и 5 рублей? Построить дерево решений.

**75.** Сколькими способами можно раскрасить квадрат, разделенный на девять частей, четырьмя цветами таким образом, чтобы в первый цвет были окрашены 3 части, во второй — 2, в третий — 3, в четвертый — 1 часть?

**76.** Определить коэффициент $c$ в слагаемом $cx_1^3x_2^4x_3^3$ после разложения выражения $(x_1+x_2+x_3)^{10}$ и приведения подобных?

**77.** Сколько делителей имеет число $q = p_1^{\alpha_1}p_2^{\alpha_2}\cdot\ldots\cdot p_n^{\alpha_n}$, где $p_i$ — простые числа, не равные единице, $\alpha_i$ — некоторые натуральные числа? Чему равна сумма этих делителей?

**78.** Доказать, что в разложении числа $n!$ на простые сомножители простое число $p$ входит с показателем $\left[\frac{n}{p^1}\right] + \left[\frac{n}{p^2}\right] + \left[\frac{n}{p^3}\right] + \ldots$.

**79.** Из выбранных $k$ различных чисел от 1 до $n$ составляют произведение, $k$ фиксировано. Какое количество полученных таким образом произведений делится на простое число $p \leqslant n$?

**80.** Сколько можно составить перестановок из $n$ элементов, в которых данные $m$ элементов не стоят рядом в любом порядке?

**81.** На шахматную доску $n \times n$ произвольным образом поставили две ладьи — черную и белую. Что вероятнее: ладьи бьют друг друга или нет?

**82.** Доказать, что из пяти грибов, растущих в лесу и не расположенных на одной прямой, всегда можно найти четыре таких, которые служат вершинами выпуклого четырехугольника?

**83.** В первенстве мира по футболу участвуют 20 команд. Какое наименьшее число игр должно быть сыграно, чтобы среди любых трех команд нашлись две, уже игравшие между собой?

**84.** Некая комиссия собиралась 40 раз. Каждый раз на заседаниях присутствовали по 10 человек, причем никакие двое из ее членов не были на заседаниях вместе больше одного раза. Доказать, что число членов комиссии больше 60.

**85.** В учреждении 25 сотрудников. Доказать, что из них нельзя составить больше 30 комиссий по 5 человек в каждой так, чтобы никакие две комиссии не имели более одного общего сотрудника.

**86.** В соревнованиях по гимнастике две команды имели одинаковое число участников. В итоге, общая сумма баллов, полученных всеми участниками, равна 156. Сколько было участников, если каждый из них получил оценки только 8 или 9 баллов?

**87.** Группа из 41 студентов успешно сдала сессию из трех экзаменов. Возможные оценки: 5,4,3. Доказать, что, по крайней мере, пять студентов сдали сессию с одинаковыми оценками.

**88.** Поступающий в высшее учебное заведение должен сдать четыре экзамена. Он полагает, что для поступления будет достаточно набрать 17 баллов. Сколькими способами он сможет сдать экзамены, набрав не менее 17 баллов и не получив ни одной двойки? Построить дерево решений.

**89.** Каких чисел больше среди первого миллиона: тех, в записи которых встречается 1, или тех, в записи которых ее нет?

**90.** Пусть числа $1, 2, \ldots, n$ расположены подряд по кругу. Двигаясь по кругу, вычеркиваем каждое второе число. Показать, что последнее не вычеркнутое число равно $2n - 2^{[\log_2(n)]+1} + 1$.

**91.** Сколькими способами можно число $n$ представить в виде суммы $k$ слагаемых (представления, отличающиеся лишь порядком слагаемых, считаются различными), если: а) каждое слагаемое является целым неотрицательным числом; б) каждое слагаемое — натуральное число?

**92.** Какова таблица инверсий для перестановки 271845936?

**93.** Какой перестановке отвечает таблица инверсий 50121200?

**94.** Пусть перестановке $a_1a_2\dots a_n$ соответствует таблица инверсий $d_1d_2\dots d_n$ . Какой перестановке тогда будет соответствовать следующая таблица инверсий: $(n-1-d_1)(n-2-d_2)\dots$ $\dots(n-n-d_n)$?

**95.** На окружности произвольным образом отмечают $n$ точек буквой $N$ и $m$ точек буквой $M$. На каждой из дуг, на которые окружность делится выбранными точками, ставят числа 2 или $1/2$ следующим образом: если концы дуги отмечены буквой $N$, то ставят число 2; если концы дуги отмечены буквой $M$, то ставят число $1/2$; если же концы дуги отмечены различными буквами, то ставят число 1. Доказать, что произведение всех поставленных чисел равно $2^{n-m}$.

**96.** Сколькими способами можно распределить $3n$ различных книг между тремя лицами так, чтобы числа книг образовывали арифметическую прогрессию?

**97.** Рассматриваются всевозможные разбиения $nk$ элементов на $n$ групп по $k$ элементов в каждой, причем разбиения, отличающиеся друг от друга только порядком элементов внутри групп и порядком расположения групп, считаются совпадающими. Сколько существует различных таких разбиений?

**98.** Сколькими способами можно разбить 30 рабочих на 3 бригады по 10 человек в каждой бригаде? На 10 групп по 3 человека в каждой группе?

**99.** Сколькими способами можно разделить колоду из 36 карт пополам так, чтобы в каждой пачке было по два туза?

**100.** Сколькими способами можно разложить 10 книг в 5 бандеролей по 2 книги в каждую (порядок бандеролей не принимается во внимание)?

**101.** Сколькими способами можно разложить 9 книг в 4 бандероли по 2 книги и в 1 бандероль 1 книгу (порядок бандеролей не принимается во внимание)?

**102.** Сколькими способами можно разделить 9 книг в 3 бандероли по 3 книги в каждую (порядок бандеролей не принимается во внимание)?

**103.** На первые две линии шахматной доски выставляют белые и черные фигуры (по два коня, два слона, две ладьи, ферзя и короля каждого цвета). Сколькими способами можно это сделать?

**104.** Сколькими способами можно расположить в 9 лузах 7 белых шаров и 2 черных шара? Часть луз может быть пустой, и лузы считаются различными.

**105.** В лифт сели 8 человек. Сколькими способами они могут выйти на четырех этажах так, чтобы на каждом этаже вышел, по крайней мере, один человек?

**106.** Доказать, что число упорядоченных разбиений числа $n$ на $k$ натуральных слагаемых, т. е. число решений уравнения $n = x_1 + x_2 + \ldots + x_k$, $x_i > 0$, $i = 1, 2, \ldots, k$, равно $C_{n-1}^{k-1}$, а общее число упорядоченных разбиений для различных $k$ равно $2^{n-1}$.

**107.** Сколькими способами можно разложить $n$ различных шаров по $k$ различным корзинам так, чтобы в первую корзину попало $n_1$ шаров, во вторую корзину попало $n_2$ шаров и т. д., в $k$-ю корзину попало $n_k$ шаров, где $n = n_1 + n_2 + \ldots + n_k$?

**108.** Сколько существует чисел от 0 до $10^n$, которые не содержат две идущие друг за другом одинаковые цифры?

**109.** Сколько существует натуральных $n$-значных чисел, у которых цифры расположены в неубывающем порядке?

**110.** Сколько существует натуральных чисел, не превышающих $10^n$, у которых цифры расположены в неубывающем порядке?

**111.** Сколькими способами можно расставить $n$ нулей и $k$ единиц так, чтобы никакие две единицы не стояли рядом?

**112.** Город имеет вид прямоугольника, разделенного улицами на квадраты. Таких улиц в направлении с севера на юг равно $n$, а в направлении с востока на запад — $k$. Сколько имеется кратчайших дорог от одной из вершин прямоугольника до противоположной?

**113.** Как разбить квадратное поле на участки так, чтобы высеять на нем $m$ сортов пшеницы для сравнения урожайности этих сортов, исключающего влияние изменения плодородия в пределах участка? Считаем, что плодородие убывает при удалении от одной стороны поля (неизвестно, какой именно) к противоположной.

**114.** Бросают $m$ игральных костей, помеченных числами 1,2,3,4,5,6. Сколько может получиться различных результатов (результаты, отличающиеся порядком очков, считаются одинаковыми)?

**115.** Имеем $m$ различных шаров и $k$ различных корзин. Сколькими способами можно разместить предметы по корзинам, допускаются пустые корзины?

**116.** Имеем $m$ различных шаров и $k$ различных корзин. Сколькими способами можно разместить предметы по корзинам,

пустые корзины не допускаются? *Указание*: воспользоваться правилом включения и исключения.

**117.** Найти число способов разложения $m$ шаров по $k$ корзинам так, чтобы $r$ корзин остались пустыми. *Указание*: воспользоваться правилом включения и исключения.

**118.** Имеем $m$ различных шаров $a_1, a_2, \ldots, a_n$ и столько же различных корзин $k_1, k_2, \ldots, k_n$. Сколькими способами можно разместить предметы по корзинам так, чтобы никакой предмет $a_i$ не попал в корзину $k_i$ (допускаются пустые корзины)? *Указание*: воспользоваться правилом включения и исключения.

**119.** *Задача о беспорядках*. Имеем $m$ различных шаров $a_1, a_2, \ldots, a_n$ и столько же различных корзин $k_1, k_2, \ldots, k_n$. Сколькими способами можно разместить предметы по корзинам так, чтобы никакой предмет $a_i$ не попал в корзину $k_i$, пустые корзины не допускаются? *Указание*: воспользоваться правилом включения и исключения.

**120.** Найти число перестановок $m$ шаров, в которых ровно $r$ элементов остаются на месте. *Указание*: воспользоваться правилом включения и исключения.

**121.** Доказать, что $r$ различных вещей можно разделить между $n+p$ людьми так, чтобы данные $n$ людей получили, по крайней мере, по одному предмету, способами $C_n^0(n+p)^r - C_n^1(n+p-1)^r + C_n^2(n+p-2)^r - \ldots + (-1)^n C_n^n(n+p-n)^r$. *Указание*: воспользоваться правилом включения и исключения.

**122.** *Рыцарские переговоры*. К обеду за круглым столом приглашены $n$ пар враждующих рыцарей ($n \geqslant 2$). Требуется рассадить их так, чтобы никакие два врага не сидели рядом. Показать, что это можно сделать $\sum\limits_{k=0}^{n}(-1)^k C_n^k 2^k (2n-k)!$ способами. *Указание*: воспользоваться правилом включения и исключения.

**123.** Найти число целых положительных чисел, не превосходящих 1000 и не делящихся ни на одно из чисел 3, 5 и 7.

**124.** Найти число целых положительных чисел, не превосходящих 1000 и не делящихся ни на одно из чисел 6, 10 и 15.

**125.** Показать, что если $n = 30m$, то число целых, не превосходящих $n$ и не делящихся ни на одно из чисел $6, 10, 15$, равно $22m$.

**126.** При обследовании читательских вкусов студентов оказалось, что 60% студентов читает журнал A, 50% — журнал B, 50% — журнал C, 30% — журналы A и B, 20% — журналы B

и C, 40% — журнал A и C, 10% — журнал A, B и C. Сколько процентов студентов а) не читает ни одного из журналов; б) читает в точности два журнала; в) читает не менее двух журналов?

**127.** На одной из кафедр университета работают тринадцать человек, причем каждый из них знает хотя бы один иностранный язык. Десять человек знают английский, семеро — немецкий, шестеро — французский. Пятеро знают английский и немецкий, четверо — английский и французский, трое — немецкий и французский. Найти: а) сколько человек знают все три языка; б) сколько знают ровно два языка; в) сколько знают только английский?

**128.** Имеются $3n+1$ предметов, из которых $n$ одинаковых, а остальные различны. Доказать, что из них можно извлечь $n$ предметов $2^n$ способами.

**129.** Применяя формулу включения и исключения, определить количество целочисленных решений системы уравнений и неравенств: $x_1 + x_2 + \ldots + x_n = r, a_i \leqslant x_i \leqslant b_i,\ i = \overline{1,n}$, $a_i, x_i, b_i, r$ — целые числа.

**130.** Определить количество целочисленных решений системы $x_1 + x_2 + x_3 = 40, x_1 \geqslant 3, x_2 \geqslant 0, x_3 \geqslant 2$.

**131.** Компания, состоящая из 10 супружеских пар, разбивается на 5 групп по 4 человека для лодочной прогулки. Сколькими способами можно разбить их так, чтобы в каждой лодке оказались двое мужчин и две женщины?

**132.** Имеем $n$ предметов расположенных в ряд. Сколькими способами можно выбрать из них три предмета так, чтобы не брать никаких двух соседних предметов?

**133.** Даны $2n$ различных предметов $a_1, a_1, a_2, a_2, \ldots, a_n, a_n$. Сколько существует перестановок из этих $2n$ предметов, в которых не стоят рядом одинаковые элементы?

**134.** В шахматной олимпиаде участвуют представители $n$ стран по 4 представителя от каждой страны. Сколькими способами они могут встать в ряд так, чтобы рядом с каждым был представитель той же страны?

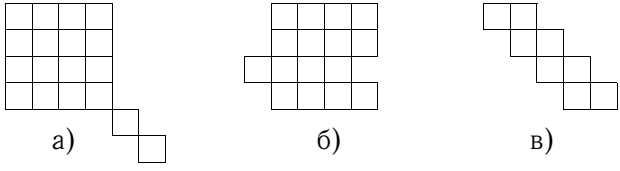
**135.** Имеется $n$ одинаковых вещей и еще $n$ различных вещей. Сколькими способами можно выбрать из них $n$ вещей? Сколькими способами можно упорядочить все $2n$ вещей?

**136.** Найти число способов распределения $2n$ одинаковых шаров по двум неразличимым корзинам.

**137.** В каждой клетке шахматной доски размером $n \times n$ поставили число, указывающее количество прямоугольников, в ко-

торые входит эта клетка. Чему равна сумма всех поставленных чисел?

**138.** Найти число расстановок $k$ ладей так, чтобы они не били друг друга, на доске $n \times n$ (см. случаи *а*,*б*,*в*) с выколотыми или добавленными клетками. В случае *в*) использовать $n$ ладей.




### Производящие функции и рекуррентные соотношения

**139.** Найти производящую функцию последовательности $\{2(n-5)+7^{n+2}\}$.

**140.** Применить технику производящих функций для нахождения суммы чисел $1^3+2^3+\ldots+n^3$.

**141.** Решить рекуррентные соотношения:

1) $u_{n+2}-4u_{n+1}+3u_n=0,\ u_0=8, u_1=10.$
2) $u_{n+3}-3u_{n+2}+u_{n+1}-3u_n=0,\ u_0=1, u_1=3, u_2=8.$
3) $u_{n+2}\pm 9u_n=0,\ u_0=1, u_1=0.$
4) $u_{n+4}\pm 4u_n=0,\ u_0=1, u_1=1, u_2=1, u_3=1.$
5) $u_{n+3}+u_{n+2}-u_{n+1}-u_n=0,\ u_0=1, u_1=2, u_2=3.$
6) $u_{n+2}-4u_{n+1}+4u_n=0,\ u_0=1, u_1=2.$

**142.** Решить неоднородные рекуррентные соотношения:

1) $u_{n+1}=u_n+n,\ u_0=1.$
2) $u_{n+2}=-2u_{n+1}+8u_n+27\cdot 5n,\ u_0=0, u_1=-9.$
3) $u_{n+2}-3u_{n+1}+2u_n=n,\ u_0=1, u_1=1.$
4) $u_{n+2}-4u_{n+1}+4u_n=2^n,\ u_0=1, u_1=2.$
5) $u_{n+2}=u_{n+1}-\dfrac{1}{4}u_n+2^{-n},\ u_0=1, u_1=3/2.$
6) $u_{n+2}-3u_{n+1}+2u_n=(-1)^n,\ u_0=1, u_1=2.$

**143.** Последовательность Фибоначчи $\{u_n\}$ задается рекуррентным соотношением $u_{n+2}=u_{n+1}+u_n,\ u_0=1, u_1=1$. Найти $u_n$; показать, что $u_n$ и $u_{n+1}$ — взаимно простые числа и $u_n$ делится на $u_m$, где $n=m\cdot k$.

**144.** Найти общее решение рекуррентных соотношений:

1) $u_{n+2}=u_{n+1}-\dfrac{1}{4}u_n.$

2) $u_{n+2} - 4u_{n+1} + 3u_n = 0$.
3) $u_{n+2} - u_{n+1} - 2u_n = 0$.

**145.** Найти решение системы рекуррентных соотношений: $\begin{cases} a_{n+1} = 3a_n + b_n, \\ b_{n+1} = -a_n + b_n, \end{cases}$ где $a_0 = 14$, $b_0 = -6$.

**146.** Найти число решений уравнения $x + 2y = n$, где $x, y, n \in Z^+$ — положительные целые числа; $x, y$ — неизвестные.

**147.** Найти число решений уравнения $x + 2y + 4z = n$, где $x, y, n \in Z^+$, $x, y, z$ — неизвестные.

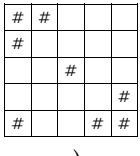
**148.** Найти определитель матрицы

$$\begin{pmatrix} \alpha+\beta & \alpha\beta & & & & \\ 1 & \alpha+\beta & \alpha\beta & & 0 & \\ & 1 & \alpha+\beta & \alpha\beta & & \\ & & \cdots & & & \\ & & & \cdots & & \\ & 0 & & 1 & \alpha+\beta & \alpha\beta \\ & & & & 1 & \alpha+\beta \end{pmatrix},$$

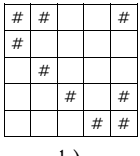
где $\alpha, \beta$ — произвольные числа; вне обозначенных диагоналей матрицы располагаются нули.

**149.** Вычислить сумму $\sum \frac{1}{2^k}$, где суммирование производится по всем натуральным $k$, не кратным 2, 3 и 5.

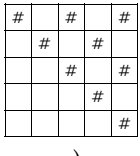
**150.** Найти ладейный многочлен запрещенных позиций для досок a) и b) и для каждого из этих случаев составить многочлен попаданий. Для доски c) составить многочлен запрещенных позиций. Для досок a), b) и c) найти число расстановок трех ладей на запрещенных позициях.



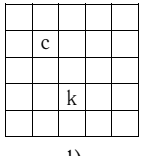
a)



b)



c)

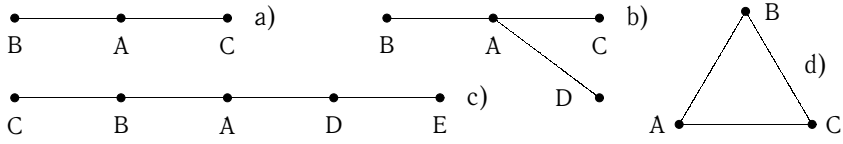



d)

**151.** Найти число способов расставить 5 ладей на доске d) так, чтобы ни одну из них не бил слон. Позиция слона указана на доске символом «c».

**152.** Найти число способов расставить 5 ладей на доске d) так, чтобы ни одну из них не бил конь. Позиция коня указана на доске символом «k».

**153.** Найти число замкнутых маршрутов длины $2n$ по ребрам графа для случаев a), b), c) и d). Длина ребра равна 1. Начало и конец пути есть вершина $A$.




**154.** Найти число замкнутых маршрутов длины $n$ по a) сторонам треугольника и б) ребрам тетраэдра. Длины сторон треугольника и ребра тетраэдра равны 1. Начало и конец пути есть любая из вершин рассматриваемой геометрической фигуры.

**155.** На листе бумаги нанесена ортогональная сетка. Сколько различных $n$-звенных ломаных можно провести по линиям сетки, если ломаная может проходить один и тот же отрезок несколько раз? *Указание*: построить дерево решений.

**156.** На листе бумаги нанесена ортогональная сетка. Сколько различных $2n$-звенных замкнутых ломаных можно провести по линиям сетки, если ломаная может проходить один и тот же отрезок несколько раз?

**157.** Маршрут выполняется по оси OX. Исходное расположение — начало координат. Начальное направление движения выбирается в любую из сторон оси. В каждой целочисленной координате оси OX направление движения можно менять. Определить количество таких замкнутых маршрутов длины $2n$.

## Теория графов и приложения

**158.** Покажите, что $[n^2/4]$ — максимальное число ребер, которое может иметь $n$-вершинный граф без треугольников.

**159.** Доказать по индукции, что для плоского графа $\Gamma$ справедливо неравенство $\chi(\Gamma) \leqslant 5$.

**160.** Показать, что в любом графе выполняется условие $2|U| = \sum\limits_{x \in X} d(x)$, где $|U|$ — число ребер, $d(x)$ — степени вершин.

**161.** Показать, что в любом графе число вершин с нечетными степенями четно.

**162.** Показать, что в любом простом графе найдутся две вершины с одинаковыми степенями.

**163.** Может ли полный граф иметь 7, 8, 9 или 10 ребер?

**164.** Показать, что связный граф с $n$ вершинами содержит не менее чем $n-1$ ребро.

**165.** Пусть $d$ — минимальная из степеней графа с $n$ вершинами. Показать, что если $d \geqslant \dfrac{n-1}{2}$, то граф связный.

**166.** Показать, что в любом графе с $n$ вершинами и ребрами не менее $C_{n-1}^2 + 2$ существует гамильтонов цикл.

**167.** Доказать, что всякий замкнутый маршрут нечетной длины содержит простой цикл. Верно ли аналогичное утверждение для маршрутов четной длины?

**168.** Показать, что для графа $\Gamma$ с $n$ вершинами выполняется $\beta(\Gamma)\chi(\Gamma) \geqslant n$, где $\beta(\Gamma)$ — максимальное число независимых вершин в графе; $\chi(\Gamma)$ — хроматическое число графа.

**169.** Дана симметричная матрица размером $n \times n$, в каждой строке которой располагается нечетное число ненулевых элементов. Показать, что $n$ является четным. Диагональные элементы матрицы равны нулю.

**170.** Показать, что если в графе с $n$ вершинами отсутствуют циклы нечетной длины и число ребер превышает $((n-1)/2)^2$, то граф связный.

**171.** Пусть $l$ — длина самой длинной простой цепи в графе $\Gamma$. Показать, что $\chi(\Gamma) \geqslant l+1$.

**172.** Пусть $d$ — максимальная степень вершины в графе $\Gamma$. Показать, что $\chi(\Gamma) \geqslant d+1$.

**173.** Сколько существует простых графов с $2n$ вершинами и $n$ ребрами, компоненты связности которых являются изолированными ребрами?

**174.** Показать, что число различных совершенных паросочетаний (максимальных) в полном графе с $2n$ вершинами равно $\dfrac{(2n)!}{(2!)^n n!}$.

**175.** Показать, что в полном графе с $n$ вершинами имеется $(n-1)!/2$ различных гамильтоновых циклов.

**176.** Показать, что полный граф с $2n+1$ вершинами можно представить в виде объединения $n$ гамильтоновых циклов.

**177.** Показать, что граф с $n$ вершинами и числом ребер больше $(n-1)(n-2)/2$ является связным.

**178.** Имеется $n$ лиц, каждые двое имеют точно одного знакомого. Показать, что в графе знакомств найдется хотя бы один подграф вида треугольника и не найдется ни одного подграфа вида прямоугольника.

**179.** Определить хроматическое число полного $n$-вершинного графа.

**180.** Определить хроматическое число связного однородного графа степени 2 с $n$ вершинами (однородный граф обозначает, что степени всех вершины одинаковые).

**181.** Определить хроматическое число графа, полученного из полного $n$-вершинного графа удалением: а) одного ребра; б) двух ребер; в) трех ребер, составляющих треугольник.

**182.** Найти хроматическое число $n$-вершинного линейного дерева, $n > 1$.

**183.** Найти хроматическое число $n$-вершинного дерева, $n > 1$.

**184.** Доказать, что в компании из 6 человек всегда найдутся либо трое знакомых друг с другом, либо трое друг с другом незнакомых.

**185.** Доказать, что при любой раскраске ребер полного 6–графа в два цвета найдется монохроматический полный 3–граф.

**186.** Рассмотрим граф $M_n$ Муна–Мозера с $3n$ вершинами $\{1, 2, \ldots, 3n\}$, в котором вершины разбиты на триады $\{1, 2, 3\}, \{4, 5, 6\}, \ldots, \{3n-2, 3n-1, 3n\}$; $M_n$ не имеет ребер внутри любой триады, но вне их каждая вершина связана с каждой из остальных. Докажите (по индукции), что $M_n$ имеет $3^n$ клик.

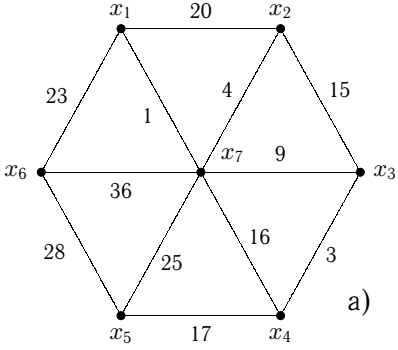
**187.** Показать, что в графе с $n$ вершинами и $k$ компонентами связности число ребер не более $(n-k)(n-k+1)/2$.

**188.** Докажите, что если в графе все вершины имеют четные степени, то в этом графе нет мостов. *Указание*: допустить, что существует мост; удалить это ребро-мост и показать, что полученный граф будет противоречить условию задачи.

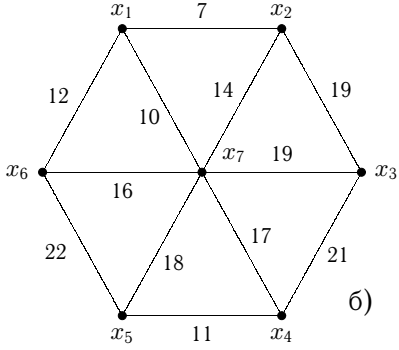
**189.** Разобьем плоскость на конечное число частей прямыми линиями. Докажите, что полученная карта имеет хроматическое число, равное двум. *Указание*: для доказательства воспользуйтесь индукцией по числу прямых.

**190.** Для каждого графа а) и б) найти остовное дерево, используя программную реализацию жадного алгоритма (алгоритм

6.7) и алгоритма ближайшего соседа (алгоритм 6.9). На каждом шаге алгоритмов отобразить состояние используемых структур данных.

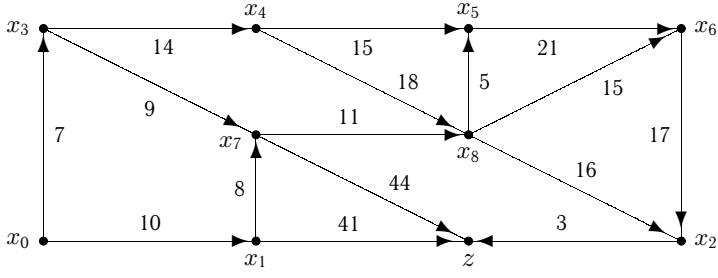



а)



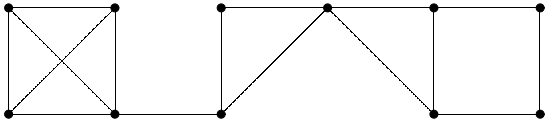

б)

**191.** Найти кратчайшие пути от вершины $x_0$ до всех остальных вершин ориентированного графа.




**192.** Пусть $A^k = [a_{ij}^{(k)}]$, $i, j = \overline{1, n}$, обозначает $k$-ю степень матрицы смежности орграфа. Доказать, что элемент $a_{ij}^{(k)}$ данной матрицы равен количеству маршрутов длины $k$ из вершины $x_i$ в $x_j$. *Указание*: при доказательстве воспользоваться индукцией по числу $k$.

**193.** Выполнить хроматическое разложение графа. Найти все клики, фундаментальное множество циклов, листы, блоки и мосты. Определить центры, радиус и диаметр графа.



**194.** Сколь много ребер может иметь $n$-вершинный граф, у которого степень всякой вершины не превосходит $d$?

**195.** Показать, что в дереве с нечетным диаметром любые две простые цепи наибольшей длины имеют хотя бы одно общее ребро.

**196.** Пусть корневое дерево с $n \geqslant 2$ висячими вершинами не имеет вершин степени 2, отличных от корня. Показать, что общее число вершин дерева не превосходит $2n - 1$.

**197.** Доказать, что дерево обладает единственным центром в случае, когда его диаметр есть число четное, и обладает двумя центрами, когда диаметр есть нечетное число.

**198.** Доказать, что в любом дереве с $n \geqslant 2$ вершинами имеется не менее двух висячих вершин.

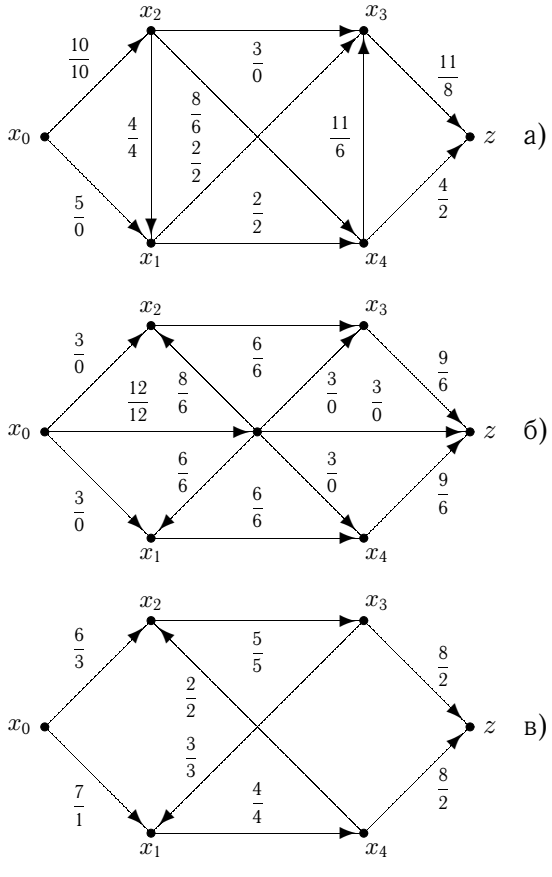
**199.** Вершины графа $\Gamma$ пронумерованы в порядке возрастания их степеней. Доказать, что если $k$ — наибольшее число, такое, что $k \leqslant d(x_k) + 1$, то $\chi(\Gamma) \leqslant k$, где $d(x_k)$ — степень вершины $x_k$.

**200.** Доказать, что если для любых двух вершин $x$ и $y$ связного $n$-вершинного графа выполняется $d(x) + d(y) \geqslant n$, то граф имеет гамильтонов цикл.

**201.** Используя алгоритм чередующихся цепей (см. п. 7.14.3), расширить заданное начальное паросочетание в двудольном графе $\Gamma = (V_1 \cup V_2, U, \Phi)$ до максимального паросочетания, где $V_1 = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ и $V_2 = \{1, 2, 3, 4, 5, 6\}$, смежные вершины в графе: $s_1$ — $\{1, 2, 3\}$, $s_2$ — $\{1, 2\}$, $s_3$ — $\{1, 2\}$, $s_4$ — $\{1, 2, 3, 4\}$, $s_5$ — $\{1, 2, 4, 5\}$, $s_6$ — $\{1, 2, 3, 5, 6\}$, и начальное паросочетание $\pi = \{(s_1, 1), (s_2, 2), (s_4, 3), (s_5, 4), (s_6, 5)\}$.

**202.** Используя алгоритм чередующихся цепей (см. п. 7.14.3), расширить заданное начальное паросочетание в двудольном графе $\Gamma = (V_1 \cup V_2, U, \Phi)$ до максимального паросочетания, где $V_1 = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ и $V_2 = \{1, 2, 3, 4, 5, 6, 7\}$; смежные вершины в графе: $s_1$ — $\{1, 2\}$, $s_2$ — $\{2, 5, 6\}$, $s_3$ — $\{5\}$, $s_4$ — $\{1, 2, 5\}$, $s_5$ — $\{3, 4, 5, 6\}$, $s_6$ — $\{4, 5, 7\}$, $s_7$ — $\{5, 6\}$, и начальное паросочетание $\pi = \{(s_1, 1), (s_2, 2), (s_3, 5), (s_5, 3), (s_6, 7), (s_7, 6)\}$.

**203.** Перераспределить заданный начальный поток по транспортной сети а), б) и в) так, чтобы он стал максимальным, а сеть — насыщенной (см. п. 7.10). Найти все разрезы транспортной сети и их мощности, сравнить мощность минимального разреза с найденным максимальным потоком.




**204.** Решить задачи о назначениях максимального выбора

а)

| | | | | | |
|---|---|---|---|---|---|
| 2 | 6 | 4 | 7 | 1 | 2 |
| 8 | 7 | 2 | 4 | 3 | 5 |
| 1 | 5 | 3 | 4 | 3 | 3 |
| 8 | 7 | 1 | 2 | 0 | 1 |
| 9 | 11 | 1 | 3 | 3 | 6 |
| 12 | 9 | 9 | 4 | 5 | 7 |

б)

| | | | | |
|---|---|---|---|---|
| 10 | 15 | 11 | 2 | 9 |
| 7 | 1 | 14 | 8 | 6 |
| 17 | 18 | 16 | 16 | 15 |
| 2 | 11 | 8 | 3 | 5 |
| 9 | 14 | 6 | 1 | 10 |

**205.** Решить задачи о назначениях минимального выбора

а)

| | | | |
|---|---|---|---|
| 4 | 6 | 9 | 7 |
| 13 | 10 | 14 | 14 |
| 9 | 9 | 16 | 13 |
| 12 | 10 | 12 | 10 |

б)

| | | | | |
|---|---|---|---|---|
| 9 | 20 | 60 | 15 | 21 |
| 38 | 71 | 69 | 49 | 60 |
| 28 | 13 | 80 | 28 | 34 |
| 58 | 34 | 13 | 37 | 25 |
| 30 | 3 | 53 | 20 | 21 |

в)

| | | | | | |
|---|---|---|---|---|---|
| 44 | 74 | 35 | 49 | 30 | 45 |
| 22 | 28 | 42 | 59 | 83 | 41 |
| 28 | 39 | 54 | 47 | 35 | 24 |
| 49 | 53 | 45 | 50 | 43 | 38 |
| 27 | 37 | 30 | 18 | 30 | 22 |
| 70 | 27 | 21 | 32 | 31 | 9 |

**206.** Фирма по перевозке грузов должна забрать пять контейнеров в пунктах района края $A, B, C, D, E$ и доставить их в пункты $a, b, c, d, e$. Расстояния в километрах между пунктами загрузки и пунктами назначения контейнеров приведены ниже:

| $A-a$ | $B-b$ | $C-c$ | $D-d$ | $E-e$ |
|---|---|---|---|---|
| 60 | 30 | 100 | 50 | 40 |

Фирма располагает пятью грузовиками двух типов $X$ и $Y$ на стоянках $S, T, U, V, W$. Три грузовика типа $X$ находятся на автостоянках $S, U, V$ и два грузовика типа $Y$ находятся на автостоянках $T, W$. Стоимость пробега одного километра для грузовиков типа $X$ и $Y$, включая горючее, страховку и т. д., приведена ниже:

| | Пустой | Груженный |
|---|---|---|
| $X$ | 20 | 40 |
| $Y$ | 30 | 60 |

Расстояния от стоянки грузовиков до места назначения приведены ниже:

| | Расстояние | | | | |
|---|---|---|---|---|---|
| Стоянки | $A$ | $B$ | $C$ | $D$ | $E$ |
| $S$ | 30 | 20 | 40 | 10 | 20 |
| $T$ | 30 | 10 | 30 | 20 | 30 |
| $U$ | 40 | 10 | 10 | 40 | 10 |
| $V$ | 20 | 20 | 40 | 20 | 30 |
| $W$ | 30 | 20 | 10 | 30 | 40 |

Определить распределение контейнеров по грузовикам, минимизирующее общую стоимость перевозок.

**207.** Ежедневно авиалиния, которая принадлежит некоторой компании, осуществляет следующие перелеты между городами $A$ и $B$:

| № полета | Отправление из города A | Прибытие в город B | № полета | Отправление из города B | Прибытие в город A |
|---|---|---|---|---|---|
| 1 | 9.00 | 11.00 | 11 | 8.00 | 10.00 |
| 2 | 10.00 | 12.00 | 12 | 9.00 | 11.00 |
| 3 | 15.00 | 17.00 | 13 | 14.00 | 16.00 |
| 4 | 19.00 | 21.00 | 14 | 20.00 | 22.00 |
| 5 | 20.00 | 22.00 | 15 | 21.00 | 23.00 |

Компания хочет организовать полеты туда и обратно так, чтобы минимизировать время простоя при условии, что каждому самолету требуется 1 час для дозаправки.

**208.** Авиалиния связывает три города A, B, C. Полеты происходят семь дней в неделю согласно расписанию:

| Вылет | | Прибытие | |
|---|---|---|---|
| Город | Время | Город | Время |
| A | 8.00 | B | 12.00 |
| A | 9.00 | C | 12.00 |
| A | 10.00 | B | 14.00 |
| A | 14.00 | B | 18.00 |
| A | 18.00 | B | 22.00 |
| A | 20.00 | C | 23.00 |
| B | 7.00 | A | 11.00 |
| B | 9.00 | A | 13.00 |
| B | 13.00 | A | 17.00 |
| B | 18.00 | A | 22.00 |
| C | 9.00 | A | 12.00 |
| C | 15.00 | A | 18.00 |

Как следует распределить самолеты по линиям для минимизации стоимости простоя в каждом из городов? Следует учесть, что самолет не может подняться менее чем через 1 час после приземления, так как требуется время на технический контроль и заправку.

## Теория групп и приложения

**209.** Показать, что $G_1 = \{n \,|\, n \in Z\}$ — группа с операцией сложения чисел, где $Z$ — множество целых чисел; $G_2 = \{2n \,|\, n \in Z\}$ — группа с операцией сложения чисел;

$G_3 = \{2^n \mid n \in Z\}$ — группа с операцией умножения чисел; $G_4$ — множество квадратных матриц порядка $n$, определитель которых не равен нулю, является группой с операцией умножения матриц; $G_5$ — множество ортогональных матриц порядка $n$ является группой с операцией умножения матриц; $G_6 = \{1, -1\}$ — группа с операцией умножения чисел; $G_7$ — множество рациональных чисел является группой относительно операций сложения и умножения (без нуля); $G_8$ — множество вещественных чисел является группой относительно операцией сложения и умножения (без нуля). Установить, какие из групп являются подгруппами других групп.

**210.** Пусть $M$ подмножество группы $G$ и $\forall a, b \in M$ выполняется $ab^{-1} \in M$ . Показать, что $M$ — подгруппа.

**211.** Пусть $G_1, G_2$ — группы и отображение $f : G_1 \to G_2$ — гомоморфизм. Показать, что ядро и образ гомоморфизма являются подгруппами.

**212.** Пусть $G_1 = \{x \in R^+\}$ и $G_2 = \{x \in R\}$, где $R^+$ — положительные вещественные числа, $R$ — вещественные числа. Показать, что $G_1$ — группа с операцией умножения, $G_2$ — группа с операцией сложения. Рассмотрим отображение $\varphi : G_1 \to G_2$ такое, что $\forall x \in G_1\, \varphi(x) = \ln(x) \in G_2$. Показать, что $\varphi$ — изоморфизм групп.

**213.** Пусть $G$ — группа и $r \in G$ — фиксированный элемент. Определим отображение $\varphi : G \to G$ формулой $\forall g \in G\, \varphi(g) = = r^{-1}gr$. Докажите, что $\varphi$ — изоморфизм группы на себя.

**214.** Докажите, что если порядок группы $G$ равен $2n$ и $H$ — подгруппа порядка $n$ группы $G$, то $H$ — ее нормальный делитель.

**215.** Доказать, что если в квадратной матрице порядка $n$ содержится нулевая подматрица размера $r \times s$ и $r + s > n$ , то определитель матрицы равен нулю.

**216.** Найти порядок группы, порожденной подстановкой $\left(\begin{smallmatrix}1&2&3&4&5\\2&3&4&5&1\end{smallmatrix}\right)$.

**217.** Разложить подстановки $\left(\begin{smallmatrix}1&2&3&4&5&6&7&8\\3&2&4&7&5&6&1&8\end{smallmatrix}\right)$ и $\left(\begin{smallmatrix}1&2&3&4&5&6&7\\7&6&2&1&4&3&5\end{smallmatrix}\right)$ в произведение независимых циклов и в произведение транспозиций, определить их четность.

**218.** Пусть $H \subset S_4$, где $S_4$ — симметрическая группа. Будет ли $H$ подгруппой в следующих случаях:

а) $H = \left\{\left(\begin{smallmatrix}1&2&3&4\\1&2&3&4\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\2&1&4&3\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\3&4&1&2\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\4&3&2&1\end{smallmatrix}\right)\right\}$.

б) $H = \left\{\left(\begin{smallmatrix}1&2&3&4\\1&2&3&4\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\2&3&1&4\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\3&1&2&4\end{smallmatrix}\right), \left(\begin{smallmatrix}1&2&3&4\\2&4&1&3\end{smallmatrix}\right)\right\}$.

**219.** Пусть $G$ — циклическая группа, $|G| = m$. Показать, что число образующих группы равно $\varphi(m)$ — функция Эйлера.

**220.** Пусть $G$ — циклическая группа. Показать, что элементы группы одинакового порядка являются образующими одной и той же подгруппы $H \subseteq G$.

**221.** Пусть $G$ — группа порядка $|G| = p$, $p$ — простое число. Показать, что $G$ — циклическая группа.

**222.** Показать, что циклическая группа — коммутативна.

**223.** Пусть $G$ — группа. Показать, что $\forall g \in G$ найдется такое целое $k$, что $g^k = e$.

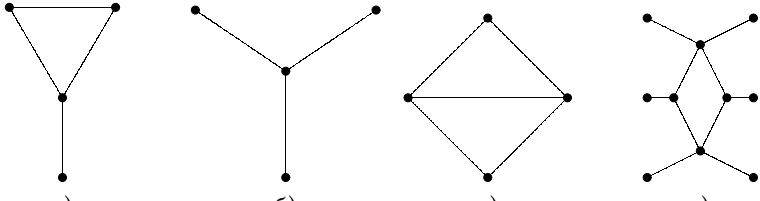
**224.** Показать, что число образующих циклической группы $G = \{g, g^2, \ldots, g^n = e\}$ равно значению функции Эйлера $\varphi(n)$ — количество чисел из множества $\{1, 2, \ldots, n-1\}$ взаимно простых с $n$.

**225.** Пусть $G$ — конечная абелева группа порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$, где $p_1, p_2, \ldots, p_k$ — простые различные числа; множество $A(p) = \{x \in G \mid |x| = p^{\alpha}\}$, где $\alpha$ принимает произвольные целые значения. Показать, что $A(p)$ — подгруппа.

**226.** Пусть $G_1, G_2$ — коммутативные группы порядков $|G_1| = 56$, $|G_2| = 64$. Найти количество возможных прямых разложений каждой из групп на циклические подгруппы.

**227.** Определить множество левых и правых смежных классов симметрической группы $S_3$ по подгруппе $H$, где $H$ — циклическая подгруппа с образующим элементом $\left(\begin{smallmatrix}1&2&3\\3&2&1\end{smallmatrix}\right)$. Доказать что $H$ — нормальный делитель.

**228.** Найти цикловой индекс группы самосовмещений, действующей на множестве а), б), в), г). Отдельно рассмотреть случаи действия группы на плоскости и в пространстве.



а) б) в) г)

**229.** Применить теорему Пойа для определения количества возможных раскрасок двумя и тремя красками вершин графов а), б), в), г) предыдущей задачи.

**230.** Используя теорему Пойа, найти количество различных ожерелий, которые можно составить из четырех бусинок пяти цветов. Рассмотреть отдельно варианты, когда группа самосовмещений (автоморфизмов) действует на плоскости и в пространстве.

## Алгоритмы и графы

**231.** *Задача о стоянке*. На некоторой улице с односторонним движением имеется $n$ мест для стоянки автомобилей, расположенных в ряд и перенумерованных от 1 до $m$. Муж везет в автомобиле жену. Внезапно жена просыпается и требует остановиться. Он послушно останавливается на первом свободном месте. Если нет свободных мест, к которым можно подъехать, не поворачивая обратно (т. е. если жена проснулась, когда машина достигла $k$-го места для стоянки, но все «позиции» $k, k+1, \ldots, m$ заняты), то муж приносит свои извинения и едет дальше.

Предположим, что это происходит с $n$ различными машинами, причем $j$-я жена просыпается в момент, когда машина находится перед местом $a_j$. Сколько имеется таких последовательностей $a_1, a_2, \ldots, a_n$, при которых все машины удачно припаркуются, предполагается, что первоначально улица пуста и никто со стоянки не уезжает? Составить алгоритм-программу вычисления всех возможных перестановок $a_1, a_2, \ldots, a_n$ и соответствующих им расположения автомобилей. Исходные данные — $m, n$. Например, если $m = n = 9$ и $a_1, a_2, \ldots, a_n = (3, 1, 4, 1, 5, 9, 2, 6, 5)$, то автомобили расположатся следующим образом: $2, 4, 1, 3, 5, 7, 8, 9, 6$.

**232.** *Фальшивая монета*. Банк «Золотой Ящик» получил информацию, что в последней партии из $n$ ровно одна монета фальшивая и отличается от других монет по весу. Для определения фальшивой монеты кассиру выдали только аптечные весы без гирь. Первым шагом кассир перенумеровал все монеты от 1 до $n$. Таким образом, каждая монета получила уникальный номер — целое число. После этого он начал взвешивать различные группы монет по составу, отмечая, какая из них больше, меньше или они равны.

Ваша задача — написать алгоритм-программу, которая по результатам взвешивания, выполненным кассиром, определит фальшивую монету или определит, что этого нельзя сделать.

Текстовый файл входных данных содержит следующую информацию.

Первая строка файла содержит два целых числа $n$ и $k$, разделенных пробелами, где $n$ — число монет в партии ($n \geqslant 2$) и $k$ — число взвешиваний, выполненных кассиром. Следующие $2k$ строк файла описывают результаты взвешивания. Каждое взвешивание представляется двумя строками файла.

Первая из них начинается числом $p_i$ ($1 \leqslant p_i \leqslant n/2$), которое обозначает число монет, участвовавших при взвешивании на каждой чашке весов. Следующие $p_i$ чисел на этой же строке файла являются номерами монет, которые располагались на левой чашке весов, и последние $p_i$ чисел на этой же строке файла являются номерами монет, которые располагались на правой чашке весов.

Вторая строка содержит один из знаков: $<$, $>$ или $=$.
Знак $<$ означает, что вес левой чашки меньше веса правой.
Знак $>$ означает, что вес левой чашки больше веса правой.
Знак $=$ означает, что вес левой чашки равен весу правой.

Ниже приведен пример файла входных данных. Результаты определения номера фальшивой монеты надо сохранить в текстовом файле. Если по выполненным взвешиваниям нельзя выявить фальшивую монету, то в качестве результата сохранить 0. Ниже приведен пример файла результатов.

| Файл исходных данных | Файл исходных данных |
|---|---|
| 5 3 | 6 4 |
| 2 1 2 3 4 | 3 1 2 3 4 5 6 |
| < | < |
| 1 1 4 | 1 1 2 |
| = | = |
| 1 2 5 | 2 1 3 4 5 |
| = | < |
| Файл результатов | 2 4 5 2 6 |
| 3 | > |
| | Файл результатов |
| | 0 |

**233.** *Грядки*. Садовый участок, имеющий прямоугольную форму и разбитый на квадратные клетки со стороной 1 метр, имеет ширину $n$ и длину $m$ метров. На этом участке вскопаны грядки. Грядкой называется совокупность клеток, удовлетворяющих условиям:

а) из любой клетки этой грядки можно попасть в любую другую клетку этой же грядки, последовательно переходя по грядке из клетки в клетку через их общую сторону;

б) никакие две грядки не пересекаются и не касаются друг друга ни по вертикальной, ни по горизонтальной сторонам клеток (касание грядок по углам клеток допускается).

Составьте алгоритм-программу расчета количества грядок на садовом участке.

Исходные данные задаются в текстовом файле. В первой строке располагаются два числа: $n$ и $m$, разделенные одним или несколькими пробелами. Далее следуют $n$ строк по $m$ символов. Символ «#» обозначает территорию грядки. Символ «.» соответствует незанятой территории. Других символов в исходном файле нет. В выходной текстовый файл вывести количество грядок на садовом участке.

Пример файла исходных данных

```
5 10
##. . . . . ##
. #. . #. . . #.
. ###. . . . ##
. . ##. . . . ##
. . . . . . . . #.
```

Выходной файл для данного примера

```
3
```

**234.** *Спуск с горы.*

```
          7
        3   8
      8   1   0
    2   7   4   4
  4   5   2   6   5
```

На рисунке показан числовой треугольник. Написать алгоритм-программу, которая находит максимальную сумму чисел в вершинах треугольника при движении сверху вниз по ребрам треугольника, т. е. из каждой вершины можно двигаться вниз влево или вправо. Например, для данного треугольника маршрут движения по узлам 7, 8, 1, 7, 2 дает сумму 25. Результаты расчетов сохранить в текстовом файле. Исходные данные представлены в текстовом файле, имеющем следующую структуру. Первая строка: $n$ — число уровней в треугольнике. Вторая и следующие строки содержат описание треугольника.

Пример входного файла:

```
5
7
3 8
8 1 0
```

```
2 7 4 4
4 5 2 6 5
```

Выходной файл данного примера

```
30 7 3 8 7 5
```

**235.** *Перестановка корзин*. Даны $2n$ корзин с шарами $A$ и $B$, расположенные в ряд, как на рисунке ($n \leqslant 5$):

$$|A|B|B|A| \quad | \quad |A|B|A|B|$$

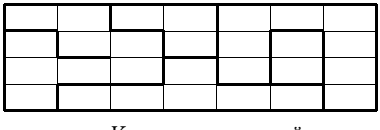
Два места под корзины пустые. Другие $n-1$ корзина содержат шары $A$ и $n-1$ корзина — шары $B$. Составьте алгоритм-программу, которая бы переставляла корзины с шарами $A$ с одной стороны, а корзины с шарами $B$ — с другой стороны. Правила перемещения корзин: разрешается переставлять на пустые места любые две смежные (пара) непустые корзины, сохраняя их первоначальный порядок.

Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: $2n$ — число корзин с пустыми местами. Вторая строка: $|A|B|B|A| \quad | \quad |A|B|A|B|$ — исходное расположение корзин. Расчетные данные сохранить в текстовом файле со следующей структурой. Каждая строка — состояние корзин после каждой перестановки.

**236.** *Комнаты музея*. Составьте алгоритм-программу определения числа комнат в музее и площадь каждой комнаты в клетках. План музея показан ниже на рисунке.

```
11  6 11  6  3 10  6
 7  9  6 13  5  7  5
 1 10 12  7 13 13  5
13 11 10  8 10 14 13
```

Цифровая карта



Карта перекрытий

Площадь музея состоит из клеток: $n$ рядов и $m$ столбцов. В каждой клетке такой матрицы (цифровая карта) проставляется число, в котором кодируется наличие стен у данной клетки. Значение числа в каждой клетке является суммой чисел: 1 (клетка имеет стену на западе), 2 (клетка имеет стену на севере), 4 (клетка имеет стену на востоке), 8 (клетка имеет стену на юге). Например, если в клетке стоит число 11 (11=8+2+1), то клетка имеет стену с южной стороны, с северной и с западной.

Исходные данные представляются в текстовом файле со следующей структурой. Первая строка: $n, m$ — размерность сетки. Вторая строка, третья и следующие строки содержат описание матрицы цифровой карты по строкам. Расчетные данные сохранить в текстовом файле со следующей структурой. Число

в первой строке — количество комнат в музее. Вторая строка — площадь каждой комнаты.

Пример входного файла:

```
4   7
11  6   11  6   3   10  6
7   9   6   13  5   7   5
1   10  12  7   13  13  5
13  11  10  8   10  14  13
```

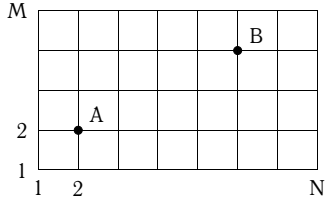
Выходной файл данного примера:

```
5
9   6   3   2   8
```

**237.** *Переворот бокалов*. На столе стоят в ряд $N$ бокалов, пронумерованных слева направо от 1 до $N$. Первоначально все бокалы стоят дном вниз. Над бокалами можно выполнять операцию «переворот». За один переворот ровно $M$ $(1 \leqslant M \leqslant N)$ любых выбранных бокалов переворачиваются так, что те бокалы, которые стояли дном вниз, оказываются перевернутыми вверх дном, а остальные из выбранных $M$ бокалов ставятся вниз дном.

Составить алгоритм-программу, которая за минимальное количество шагов позволит перевернуть все бокалы вверх дном или определить, что это сделать невозможно. Исходные данные $N$, $M$ задаются в текстовом файле. Результаты последовательных переворотов сохранить в текстовом файле.

**238.** *Течение воды*. Все улицы в некотором городе имеют направление с севера на юг ($N$ улиц) или с запада на восток ($M$ улиц), как на рисунке. Для каждого перекрестка улиц задается высота его над уровнем моря: $H[i, j]$, $i = \overline{1, M}$, $j = \overline{1, N}$, — матрица высот над уровнем моря. Требуется написать алгоритм-программу, который находит путь от перекрестка $A$ до перекрестка $B$ или в обратном направлении. Путь должен проходить по таким улицам, которые не ведут к возвышенностям (уровень при движении не должен возрастать).

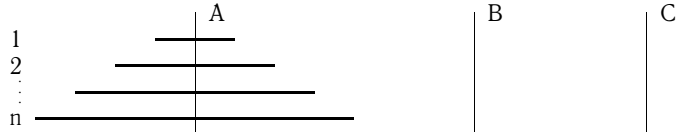



Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: $M$, $N$ — размерность матрицы и улиц. Вторая строка: $A = (ia, ja)$, $B = (ib, jb)$ — координаты двух перекрестков. Третья и следующие строки определяют значения элементов матрицы высот $H[i, j]$, $i = \overline{1, M}$, $j = \overline{1, N}$. Результаты расчетов сохранить в выходном текстовом файле со

следующей структурой. Каждая строка — координаты и высоты перекрестков, через которые пролегает маршрут.

**239.** *Бомба*. Требуется составить алгоритм-программу для определения наименьшей окружности (центр и минимальный радиус), охватывающей не менее $k$ из $n$ заданных точек на плоскости. Исходные точки на плоскости $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ задаются в текстовом файле. Результаты расчетов (координаты центра окружности, радиус ее и точки $(x_i, y_i)$, попадающие в окружность) сохранить в текстовом файле. *Решите эту же задачу, но искомая окружность должна включать все заданные точки* $(x_i, y_i)$.

**240.** *Ханойская башня*. Задача о ханойской башне проиллюстрирована на следующем рисунке:




Имеются три колышка $A, B, C$. На колышек $A$ нанизано $n$ дисков радиуса $1, 2, \ldots, n$ (каждый с отверстием в середине) таким образом, что диск радиуса $i$ является $i$-м сверху. Задача состоит в том, чтобы переместить все диски на колышек $C$ таким образом, чтобы диск радиуса $i$ был опять $i$-м сверху. За один раз разрешается перемещать только один диск с любого колышка на любой другой (можно пользоваться колышком $B$). При этом должно выполняться следующее условие: на каждом колышке ни в какой момент никакой диск не может находиться выше диска с меньшим номером. Составить рекурсивную алгоритм-программу требуемых перемещений дисков. Результаты расчетов: состояние каждого колышка на каждом шаге перекладывания дисков сохранять в текстовом файле.

**241.** *Числа*. Дана последовательность $n$ различных между собой целых чисел $a_1, a_2 \ldots, a_n$. *Трансформацией* называется следующая последовательность действий: некоторые из чисел увеличиваются на 1, если в последовательности оказываются два одинаковых числа, то одно из них исключается из последовательности. Выполним данную трансформацию $k$ раз, с целью до-

стигнуть минимального количества чисел в последовательности $a_1, a_2 \ldots, a_n$. Составить алгоритм-программу, которая вычисляет $p$ — количество чисел, оставшихся в последовательности. Исходные данные $n$, $k$ и последовательность $a_1, a_2 \ldots, a_n$ задаются в текстовом файле. Найденное число $p$ сохранить в текстовом файле.

Пример входного файла:

```
5  2
7  1  15 8  3
```

Выходной файл данного примера:

```
3
```

**242.** *Знакомства*. Имеется $n$ человек, где $1 \leqslant n \leqslant 100$. Каждому из них приписано некоторое число, которое определяет количество человек, с которыми ему предписано познакомиться. При этом знакомство взаимно, т. е. если человек с номером $i$ знакомится с человеком с номером $j$, то и человек с номером $j$ знакомится с человеком с номером $i$. Составить алгоритм-программу, задача которой состоит в следующем. Необходимо организовать знакомства, чтобы после их реализации участники могли быть разбиты на 2 команды таким образом, что в первой команде находятся участники, знакомые друг с другом (каждый знает каждого ), а во второй — только незнакомые (никто никого не знает). При этом численность первой команды должна быть максимальна. В случае невозможности реализации знакомств в выходной файл записать ответ «NO».

Формат файла входных данных. В первой строке входного файла находится число $n$. В каждой $i$-й из следующих $n$ строк находится число — количество человек, с которыми необходимо перезнакомиться человеку с номером $i$.

Формат файла выходных данных. В первой строке выходного файла находится численность первой команды или «NO». В случае, если реализация знакомств возможна, то в каждой из следующих строк должны находится 2 числа, разделенные пробелом — номера людей, которым необходимо познакомиться. Приведем примеры расчетов для конкретных исходных данных.

| Пример входного файла | Выходной файл данного примера |
|---|---|
| 4 | 3 |
| 3 | 1 2 |
| 2 | 1 3 |
| 2 | 1 4 |
| 1 | 2 3 |

**243.** *Обмен.* Каждому из $n$ жителей одного городка шериф присвоил личный номер — целое число от 1 до $n$ и сообщил его, а затем поручил секретарше разослать по почте жетоны с личными номерами. Но она разложила их по конвертам случайным образом. Для восстановления порядка, когда у жителя находится жетон с его номером, необходимо организовать обмен жетонами. Каждый житель может обменять в один день жетон, который находится у него, на другой только с одним другим жителем. В один день обмениваться жетонами могут любое количество пар жителей. Составить алгоритм-программу поиска минимального количества дней $m$, необходимых для того, чтобы все жители получили свои жетоны.

*Формат файла входных данных.* В первой строке содержит целое число $n$, $1 \leqslant n \leqslant 1000$, равное количеству жителей городка. Следующие $n$ строк содержат по одному числу — номеру жетона, т. е. в $(i+1)$-й строке, соответствующей $i$-у жителю, находится номер того жетона, который он получил по почте. Выходной файл данных содержит одну строку с найденным числом дней $m$.

**244.** *Кокосы.* В газетах 9 октября 1926 г. сообщалось о кораблекрушении судна в Индийском океане. Пять человек и одна обезьянка спаслись на необитаемом острове. В первый день пребывания на острове они собирали кокосы. Среди ночи один из них проснулся и решил разделить кокосы. Он разделил кокосы на пять равных частей. Однако остался один кокос, который он отдал обезьянке. Свою часть он взял с собой и пошел досыпать. Далее второй, третий, четвертый и пятый поступили аналогичным образом и каждый раз оставался один кокос, который они с удовольствием отдавали обезьянке. Напрашивается вопрос: сколько было кокосов? Однако мы решим несколько другую задачу (обратную). Составить алгоритм-программу поиска максимально возможного числа людей и одной обезьянки, которые могут проделать рассмотренную ночную операцию, если изначально известно количество собранных кокосов?

Исходные данные задаются в текстовом файле, который содержит последовательность целых чисел, каждое из которых является числом собранных кокосов. Последовательность чисел-кокосов заканчивается числом $-1$ — признак конца данных. Результаты расчетов числа людей и одной обезьянки для каждого случая исходных данных сохранить в выходном текстовом файле. Приведем примеры расчетов для конкретных исходных данных.

| Пример входного файла | Выходной файл данного примера |
|---|---|
| 25 кокосов | 3 человека и 1 обезьянка. |
| 20 кокосов | Нет решений. |
| 3121 кокосов | 5 человека и 1 обезьянка. |
| -1 | |

**245.** *Быстрый ФИЛ.* Фил работает в последнюю смену. После работы ровно в 2:00 утра Фил уезжает домой с автомобильной стоянки. Его маршрут пролегает по дороге, на которой установлены один или несколько светофоров. Фил всегда удивлялся, что, выбирая определенную скорость движения по маршруту и не изменяя ее, он мог иногда доехать к дому без задержки на светофорах, т. е. все светофоры он проезжал на зеленый свет. Скорость движения в городской черте не должна превосходить 60 миль/ч. Однако Фил не любил и тихо ездить. Минимальная скорость его движения 30 миль/ч. Составьте алгоритм-программу, которая находит все целочисленные скорости (в милях/ч), с которыми Фил может двигаться домой без остановки на светофорах, начало движения с автомобильной стоянки ровно в 2:00 утра. В этот момент времени все светофоры сбрасываются в зеленый цвет.

Исходные данные задаются в текстовом файле, в котором приводится набор данных для описания режимов работы светофоров. Последнее число в файле $(-1)$, является признаком конца данных в файле. Первое число $n$ каждого набора определяет количество светофоров на маршруте. Далее следуют $n$ наборов чисел: *Length Green Yellow Red* (*LGYR*) для каждого светофора, где $L$ — положительное действительное число, указывающее место расположения светофора от начала маршрута Фила; $GYR$ — интервал продолжительности времени в секундах непрерывного цвета светофора: зеленого, желтого и красного. Результаты расчетов допустимых целочисленных скоростей движения Фила сохранить в выходном текстовом файле:

| Пример входного файла | Выходной файл данного примера |
|---|---|
| 1<br>5.5 40 8 25 | 30, 32, 33, 36, 37, 38, 41, 42, 43, 44, 45, 48, 49, 50, 51, 52, 53, 54, 59, 60 |
| 3<br>10.7 10 2 75<br>12.5 12 5 57<br>17.93 15 4 67<br>-1 | 0 — признак отсутствия такой скорости |

**246.** *Парламент.* Парламент состоит из $N$ делегатов. Делегаты должны разделиться на группы (фракции), количество депутатов в каждой группе отличается от количества депутатов в любой другой группе. Каждый день каждая фракция посылает одного представителя в президиум. Парламент начинает работу в том случае, когда состав президиума отличен от составов президиумов предыдущих дней.

Составьте алгоритм-программу, которая бы определяла оптимальное число фракций и количество делегатов в каждой из них так, чтобы парламент мог работать как можно дольше. Число делегатов задается в исходном текстовом файле. Рассчитанные значения количества делегатов в каждой фракции, сортированные по возрастанию, сохранить в выходном текстовом файле. Приведем примеры оптимальных расчетов для $N = 7$ и $N = 31$.

| Пример входного файла | Выходной файл данного примера |
|---|---|
| 7 | 3 4 |
| 31 | 2 3 5 6 7 8 |

**247.** *Кот и Лиса.* Дан ориентированный граф $\Gamma(X, U, \Phi)$, вершины которого являются позициями в следующей игре. Участвуют два игрока — Кот и Лиса. Они двигают фишку из позиции в позицию по дугам графа. Множество позиций $X$ разделено на два подмножества $X = K \cup L$. В позициях $L$ ход делает Лиса, а в позициях $K$ — Кот. Если игра находится в позиции $x \in X$, владелец этой позиции выбирает произвольную выходящую дугу $(x, y) \in U$ и двигает фишку из $x$ в $y$. Игра начинается в некоторой позиции. Лиса выигрывает, если фишка оказалась в фиксированной вершине $z \in X$. Если за любое количество ходов фишка не попадает в эту позицию $z \in X$, то выигрывает Кот.

Составить алгоритм-программу, которая определяет все выигрышные стартовые позиции для Лисы при оптимальной стратегии обеих сторон. Считаем, что $X = \{1, 2, \ldots, n\}$, $1 \leqslant n \leqslant 1000$. Исходные данные представлены в текстовом файле, имеющем следующую структуру. Первая строка: $n, m, z$, где $n$ — число вершин, $m$ — число дуг, $z$ — заключительная позиция. Со второй строки записаны: $b_1, b_2, \ldots, b_n$ и $x_1y_1, x_2y_2, \ldots, x_my_m$, где $b_i = 1$, если вершина $i$ принадлежит Лисе; $b_i = 0$, если вершина $i$ принадлежит Коту; $x_iy_i$ — список дуг графа, $x_i$ — начальная вершина, $y_i$ — конечная вершина соответствующей дуги. Все параметры — целые числа, разделенные пробелами.

Результат вычислений представить в текстовом файле, в котором записать целых чисел $c_1, c_2, \ldots, c_n$, где $c_i = 1$, если позиция $i$ — выигрышная для Лисы; иначе $c_i = 0$.

Пример входного файла:

```
7  12 7
0  1  0  0  0  1  0
1  2  1  4  1  3
2  4  2  5
3  1  3  6  3  7
4  3  4  6
5  6
6  7
```

Выходной файл данного примера:

```
0  1  0  0  1  1  1
```

**248.** *Ящик с молоком*. Ящик имеет $n \times n$ ячеек для бутылок с молоком. Мистер Смит для каждого столбца и каждой строки заготовил отдельный листок бумаги, где записал наличие или отсутствие в ячейке бутылки молока: 1 — ячейка занята молоком, 0 — ячейка не занята молоком, но забыл проставить на каждом листе, чему он соответствует: строке или столбцу и все это вложил в коробку. В пути ящик попал под дождь, и часть записей на листах пропали. Испорченные цифры 1 и 0 были заменены цифрой 2. Составить алгоритм-программу, которая по таким записям восстанавливает исходное расположение бутылок с молоком, т. е. определить, какие записи относятся к строкам, а какие — к столбцам. Исходные испорченные записи по строкам и столбцам задаются в текстовом файле. Результаты сохранить в текстовом файле. Пример.

| Исходные данные | Восстановленные данные |
|---|---|
| 1) 0 1 2 1 0 | |
| 2) 2 1 1 2 0 | |
| 3) 2 1 0 0 1 | |
| 4) 1 2 1 1 0 | |
| 5) 1 2 1 0 1 | |
| 6) 1 2 1 0 1 | |
| 7) 0 0 0 1 1 | |
| 8) 2 2 2 2 2 | |
| 9) 1 1 0 0 1 | |
| 10) 1 0 0 1 0 | |

| | 4 | 1 | 10 | 7 | 5 |
|---|---|---|---|---|---|
| 6 | 1 | 0 | 1 | 0 | 1 |
| 9 | 1 | 1 | 0 | 0 | 1 |
| 3 | 1 | 1 | 0 | 0 | 1 |
| 2 | 1 | 1 | 1 | 1 | 0 |
| 8 | 0 | 0 | 0 | 1 | 1 |

**249.** *Длина объединения*. Текстовый файл содержит целые числа: $a_1, b_1, a_2, b_2, \ldots, a_n, b_n$. Эта последовательность определяет

на оси OX $n$ отрезков. Составить алгоритм-программу, которая определяет длину объединения всех отрезков. Результаты расчетов сохранить в текстовом файле. Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: $n$ — количество отрезков. Вторая строка, третья строка и т.д: $a_i, b_i$ — параметры соответствующего отрезка. Результаты расчетов сохранить в текстовом файле.

Пример входного файла:

```
 3
 0  2
-1  1
 0  1
```

Выходной файл данного примера:

```
3
```

**250.** *Площадь объединения*. Условие данной задачи — это условие предыдущей задачи, где вместо отрезков на прямой рассматриваются прямоугольники на плоскости, стороны которых параллельны осям координат. Составить алгоритм-программу определения площади объединения таких прямоугольников.

**251.** *Прямоугольные области*. Условие данной задачи — это условие предыдущей задачи. Однако требуется составить алгоритм-программу определения количества областей, на которые границы прямоугольников разбивают плоскость.

**252.** *Несвязные области*. На плоскость $XOY$ произвольным образом размещают $N$ произвольных прямоугольников со сторонами, параллельными осям координат. Взаимное расположение прямоугольников допускает их пересечение. Требуется составить алгоритм-программу определения количества несвязных областей, на которые прямоугольники разбивают плоскость $XOY$. *Входные и выходные данные*. В первой строке входного файла содержится число $N$. В каждой из следующих $N$ строках располагаются координаты одного прямоугольника: $(a_i, b_i)$ — координаты левого верхнего угла; $(c_i, d_i)$ — координаты правого нижнего угла. Выходной файл должен содержать единственное число, равное количеству несвязных областей.

Пример входного файла:

```
6
1  3  7  1
3  5  5  2
2  7  9  4
6  5  14 2
8  8  13 6
12 7  15 4
```

Выходной файл данного примера:
3

**253.** *Площадь участка.* Требуется составить алгоритм-программу определения площади участка, ограниченного замкнутой ломаной, составленной из отрезков единичной длины, параллельных осям координат. Исходные данные — координаты концов отрезков — задаются в текстовом файле. Найденную площадь сохранить в файле результатов. Отметим, что отрезки, составляющие границу участка, во входном файле могут следовать в произвольном порядке.

Пример входного файла:
24 — количество отрезков
0 0 1 0 1 0 2 0 2 0 3 0 4 0 5 0
1 1 2 1 2 1 2 2
1 2 2 2 2 2 3 2 3 2 4 2
0 3 1 3 1 3 2 3 2 3 3 3 3 3 4 3 4 3 5 3
0 0 0 1 0 1 0 2 0 2 0 3
1 1 1 2
3 0 3 1
4 0 4 1 4 1 4 2
5 0 5 1 5 1 5 2 5 2 5 3

Выходной файл данного примера:
11 — площадь участка

**254.** *Совмещение ломаных.* Две ломаные построены по ребрам сеточной области с целочисленными координатами. Требуется составить алгоритм-программу проверки совпадения двух ломаных, составленных из отрезков, с точностью до параллельного переноса и поворота на 90°, 180°, 270°. Исходные данные — число отрезков ломаных и значения координат их концов — определяются в текстовом файле. Выходной файл результатов должен содержать признак 1, если ломаные совпадают, и 0 — в противном случае.

Пример входного файла:
4 — количество отрезков первой ломаной
0 0 1 0 3 0 2 0 1 0 2 0 3 0 3 1
2 — количество отрезков второй ломаной
1 1 1 4 0 4 1 4

Выходной файл данного примера:
1 — ломаные совпадают

**255.** *Деление на равные половины.* Текстовый файл содержит последовательность целых чисел — веса элементов: $a_1, a_2, \ldots, a_n$. Составить алгоритм-программу деления этих предметов на две группы так, чтобы общие веса двух групп были мак-

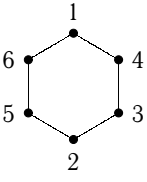
симально близкими друг к другу. Результаты расчетов сохранить в текстовом файле. Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: $n$ — количество чисел. Следующие строки содержат веса элементов $a_i$.

Пример входного файла:

```
5
6 3 1 4 5
```

Выходной файл данного примера:

```
6 4      — первая группа,
3 1 5    — вторая группа
```




**256.** *Простой круг.* Натуральные числа $1, 2, \ldots, n$ ($n$ — четное число) располагают по кругу как на диаграмме. Первое число по кругу всегда должно быть 1. Все остальные числа располагаются таким образом, чтобы сумма двух соседних чисел по кругу была простым числом, начиная с 1. Составить алгоритм–программу определения всех указанных расстановок. Исходное число задается в текстовом файле. Результаты расчетов сохранить в текстовом файле.

| Пример входного файла | Выходной файл данного примера |
|---|---|
| 6 | 1 4 3 2 5 6 1 |
| 8 | 1 2 3 8 5 6 7 4 1 |
| | 1 2 5 8 3 4 7 6 1 |

**257.** *Почтальон.* Дана последовательность из $N$ улиц (по названиям). Каждая улица соединяет два перекрестка. Первая и последняя буквы названия улицы определяют два перекрестка для этой улицы. Длина названия улицы определяет стоимость проезда по ней. Все названия улиц состоят из строчных символов алфавита. Например, название улицы «computer» показывает, что улица находится между перекрестками «c» и «r», а длина ее 8. Нет улиц, которые имеют одинаковые первые и последние символы. Есть не более одной улицы, напрямую соединяющей два любых перекрестка. Всегда есть путь между любыми двумя перекрестками. Число улиц с данным перекрестком называется степенью этого перекрестка. Есть не более двух перекрестков нечетной степени. Все остальные перекрестки — четной степени. Составить алгоритм-программу определения минимальной стоимости проезда по всем улицам, по крайней мере, один раз. Путешествие должно начаться и закончиться на одном и том же перекрестке. Стоимость проезда по улице равна ее длине.

Пример входного файла:
3
one
two
three

Выходной файл данного примера:
11

**258.** *Пересечение с отрезками.* Имеются $N$ отрезков, концы которых задаются двумя парами точек на плоскости: $(x1_i, y1_i)$ и $(x2_i, y2_i)$, $i = 1, 2, \ldots, N$. Требуется составить алгоритм-программу определения такой прямой линии, которая пересекает как можно большее количество заданных отрезков. Исходные данные определяются в текстовом файле, имеющем следующую структуру. Первая строка: $N$ — количество отрезков. Вторая и следующие строки: $x1_i, y1_i$ и $x2_i, y2_i$ — пары точек на плоскости (концы отрезков). Результаты расчетов сохранить в выходном текстовом файле, имеющем следующую структуру данных. Строки файла — номера отрезков в возрастающем порядке, которые пересекает найденная прямая. Если найденная прямая линия проходит через концы отрезков, то это учитывать как пересечение.

**259.** *Простые суммы.* Числа от 1 до $n$, $1 < n \leqslant 5000$. Требуется составить алгоритм-программу разбиения данных чисел на минимальное количество групп, в каждой из которых сумма является простым числом. Например, при $n = 8$ такими группами могут быть: $\{1, 4, 5, 6, 7\}, \{2, 3, 8\}$. Примечание: число 1 не считается простым. Файл исходных данных содержит число $n$. Выходной файл должен содержать $n$ чисел (номера групп), которые показывают в какую группу входит соответствующее по порядку число. Для нумерации групп должны использоваться идущие подряд натуральные числа, начиная с единицы.

Пример входного файла:
8

Выходной файл данного примера:
1 2 2 1 1 1 1 2

**260.** *Движение плота.* Квадратное озеро, покрытое многочисленными островами, задается матрицей размером $N \times N$. Каждый элемент матрицы — либо символ $'\#'$ — решетка, обозначающий остров, либо символ $'-'$ — минус, обозначающий участок воды. В верхнем левом углу озера находится квадратный плот размером $M \times M$ клеток. За один шаг плот может перемещаться на одну клетку по горизонтали или вертикали.

Составить алгоритм–программу для определения минимального числа шагов, за которое плот может достигнуть правого нижнего угла озера. Входной файл исходных данных содержит числа $N$ и $M$. В следующих $N$ строках располагается матрица, представляющая озеро. Выходной файл должен содержать единственное число — количество необходимых шагов. Если правого нижнего угла достичь невозможно, то выходной файл должен содержать число $-1$ (минус один).

Пример входного файла:

```
8 2
– – # – – – – –
– – – – – – – –
– – – – – # – –
# – # # – – – –
– – – – – – – –
– – – – – – # #
– – – – – – – –
– – – – – – – –
```

Выходной файл данного примера:

```
18
```

**261.** *Пират в подземелье.* В поисках драгоценных камней пират проваливается в подземелье. План подземелья — матрица $N \times M$ комнат с драгоценными камнями. Камни из одной комнаты имеют одинаковую стоимость. Пирату в каждой комнате разрешается взять с собой лишь один камень и следовать в любую другую соседнюю с ним комнату. Каждую из комнат пират может посещать неоднократно. Требуется составить алгоритм-программу определения маршрута посещения пиратом $K$ комнат лабиринта таким образом, чтобы он набрал камней на максимально возможную сумму. *Входные и выходные данные.* В первой строке входного файла содержатся числа $N$, $M$ и $K$. В следующих $N$ строках располагается матрица $N \times M$ лабиринта. Каждый элемент матрицы представляется стоимостью камня соответствующей комнаты. Маршрут начинается с левой верхней угловой комнаты лабиринта. Выходной файл должен содержать единственное число, равное общей стоимости взятых с собой камней.

Пример входного файла:

```
3 4 7
1 1 1 1
1 1 2 1
1 1 2 3
```

Выходной файл данного примера:

```
12
```

**262.** Оставить условие задачи № 261 и предписать пирату, чтобы он за $K$ шагов еще и вернулся в начальную комнату лабиринта.

**263.** *Задача Д. Андре.* Составить алгоритм-программу поиска всех способов заполнения числами $1, 2, \ldots, n$ из $n$ ячеек матрицы так, чтобы во всех строках и столбцах они располагались в возрастающем порядке (слева–направо и сверху–вниз). На рисунке показан пример заполнения этих ячеек. Исходное число $n$ задается в текстовом файле. Результаты заполнения сохранить в файле результатов.

| | | | |
|---|---|---|---|
| | | | 3 |
| | | 1 | 7 |
| | 4 | 6 | |
| 2 | 5 | | |

**264.** *Сумма чисел.* Дано натуральное число $m$. Вставить между некоторыми цифрами: $1, 2, \ldots, n$, записанными именно в таком порядке, знаки $+$, $-$ так, чтобы значением получившегося выражения было число $m$. Например, если $m = 122$, то подойдет следующая расстановка знаков: $12 + 34 - 5 - 6 + 78 + 9$. Требуется составить алгоритм-программу определения всех расстановок знаков $+$, $-$, отвечающих условию задачи. Исходное число $m$ задается во входном текстовом файле. Выходной текстовый файл должен содержать найденные расстановки знаков. Если требуемая расстановка знаков невозможна, то выходной файл должен содержать число $-1$.

**265.** *Обслуживание авиалиний.* Имеется некоторый город $M$, который связан маршрутами с городами $A_1, A_2, \ldots, A_n$. Пусть, согласно расписанию, маршрут $MA_iM$ обслуживается в интервале времени $[a_i, b_i]$. Другими словами, $a_i$ — это тот момент, начиная с которого самолет связан с маршрутом $MA_iM$, а $b_i$ — тот момент, когда эта связь прекращается. Таким образом, задано $n$ временных интервалов $[a_i, b_i]$, $i = 1, 2, \ldots, n$. Требуется составить алгоритм-программу определения минимального числа самолетов, достаточного для обслуживания всех рейсов.

Пример входного файла:
5 — количество городов
1 3 — интервалы времени
7 12
6 8
2 4
9 10

Выходной файл данного примера:
2 — самолета на все рейсы

**266.** *Симпатичный прием.* Генерал желает устроить юбилей с максимальным числом гостей из своих знакомых. Стремясь

сделать юбилейный вечер приятным, он должен организовать все так, чтобы на этом вечере присутствовали люди, симпатизирующие друг другу. Оказалось, что у генерала $n$ знакомых. Каждый из них получил соответствующий номер от 1 до $n$. Исходные данные задачи — это список пар симпатизирующих гостей генерала. Составить алгоритм-программу определения по исходным данным максимально возможное число гостей на юбилейном вечере и сохранить его в выходном файле результатов.

Пример входного файла:
5 — число знакомых у генерала
6 — число симпатизирующих пар
1 2
1 3
2 4
2 5
3 4
3 5

Выходной файл данного примера:
2 — приглашенных на вечере

**267.** *Таблица инверсий*. Составить алгоритм-программу определения таблицы инверсий $(d_1, d_2, \ldots, d_n)$ перестановки $(a_1, a_2, \ldots, a_n)$, где $d_j$ — число элементов, больших $j$ и расположенных левее $j$ (см. п. 2.13). Исходные данные — число $n$ и произвольная перестановка чисел $1, 2, \ldots, n$ — определяются в текстовом файле. Выходной файл результатов должен содержать найденную таблицу инверсий.

Пример входного файла:
5 — число $n$
5 3 4 2 1 — перестановка

Выходной файл данного примера:
4 3 1 1 0 — таблица инверсий

## Ответы

**1.** 1). $x_2{=}101100010_2$, $y_8{=}542_8$, $z_{16}{=}354_{16}$. 1а). $x_2{=}0.001_2\ldots$, $y_8 = 0.103_8\ldots$, $z_{16} = 0.218_{16}\ldots$ **2.** 1). $237.75_{10}$. 1а). $1362.412\ldots$ **4.** 1). $f_{\text{СДНФ}} = \overline{x}\,\overline{y}\,z \vee \overline{x}\,y\,z \vee x\,y\,\overline{z} \vee x\,y\,z$, $f_{\text{СКНФ}} = (x \vee y \vee z)(x \vee \overline{y} \vee z)$ $(\overline{x} \vee y \vee z)(\overline{x} \vee y \vee \overline{z})$, $f_{\text{Ж}} = z \oplus x\,y \oplus xz$. **5.** 1). $z \vee \overline{x}\,\overline{y} \vee x\,y$. 2). $\overline{y} \vee \overline{z}$. 3). $\overline{x}\,z \vee x\,\overline{z} \vee x\,y$ или $\overline{x}\,z \vee x\,\overline{z} \vee y\,z$. 4). $z \vee \overline{x}\,y \vee x\,\overline{y}$. 5). $y \vee \overline{x}\,\overline{z} \vee x\,z$. 6). $x \vee \overline{y} \vee z$. 7). $x\,\overline{y} \vee \overline{x}\,\overline{z} \vee y\,z$ **6.** 1). $\overline{x}\,z \vee \overline{x}\,\overline{w} \vee \overline{y}\,\overline{z}\,\overline{w} \vee \overline{y}\,z\,w$. 2). $\overline{y}\,\overline{w} \vee x\,y\,\overline{z} \vee \overline{x}\,y\,z$. 3). $\overline{x}\,\overline{y}\,\overline{z} \vee \overline{x}\,\overline{z}\,w \vee x\,\overline{y}\,z \vee x\,z\,w \vee \overline{y}\,w$. 4). $\overline{y}\,w \vee y\,\overline{w}$. 5). $\overline{x}\,y\,\overline{w} \vee \overline{y}\,\overline{z}\,w \vee x\,z\,w \vee \overline{x}\,z\,\overline{w} \vee x\,\overline{y}\,\overline{z}$. 6). $\overline{x}\,z\,\overline{w} \vee \overline{x}\,\overline{y}\,z \vee y\,\overline{z}\,w \vee x\,\overline{z}\,w \vee$ $\vee x\,y\,\overline{z} \vee x\,y\,w$. 7). $\overline{z}\,w \vee \overline{x}\,w \vee x\,\overline{y}\,\overline{z} \vee \overline{x}\,z\,\overline{y}$. 8). $\overline{w}$. 9). $y\,z \vee x\,\overline{y}\,\overline{w} \vee \overline{x}\,\overline{y}\,w \vee$ $\vee \overline{x}\,\overline{z}\,\overline{w}$. 10). $\overline{z}\,w \vee y\,\overline{z} \vee \overline{x}\,y\,w \vee x\,\overline{y}\,w$. **7.** *Указание*: см. п. 1.4.2 и п. 1.4.3. **8.** *Указание*: см. п. 1.4.2 и п. 1.4.3. **9.** *Указание*: смотрите п. 1.4.1. **10.** 1). Монотонная. **11.** 1). Несамодвойственная. **12.** 1). Лучше доказать от противного. Предположим, что $f(x,x) \neq 1$. Значит, $f(x,x) = 0$ для всех $x \in \{0,1\}$. Подставим в данное выражение $f(f(x,f(y,z)),f(f(x,y),f(x,z))) = 1$ вместо $y$ и $z$ переменную $x$. Получим $f(f(x,f(x,x)),f(f(x,x),f(x,x))) = 1$. Так как по предположению $f(x,x) = 0$, то $f(f(x,0),f(0,0)) = 1$ и далее $f(f(x,0),0) = 1$. Это верно для все $x$. Положив $x = 0$, получим $f(f(0,0),0) = 1$ или $f(0,0) = 1$. Это противоречит предположению, что $f(x,x) = 0$ для все $x$. **13.** Всего наборов $m = 2^n$. Тогда искомое число равно $C_m^0 + C_m^1 + \ldots + C_m^{k-1}$. **14.** 1). Пусть имеем $f_{\text{СДНФ}}(x_1, x_2, \ldots, x_n)$ и $g_{\text{СДНФ}}(y_1, y_2, \ldots, y_m)$. Так как переменные $x_1, x_2, \ldots, x_n$ и $y_1, y_2, \ldots, y_m$ — независимые, то после перемножения $f_{\text{СДНФ}} \wedge g_{\text{СДНФ}}$ каждое слагаемое будет являться конституентой 1, а значит, $f_{\text{СДНФ}} \wedge g_{\text{СДНФ}}$ тоже является СДНФ, слагаемых в которой $r \cdot s$. 2). Определим число слагаемых в функции $f(x_1, x_2, \ldots, x_n) \oplus g(y_1, y_2, \ldots, y_m)$. Представим ее в СДНФ: $\overline{f}_{\text{СДНФ}} \wedge g_{\text{СДНФ}} \vee f_{\text{СДНФ}} \wedge \overline{g}_{\text{СДНФ}}$. Число слагаемых в СДНФ для $\overline{f}_{\text{СДНФ}}$ равно $2^n - r$. Тогда искомое число слагаемых равно $(2^n - r) \times$ $\times\, s \;+ r \cdot (2^m - s)$. **15.** *Указание*: составьте (условную) таблицу истинности для каждой из функций. **16.** *Указание*: см. п. 1.5. **17.** *Указание*: см. п. 1.6.7. **21.** Множество $D$ всех сочетаний $C_n^k$ разобьем на два непересекающихся подмножества $D = D_1 \cup D_2$, где $D_1 \cap D_2$. Множество $D_1$ включает все сочетания с произвольным фиксированным элементом, $|D_1| = C_{n-1}^{k-1}$. Множество $D_2$ включает все сочетания без выделенного фиксированного элемента, $|D_2| = C_{n-1}^k$. Следовательно, $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$. **22.** Используйте бином Ньютона

$(1+x)^n = \sum_{k=0}^n C_n^k x^k$. **23.** Применить индукцию по $n$. **27.** $2^n$. **28.** Воспользуйтесь тождеством $(a+b)^n = \sum_{k=0}^n C_n^k a^k b^{n-k}$ **29.** Воспользуйтесь тождеством $(1+x)^n(1+x)^m = (1+x)^{n+m}$. **30.** Воспользуйтесь тождеством $(1+(1+1))^n = 3^n$. **31–33.** Воспользуйтесь тождеством $(1+x)^n = \sum_{k=0}^n C_n^k x^k$. **34.** Воспользуйтесь полиномиальным разложением формулы $(1+1+\ldots+1)^n = k^n$. **35.** 21. **36.** $C_{n-2}^2$. **37.** $2n^2 \times 2n^2$. **38.** $2n^2 \times (2n^2-2n)$. **39.** $2^{nm}$. **40.** $A_5^3, C_5^2 \cdot 3!$. **41.** $3^6$. **42.** 63. **43.** $C_n^2$ — двусторонних словарей, $A_n^2$ — односторонних словарей, $n$ — словарей при переводе по циклу. **44.** $A_m^4$. **45.** $P(5,7,3) = \dfrac{15!}{5!\,7!\,3!} = C_{15}^5 C_{10}^7 C_3^3$. **46.** $C_{n+m}^m$. **47.** $C_n^r$. **48.** $A_m^1 + A_m^2 + A_m^3$ — учитывается порядок имен. **49.** $P(2,2,2,1,1)$. **50.** $C_n^k C_m^k k!$. **51.** $2n!n!$ и $2n!\,n!/2n = n!\,(n-1)!$. **52.** $C_{12}^4 C_{15}^4 4!$. **53.** $n!/2n$, где деление на число $n$ означает совпадение соседей при циклическом перемещении исходной перестановки; в цифре 2 учитываются одни и те же соседи при отраженном (зеркальном) расположении исходной перестановки. **54.** $C_{10}^6(C_{10}^6-1)(C_{10}^6-2) = A_{C_{10}^6}^3$. **55.** $C_{3n}^n C_{2n}^n C_n^n$. **56.** $C_n^6((C_6^2 C_4^2 C_2^2)/3!)$. **57.** $C_n^6((C_6^2 C_4^2 C_2^2)/3!) = 15C_n^6$. **58.** $((2n)!/(2!)^n)/n!$ — неупорядоченное разбиение множества. **59.** Смотрите решение задач 7 и 8. **60.** $9! - C_3^1 7!\,3! + C_3^2 5!\,3!\,3! - C_3^3 3!\,3!\,3!\,3!$. **61.** $C_4^1 C_{48}^9$, $C_{52}^{10} - C_{48}^{10}$, $C_{52}^{10} - C_{48}^{10} - C_4^1 C_{48}^9$, $C_4^2 C_{48}^8$. **62.** $(C_4^1 C_{13}^3)(C_{13}^1)^3 + (C_4^2(C_{13}^2)^2)(C_{13}^1)^2$. **63.** $3^m$. **64.** $3^{17}$. **65.** $m^n$. **66.** $C_{n+m-1}^{m-1}$. **67.** $C_{(n-m)+m-1}^{m-1} = C_{n-1}^{m-1}$, $C_{(n-k)+m-2}^{m-2}$, $C_{(n-\sum a_i)+m-k-1}^{m-k-1}$. **68.** $C_{n_1+m-1}^{m-1} C_{n_2+m-1}^{m-1} C_{n_3+m-1}^{m-1}$. **69.** $C_8^2 (C_3^2)^6$. **70.** $C_{n+m-1}^{m-1}$. **71.** $5^4, A_5^4$. **72.** $C_{36}^5$. **73.** $C_5^3 C_{31}^2$, $C_{36}^5 - C_{31}^5 - C_5^1 C_{31}^4 - C_5^2 C_{31}^3$. **75.** $P(3,2,3,1)$. **76.** $C_{10}^3 C_7^4 C_3^3$. **77.** $(1+\alpha_1)(1+\alpha_2)\cdot\ldots\cdot(1+\alpha_n)$, $\dfrac{1-p_1^{\alpha_1}}{1-p_1}\dfrac{1-p_2^{\alpha_2}}{1-p_2}\cdot\ldots\cdot\dfrac{1-p_n^{\alpha_n}}{1-p_n}$. **79.** $n-[n/p]$ — столько чисел среди $1,2,\ldots,n$, не кратных $p$. Всего произведений $C_n^k$, тогда произведений, кратных $p$, равно $C_n^k - C_{n-[n/p]}^k$. **80.** $n! - m!\,(n-m+1)!$ (объединить $m$ элементов в один элемент). **81.** $2(C_n^2)^2 = \dfrac{n^2(n^2-1)}{2}$ способов поставить две ладьи так, чтобы они не угрожали друг другу. $(C_n^1)^2(n-1) = n^2(n-1)$ способов поставить две ладьи так, чтобы они угрожали друг другу. Для ответа на вопрос задачи осталось сравнить указанные числа. **83.** $C_{10}^2 + C_{10}^2 = 90$. **84.** Подсчитайте общее возможное число различных пар сотрудников и различных пар сотрудников на всех заседаниях комиссии и сравните их. **85.** Сказано, что никакая из 30 комиссий (в каждой комиссии 5 человек) не может иметь двух одинаковых сотрудников.

Подсчитайте число различных пар сотрудников во всех комиссиях и общее возможное число различных пар сотрудников в учреждении и сравните их. **86.** 18. **88.** 31. **91.** $C_{n+k-1}^{k-1}$, $C_{n-1}^{k-1}$. **92.** 205223000, см. п. 2.13. **93.** 27354186, см. п. 2.13. **95.** Указание: воспользуйтесь тем, что перестановка двух соседних меток $N$ и $M$ не оказывает влияния на произведение $2^{n-m}$. **96.** $3C_{3n}^{n}2^{n}$. **97.** $(kn)!/((k!)^{n}n!)$. **98.** $(30)!/((10!)^{3}3!)$, $(30)!/((3!)^{10}10!)$. **99.** $(C_4^2C_2^2/2!)(C_{32}^{16}C_{16}^{16})$. **100.** $(C_{10}^2C_8^2C_6^2C_4^2C_2^2)/5! = 10!/((2!)^25!)$. **101.** $9!/((1!)^1(2!)^41!4!)$. **102.** $9!/((3!)^33!)$. **103.** $P(2,2,2,2,2,2,1,1,1,1)$. **104.** $C_{15}^{8}C_{10}^{8}$. **105.** $4^8 - C_4^1 3^8 + C_4^2 2^8 - C_4^3 1^8$ — воспользоваться формулой включений и исключений для свойств: $P_i$ — пустой $i$-й этаж, $i = \overline{1,4}$. **107.** $\dfrac{n!}{n_1!\,n_2!\cdot\ldots\cdot n_k!}$. **108.** $(9^0 + 9^1) + 9^2 + 9^3 + \ldots + 9^n = \dfrac{1-9^{n+1}}{1-9}$. **109.** $C_{n+9-1}^{9-1}$. **110.** $C_{n+10-1}^{10-1} - 1$, где $-1$ обозначает число 0. **111.** $C_{n+1}^{k}$. **112.** $C_{n+k}^{k}$. **113.** Схема посева сортов пшеницы должна соответствовать латинскому квадрату $m \times m$. **114.** $C_{m+6-1}^{6-1}$. **115.** $k^m$. **116.** $\sum_{i=0}^{k}(-1)^i C_k^i (k-1)^m$. **117.** $\sum_{i=0}^{k-r}(-1)^i C_{r+i}^{r} C_k^{r+i}(k-r-i)^m = C_k^r \sum_{i=0}^{k-r}(-1)^i C_{k-r}^i (k-r-i)^m$. **118.** $\sum_{k=0}^{m}(-1)^k C_m^k m^{m-k} = m^m \sum_{k=0}^{m}(-1)^k \dfrac{C_m^k}{m^k}$. **119.** $\sum_{k=0}^{m}(-1)^k C_m^k (m-k)! = m!\sum_{k=0}^{m}\dfrac{(-1)^k}{k!} \simeq m!e^{-1}$. **120.** $\sum_{k=0}^{m-r}(-1)^k C_{r+k}^{r} C_m^{r+k}(m-r-k)! = \dfrac{m!}{r!}\sum_{k=0}^{m-r}\dfrac{(-1)^k}{k!} \simeq \dfrac{m!}{r!}e^{-1}$. **122.** Введем обозначения. $P_k$ — свойство, что $k$-я пара враждующих рыцарей сидит рядом, $k = 1, 2, \ldots, n$. $E(0)$ — размещение рыцарей, которые не обладают ни одним из указанных свойств, т. е. требуемые размещения по условию задачи. $E(0) = W(0) - W(1) + W(2) - \ldots + (-1)^n W(n)$, где $W(k)$ — количество размещений рыцарей, когда $k$ и более враждующих пар сидят вместе. Объединим каждую из $k$ указанных пар в один объект. Тогда имеем $2n-k$ объектов, которые можно расположить $(2n-k)!$ способами. В каждой из $k$ заданных пар врагов можно поменять местами $2^k$ способами. Выбор $k$ враждующих пар можно сделать $C_n^k$ способами. Следовательно, искомое число равно $E(0) = \sum\limits_{k=0}^{n}(-1)^k C_n^k 2^k (2n-k)!$. **123.** 458. **124.** 734. **126.** 20%, 60%, 70%. **127.** 2, 6, 3. **129–130.** Решение подобного уравнения рассмотрено в п. 2.7. **131.** $\dfrac{10!}{(2!)^5 5!}\dfrac{10!}{(2!)^5}$. **132.** $C_{n-2}^{3}$. Действительно, выбирая из $n-2$ три различных предмета

$C_{n-2}^3$ способами, можно однозначно отобразить их в разбиения, требуемые по условию. Если выбраны предметы с номерами $k_1 < k_2 < k_3$, то в исходном ряду их номера будут $k_1 < k_2 + 1 < k_3 + 2$.

**133.** $\sum\limits_{i=0}^{n}(-1)^i C_n^i \frac{(2n-i)!}{(2!)^{n-i}}$. **134.** $\left(\frac{4!}{((2!)^2 2!}2\cdot 2\right)^n (2n)! = (12)^n (2n)!$.

**135.** $2^n$, $\frac{(2n)!}{n!}$. **136.** $n+1$. **137.** Количество прямоугольников размера $i \times j$ равно $(n-i+1)(n-j+1)$. Каждый прямоугольник учитывается в сумме столько раз, какова его площадь. Тогда сумма равна $\sum\limits_{i=0}^{n}\sum\limits_{j=0}^{n}(n-i+1)(n-j+1)\cdot i\cdot j = \left(\sum\limits_{i=0}^{n}(n-i+1)\cdot i\right)^2 = \left(\frac{n(n+1)(n+2)}{6}\right)^2$.

**138.** а) $k!\,C_n^k C_m^k + 2(k-1)!\,C_n^{k-1}C_m^{k-1} + (k-2)!\,C_n^{k-2}C_m^{k-2}$; в) $n+1$.

**139.** $\frac{2x}{(1-x)^2} - \frac{10}{1-x} + \frac{49}{1-7x}$. **140.** $(n(n+2)/2)^2$. **141.** 1) $7 + 3^n$; 2) $u_{2n} = (9\cdot 3^{2n} + (-1)^n)/10$, $u_{2n+1} = (9\cdot 3^{2n+1} + 3(-1)^n)/10$, $n \geqslant 0$; 3) $(\mp 9)^n$; 4) $(\mp 4)^n$; 5) $(-1)^n(n-1)+2$; 6) $2^n$. **142.** 1) $C_{n+2}^2 - 2\,C_{n+1}^2 + 2\,C_n^2$; 2) $-2(-4)^n + 3\cdot 2^n + 5^n$; 3) $2^n - (n^2+n)/2$; 4) $2^n + 2^{n-2}C_n^2$; 5) $(n+2)(n+1)/2^{n+1}$; 6) $(2^{n+3} - 3 + (-1)^n)/6$. **144.** 1) $\frac{A\cdot n}{2^{n-1}} - \frac{B\cdot(n-1)}{2^n}$; 2) $A + B\cdot 3^n$; 3) $A((1+\sqrt{5}\,)/2)^n + B((1-\sqrt{5}\,)/2)^n$. **145.** $a_n = 2^{n+1}(2n+7)$, $b_n = 2^{n+1}(-2n-3)$. **146.** $(2^n - 3 + (-1)^n)/4$. **148.** $(\alpha^{n+1} - \beta^{n+1})/(\alpha - \beta)$. **149.** Использовать формулу включений и исключений. **150.** а) $R(x,C) = x(1+x)^3 + (1+3x+x^2)(1+x)$. **153.** а) $2^n$; с) $2\cdot 3^{n-1}$. Составить рекуррентное соотношение (см. п. 4.3). **154.** а) $(2^n + (-1)^n 2)/3$; б) $3(3^{n-1} - (-1)^{n-1})/4$. Составить рекуррентное соотношение (см. п. 4.3). **155.** $4^n$. **156.** $C_{2n}^n \sum\limits_{k=0}^{n}(C_n^k)^2 = (C_{2n}^n)^2$. Воспользуйтесь производящей функцией $\left(x + \frac{1}{x} + y + \frac{1}{y}\right)^{2n} = \sum\limits_{k=0}^{2n} C_{2n}^k \left(x+\frac{1}{x}\right)^k \left(y+\frac{1}{y}\right)^{2n-k}$ всех ломаных маршрутов длины $2n$ по рассматриваемой сетке. Свободный член в правой части является искомым числом для замкнутых маршрутов (см. п. 4.4). **157.** $C_{2n}^n$ (см. указание к задаче 136). **203.** а) 14; б) 18; в) 9. **204.** а) 43; б) 61. **205.** а) 37; б) 100; в) 158.

## Список литературы


1. *Ахо А.*, *Хопкрофт Дж.*, *Ульман Дж.* Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979. — 536 с.
2. *Берж К.* Теория графов и ее применения. — М.: ИЛ, 1962. — 320 с.
3. *Виленкин Н.Я.* Комбинаторика. — М.: Наука, 1969. — 328 с.
4. *Виноградов И.М.* Основы теории чисел. — М.: Наука, 1981. — 176 с.
5. *Гаврилов Г.П.*, *Сапоженко А.А.* Сборник задач по дискретной математике. — М.: Наука, 1977. — 368 с.
6. *Горбатов В.А.* Дискретная математика. — М.: Издательская группа АСТ, 2003. — 447 с.
7. *Грин Д.*, *Кнут Д.* Математические методы анализа алгоритмов. — М.: Мир, 1987. — 120 с.
8. *Гроссман И.*, *Магнус В.* Группы и их графы. — М.: Мир, 1971. — 248 с.
9. *Гудман С.*, *Хидетниеми С.* Введение в разработку и анализ алгоритмов. — М.: Мир, 1981. — 366 с.
10. *Клини С.* Математическая логика. — М.: Мир, 1993. — 480 с.
11. *Кнут Д.* Искусство программирования для ЭВМ: В 3 т. Т. 3: Сортировка и поиск. — М.: Мир, 1978. — 844 с.
12. Комбинаторный анализ. Задачи и упражнения: Учеб. пособие / Под ред. К.А. Рыбникова — М.: Наука, 1982 — 366 с.
13. *Кристофидес М.* Теория графов. — М.: Мир, 1978. — 422 с.
14. *Кузнецов О.П.*, *Андельсон-Вельский Г.М.* Дискретная математика для инженеров. — М.: Энергоатомиздат, 1988. — 479 с.
15. *Курош А.Г.* Лекции по общей алгебре. — М.: Наука, 1973. — 400 с.
16. *Линдон Р.* Заметки по логике. — М.: Мир, 1968. — 128 с.
17. *Липский В.* Комбинаторика для программистов. — М.: Мир, 1988. — 213 с.
18. *Мендельсон Э.* Введение в математическую логику. — М.: Наука, 1984. — 319 с.
19. *Нефедов В.Н.*, *Осипова В.А.* Курс дискретной математики. — М.: МАИ, 1992. — 262 с.
20. *Новиков Ф.А.* Дискретная математика для программистов. — СПб.: Питер, 2001. — 302 с.
21. *Оре О.* Теория графов. — М.: Наука, 1980. — 336 с.
22. *Препарата Ф.*, *Шеймос М.* Вычислительная геометрия. — М.: Мир, 1989. — 478 с.

23. *Рейнгольц Э.*, *Нивергельт Ю.*, *Део Н.* Комбинаторные алгоритмы. Теория и практика. — М.: Мир, 1980. — 1980 с.
24. *Риордан Дж.* Введение в комбинаторный анализ. — М.: ИЛ, 1963. — 287 с.
25. *Рыбников К. А.* Введение в комбинаторный анализ. — М.: МГУ, 1972. — 255 с.
26. *Столл Р.* Множества, логика, аксиоматические теории. — М. Просвещение, 1968. — 231 с.
27. *Уилсон Р. Дж.* Введение в теорию графов. — М.: Мир, 1977. — 208 с.
28. *Форд Л. Р.*, *Фалкерсон Д. Р.* Потоки в сетях. — М.: Мир, 1966. — 276 с.
29. *Харрари Ф.* Теория графов. — М.: Мир, 1973. — 300 с.
30. *Харрари Ф.*, *Палмер Э.* Перечисление графов. — М.: Мир, 1977. — 324 с.
31. *Холл М.* Комбинаторика. — М.: Мир, 1970. — 424 с.
32. *Холл П.* Вычислительные структуры. Введение в нечисленное программирование. — М.: Мир, 1978. — 214 с.
33. *Яблонский С. В.* Введение в дискретную математику. — М.: Наука, 1986. — 384 с.

## Предметный указатель

Учебное издание

*ИВАНОВ Борис Николаевич*

**ДИСКРЕТНАЯ МАТЕМАТИКА**
**АЛГОРИТМЫ И ПРОГРАММЫ**
**ПОЛНЫЙ КУРС**

Редактор *И.Л. Легостаева*
Оригинал-макет: *А.М. Садовский*
Оформление переплета: *Н.В. Гришина*

Подписано в печать 26.12.06. Формат 60 90/16.
Бумага офсетная. Печать офсетная. Усл. печ. л. 25,5.
Уч.-изд. л. 25,5. Тираж 1500 экз. Заказ №

Издательская фирма «Физико-математическая литература»
МАИК «Наука/Интерпериодика»
117997, Москва, ул. Профсоюзная, 90
E-mail: fizmat@maik.ru, fmlsale@maik.ru;
http://www.fml.ru

Отпечатано с готовых диапозитивов
в ППП «Типография «Наука»
121099, г. Москва, Шубинский пер., 6

ISBN 978-5-9221-0787-7