


ТЕХНИЧЕСКИЙ  УНИВЕРСИТЕТ

Б. Н. ИВАНОВ

ДИСКРЕТНАЯ МАТЕМАТИКА

АЛГОРИТМЫ
И ПРОГРАММЫ



Москва
Лаборатория Базовых Знаний
2003

ББК 32.973.3
УДК 519(075.8)+681.142.2
И 20

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

Иванов Б. Н.

И20 Дискретная математика. Алгоритмы и программы: Учеб. пособие/Б. Н. Иванов. — М.: Лаборатория Базовых Знаний, 2003. — 288 с.: ил.

ISBN 5-93208-093-0

Книга посвящена современному курсу дискретной математики. 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.

ББК 32.973.3
УДК 519(075.8)+681.142.2

Серия «Технический университет»

Учебное издание

Иванов Борис Николаевич

ДИСКРЕТНАЯ МАТЕМАТИКА. АЛГОРИТМЫ И ПРОГРАММЫ

Художник *Н. Лозинская*
Компьютерная верстка *В. Носенко*

Подписано в печать 22.04.01. Формат 60×90¹/₁₆.
Гарнитура Ньютон. Бумага офсетная. Печать офсетная.
Усл. печ. л. 18,0. Тираж 3000 экз. *Заказ 2697*

Издательство «Лаборатория Базовых Знаний»
Телефон (095)955-0398. E-mail: lbz@aha.ru

Лицензия на издательскую деятельность № 066140 от 12 октября 1998 г.
Отпечатано с готовых диапозитивов в полиграфической фирме
«Полиграфист». 160001, г. Вологда, ул. Челюскинцев, 3.

ISBN 5-93208-093-0

© Б. Н. Иванов, 2002
© Лаборатория Базовых Знаний, 2003

Содержание

Предисловие	6
Глава 1. Комбинаторные схемы	8
1.1. Правило суммы	8
1.2. Правило прямого произведения	9
1.3. Размещения с повторениями	9
1.4. Размещения без повторений	10
1.5. Перестановки	11
1.6. Сочетания	11
1.7. Сочетания с повторениями	12
1.8. Перестановки с повторениями, мультимножества	14
1.9. Упорядоченные разбиения множества	15
1.10. Неупорядоченные разбиения множества	16
1.11. Полиномиальная формула	18
1.12. Бином Ньютона	19
1.13. Инверсии	20
1.14. Обратные перестановки	21
Глава 2. Представление абстрактных объектов	24
2.1. Представление последовательностей	24
2.1.1. Смежное представление	24
2.1.2. Характеристические векторы	25
2.1.3. Связанное размещение	26
2.2. Представление деревьев	31
2.2.1. Представление деревьев на связанной памяти	32
2.2.2. Представление деревьев на смежной памяти	33
2.3. Представление множеств	37
Глава 3. Методы подсчета и оценивания	39
3.1. Производящие функции	39
3.1.1. Линейные операции	41
3.1.2. Сдвиг начала вправо	41
3.1.3. Сдвиг начала влево	42
3.1.4. Частичные суммы	42
3.1.5. Дополнительные частичные суммы	42
3.1.6. Изменение масштаба	43
3.1.7. Свертка	44
3.2. Линейные рекуррентные соотношения	49
3.3. Неоднородные линейные рекуррентные соотношения	51
3.4. Обобщенное правило произведения	53
3.5. Принцип включения и исключения	56
3.6. Ладейные многочлены и многочлены попаданий	59
3.6.1. Ладейные многочлены	60
3.6.2. Многочлены попаданий	63

Глава 4. Генерация комбинаторных объектов	66
4.1. Поиск с возвратом	66
4.2. Перестановки различных элементов	68
4.3. Эффективное порождение перестановок	71
4.4. Порождение подмножеств множества	76
4.5. Генерация размещений с повторениями	79
4.6. Порождение сочетаний	80
4.7. Порождение композиций и разбиений	83
4.8. Генерация случайных перестановок	89
Глава 5. Сортировка и поиск	91
5.1. Сортировка вставками	92
5.2. Пузырьковая сортировка	93
5.3. Сортировка перечислением	94
5.4. Сортировка всплытием Флойда	95
5.5. Последовательный поиск	102
5.6. Логарифмический поиск	104
5.7. Сортировка с вычисляемыми адресами	106
Глава 6. Введение в теорию графов. Алгоритмы на графах	110
6.1. Основные понятия и определения	ПО
6.2. Представления графов	114
6.2.1. Матрица смежности графа	114
6.2.2. Матрица инцидентности графа	115
6.2.3. Матрица весов графа	116
6.2.4. Список ребер графа	116
6.2.5. Структура смежности графа	117
6.3. Метод поиска в глубину	117
6.4. Отношение эквивалентности	124
6.5. Связные компоненты	125
6.6. Выделение компонент связности	126
6.7. Эйлеровы графы	130
6.8. Остовные деревья	137
6.8.1. Жадный алгоритм построения минимального остовного дерева	139
6.8.2. Алгоритм ближайшего соседа построения остовного дерева	145
6.9. Кратчайшие пути на графе	151
6.10. Потоки в сетях	156
6.11. Клики, независимые множества	160
6.12. Циклы, фундаментальные множества циклов	172
6.13. Листы и блоки	177
6.13.1. Листы	178
6.13.2. Блоки	180
6.13.3. Поиск блоков в глубину	182

6.14. Двудольные графы	185
6.14.1. Условия существования двудольных графов	185
6.14.2. Паросочетания	186
6.14.3. Алгоритм определения максимального паросочетания	186
6.14.4. Системы различных представителей	189
6.14.5. Связь системы различных представителей и двудольных графов	189
6.14.6. Задача о назначениях	190
6.15. Хроматические графы	194
6.16. Диаметр, радиус и центры графа	196
Глава 7. Введение в теорию групп. Приложения	197
7.1. Определение группы	197
7.2. Гомоморфизм групп	198
7.3. Смежные классы	199
7.4. Строение коммутативных (абелевых) групп	203
7.5. Строение некоммутирующих групп	207
7.6. Симметрическая группа подстановок	208
7.7. Действие групп на множестве	212
7.8. Цикловой индекс группы	217
7.9. Теория перечисления Пойа	218
7.10. Цикловая структура групп подстановок	223
7.10.1. Цикловой индекс группы, действующей на себе	224
7.10.2. Цикловой индекс циклической группы	224
7.10.3. Цикловой индекс симметрической группы	225
Глава 8. Элементы теории чисел	227
8.1. Наибольший общий делитель	227
8.2. Наименьшее общее кратное	228
8.3. Простые числа	228
8.4. Сравнения, свойства сравнений	232
8.5. Полная система вычетов	233
8.6. Приведенная система вычетов	234
8.7. Функция Эйлера	234
8.8. Функция Мёбиуса. Формула обращения Мёбиуса	238
Задачи и упражнения	240
Ответы	281
Литература	285
Предметный указатель	286

Предисловие

Традиционно к дискретной математике относят такие области математического знания, как комбинаторика, теория чисел, математическая логика, теория алгебраических систем, теория 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. Дискретная математика и примыкающие к ней дисциплины изучаются во всех университетах и институтах, где осуществляется подготовка специалистов в области программирования, математики, а также по экономическим, техническим и гуманитарным направлениям.

Цель написания учебного пособия — 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. Для любого заинтересованного читателя рассматриваемый в пособии набор приемов и правил алгоритмического характера может составить хорошую основу формирования культуры разработки, анализа и программной реализации алгоритмов.

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 некоторые важные обозначения. Множества будем обозначать заглавными буквами. Множества состоят из элементов, которые будем обозначать малыми буквами. Так, запись $a \in A$ обозначает, 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.

$A \times B = \{(a, b) | a \in A, b \in B\}$ — прямое произведение множеств.

$A \cup B = \{x | x \in A \vee x \in B\}$ — объединение множеств.

$A \cap B = \{x | x \in A \wedge x \in B\}$ — пересечение множеств.

$A \setminus B = \{x | x \in A \wedge x \notin B\}$ — разность множеств.

\emptyset — пустое множество.

U — универсальное множество.

$\bar{A} = U \setminus A = \{x | x \notin A\}$ — дополнение множества.

1.1. Правило суммы

Пусть A и B — конечные множества такие, что $A \cap B = \emptyset$, $|A| = m$ и $|B| = n$. Тогда $|A \cup B| = m + n$.

Практическая компьютерная реализация большого количества рассматриваемых задач потребовала включения в книгу специальных разделов дискретной математики, таких как представление абстрактных объектов, сортировка и поиск, порождение комбинаторных объектов. Однако увеличение объема материала пособия только лишь усилило многие вопросы, позволило уточнить их суть, а также 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 пространственные программы. Необходимый уровень математического образования соответствует типичной подготовке студента, прослушавшего ряд основных математических курсов, таких как математический анализ, аналитическая геометрия, линейная алгебра.

Учебное пособие ориентировано на семестровый 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.

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

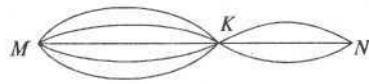
Интерпретация. Если элемент $a \in A$ можно выбрать m способами, а элемент $b \in B$ — n способами, то выбор элемента $x \in A \cup B$ можно осуществить $m + n$ способами. Пусть X_1, X_2, \dots, X_k — попарно непересекающиеся множества, $X_i \cap X_j = \emptyset$, где $i \neq j$. Тогда, очевидно, выполняется равенство
$$\left| \bigcup_{i=1}^k X_i \right| = \sum_{i=1}^k |X_i|.$$

1.2. 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, \dots, X_k — произвольные множества, $|X_i| = n_i, i = \overline{1, k}$. Тогда

$$|X_1 \times X_2 \times \dots \times X_k| = |\{(x_1, x_2, \dots, x_k) | x_i \in X_i, i = \overline{1, k}\}| = n_1 \cdot n_2 \cdot \dots \cdot n_k.$$



Задача. 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.

Решение. 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 $i = 1, 2, 3, 4, 5; j = 1, 2, 3$. Значит, $S \times T$ — это множество всех дорог из M в N , количество которых равно $|S \times T| = 5 \cdot 3 = 15$.

1.3. Размещения с повторениями

Задача формулируется следующим образом. Имеются предметы n различных видов a_1, a_2, \dots, a_n . Из них составляют всевозможные расстановки длины k . Например, $a_2 a_1 a_3 a_4 a_3 a_2 a_1$ — 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, \dots, X_k такие, что $X_1 = X_2 = \dots = X_k = \{a_1, a_2, \dots, a_n\}$. Тогда все размещения с повторениями составят множество $X_1 \times X_2 \times \dots \times X_k$. По правилу прямого произведения получаем, что общее число размещений с повторениями из n по k равно $|X_1 \times X_2 \times \dots \times X_k| = n^k$.

Задача. Найти количество всех пятизначных чисел.

Решение. Введем пять множеств: $A_2 = A_3 = A_4 = A_5 = \{0, 1, \dots, 9\}$, $A_1 = \{1, 2, \dots, 9\}$. 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 $9 \cdot 10 \cdot 10 \cdot 10 \cdot 10 = 90000$.

1.4. Размещения без повторений

Имеется n различных предметов a_1, a_2, \dots, 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 $A_n^k = n(n-1)\dots(n-k+1) = n! / (n-k)!$. Напомним, что $n! = n(n-1)\dots 1$ 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 $A_{17}^3 = 17 \cdot 16 \cdot 15 = 4080$.

1.5. 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 $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ элементов $1, 2, \dots, n$ записывают и в матричной форме $\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix}$, где верхняя строка — это по-

рядковые номера $1, 2, \dots, n$ позиций элементов в перестановке; нижняя строка — тот же набор чисел $1, 2, \dots, n$, взятых в каком-либо порядке; π_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 верхней 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 о ладьях. 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 $\pi = \begin{pmatrix} 1 & 2 & \dots & 8 \\ \pi_1 & \pi_2 & \dots & \pi_8 \end{pmatrix}$. Верхняя строка перестановки — это номера горизонталей, нижняя — вертикалей, пересечение которых определяет положение ладей на доске. Следовательно, число расстановок равно числу перестановок $P_8 = 8!$ из 8 элементов.

1.6. Сочетания

В тех случаях, когда нас не интересует порядок элементов в расстановке, а интересует лишь ее состав, то 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 эле-

ментов. 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, то по правилу произведения можно записать $C_n^k \times k! = A_n^k$. Тогда $C_n^k = \frac{A_n^k}{k!}$ или $C_n^k = \frac{n!}{k!(n-k)!}$ и

$$C_n^k = C_n^{n-k}.$$

Задача о прямоугольниках. 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 $m + 1$ положений. Тогда число способов их выбора равно C_{m+1}^2 . Вертикальные стороны можно выбрать C_{n+1}^2 способами. По правилу прямого произведения заключаем, что количество прямоугольников равно $C_{m+1}^2 \cdot C_{n+1}^2$.

1	2	3	...	n
2				
3				
...				
m				

1.7. 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 $\bar{C}_n^k = C_{n+k-1}^{n-1} = C_{n+k-1}^k$. Выведем данную формулу.

Пусть a, b, c, \dots, d — это исходные различные типы элементов, количество которых n . Рассмотрим произвольное сочетание с повторениями $cbbcacdda\d\dacbbb$ из данных типов элементов. Так как порядок элементов в сочетаниях не учитывается, то расстановку можно записать и так: $aa\dots a \setminus bb\dots b \setminus cc\dots c \setminus \dots \setminus dd\dots d$, где элементы каждого из типов упорядочены и завершаются вертика-

льной 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 из $n + k - 1$ места $n - 1$ место для положений вертикальных линий. Это можно сделать C_{n+k-1}^{n-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 $\bar{C}_3^{63} = C_{3+63-1}^{63} = C_{3+63-1}^2 = \frac{65 \cdot 64}{2} = 2080$.

Задача. Найти количество целочисленных решений системы

$$x_1 + x_2 + \dots + x_n = k, \quad k \geq 0, \quad x_i \geq 0, \quad i = 1, 2, \dots, n; \quad n > 1.$$

Решение. Рассмотрим следующую интерпретацию решения уравнения. Каждое значение $x_i = 1_i + 1_i + \dots + 1_i$ 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 $\{1_1, 1_2, \dots, 1_n\}$, значение каждого из них равно единице. Теперь любое решение исходного уравнения можно представить как сумму, составленную из k произвольных единиц множества $\{1_1, 1_2, \dots, 1_n\}$. 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 $\bar{C}_n^k = C_{n+k-1}^{n-1} = C_{n+k-1}^k$.

1.8. 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: $M = \{3 \cdot a, 2 \cdot b, 1 \cdot c, 4 \cdot d\}$. Таким образом, искомые перестановки с повторениями — это перестановки элементов мультимножества. Если бы мы рассматривали все элементы множества M как различные, обозначив их $a_1, a_2, a_3, b_1, b_2, c_1, d_1, d_2, d_3, d_4$, то получили бы $10!$ перестановок, но после отбрасывания индексов многие из них оказались бы одинаковыми. Фактически каждая перестановка множества M встретилась бы ровно $3! \cdot 2! \cdot 1! \cdot 4!$ раз, поскольку в любой перестановке M 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 $\frac{10!}{3! \cdot 2! \cdot 1! \cdot 4!}$. В применении к общему случаю те же рассуждения показывают, что число перестановок любого мультимножества (перестановки с повторениями) равно полиномиальному коэффициенту

$$P(n_1, n_2, \dots, n_k) = \binom{n}{n_1, n_2, \dots, n_k} = \frac{n!}{n_1! n_2! \dots n_k!}$$

где $n = n_1 + n_2 + \dots + n_k$ — 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 $C_{n-1}^{n_1}$ способами. Из оставшихся $n - n_1$ мест элементы второго типа занимают n_2 места, которые можно выбрать $C_{n-n_1-1}^{n_2}$ способами. Те же рассуждения показывают, что элементы k -го типа можно расположить в перестановке $C_{n-n_1-n_2-\dots-n_{k-1}}^{n_k}$ способами. Согласно правилу прямо-

го произведения, число перестановок с повторениями равно

$$P(n_1, n_2, \dots, n_k) = C_n^{n_1} C_{n-n_1}^{n_2} \dots C_{n-n_1-\dots-n_{k-1}}^{n_k} = \frac{n!}{n_1! n_2! \dots n_k!}$$

Задача. Сколько существует различных перестановок из букв слова «Уссури»?

Решение. $P(2у, 1и, 1р, 2с) = \frac{6!}{2! \cdot 1! \cdot 1! \cdot 2!} = 180.$

1.9. Упорядоченные разбиения множества

Подсчитаем число разбиений конечного множества S , где $|S| = n$, на k различных подмножеств $S = S_1 \cup S_2 \cup \dots \cup S_k$, попарно не пересекающихся, $|S_i| = n_i$, $i = 1, 2, \dots, k$ и $\sum_{i=1}^k n_i = n$. Последовательность различных S_1, S_2, \dots, S_k рассматривается как упорядоченная последовательность подмножеств. При формировании упорядоченной S_1, S_2, \dots, 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 $n - n_1 - n_2 - \dots - n_{k-1}$ элементов $C_{n-n_1-\dots-n_{k-1}}^{n_k}$ способами. По правилу прямого произведения получаем, что общее число упорядоченных разбиений множества S на k подмножеств равно

$$C_n^{n_1} C_{n-n_1}^{n_2} \dots C_{n-n_1-\dots-n_{k-1}}^{n_k} = \frac{n!}{n_1! n_2! \dots n_k!},$$

что совпадает с числом $P(n_1, n_2, \dots, n_k)$ 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 $S = \{1, 2, \dots, n\}$ в перестановке на подмножества S_1, S_2, \dots, 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 непересекающиеся подмножества $S_1 \cup S_2$ и ... и $S_k = S$ 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, \dots, 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 $n_1 + n_2 + \dots + n_k = n$. Как установлено, число таких разложений равно

$$C_n^{n_1} C_{n-n_1}^{n_2} \dots C_{n-n_1-\dots-n_{k-1}}^{n_k} = \frac{n!}{n_1! n_2! \dots n_k!}.$$

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 $C_{25}^{19} \cdot C_6^3 \cdot C_3^3 = \frac{25!}{19! \times 3! \times 3!}$.

1.10. Неупорядоченные разбиения множества

Подсчитаем, сколькими способами можно разбить множество S , где $|S| = n$, на подмножества, среди которых для каждого $i = 1, 2, \dots, n$ имеется $m_i > 0$ подмножеств с i элементами. Тогда верно, что $\sum_{i=1}^n i \cdot m_i = n$. Данное разбиение позволяет представить исходное множество следующим образом:

$$S = \bigcup_{j=1}^{m_1} S_{1j} \cup \bigcup_{j=1}^{m_2} S_{2j} \cup \dots \cup \bigcup_{j=1}^{m_n} S_{nj} = \bigcup_{i=1}^n \bigcup_{j=1}^{m_i} S_{ij},$$

где S_{ij} попарно не пересекаются и $|S_{i1}| = |S_{i2}| = \dots = |S_{im_i}| = i$ для каждого $i = 1, 2, \dots, n$. Порядок подмножеств в разбиении не является существенным. Так, например, разбиения множества $S = \{1, 2, 3, 4, 5\}$ вида

- {1, 3}, {4}, {25};
- {1, 3}, {25}, {4};
- {25}, {1, 3}, {4};
- {4}, {1, 3}, {25};

считаются одинаковыми.

Обозначим число неупорядоченных разбиений множества S через $N(m_1, m_2, \dots, m_n)$. Рассмотрим схему формирования *упорядоченных разбиений* для представления $n = 1 \cdot m_1 + 2 \cdot m_2 + \dots + n \cdot m_n$:

$$\begin{aligned}
 & \underbrace{C_n^1 C_{n-1}^1 \dots C_{n-1}^{m_1}}_{m_1} \underbrace{C_{n-1}^{m_2} C_{n-1-m_1}^{m_2} \dots C_{n-1-m_1-1}^{m_2}}_{m_2} \underbrace{C_{n-1-m_1-2m_2}^{m_3} C_{n-1-m_1-2m_2-1}^{m_3} \dots C_{n-1-m_1-2m_2-1}^{m_3}}_{m_3} \dots \underbrace{C_{n-1-m_1-2m_2-\dots-(n-1)m_{n-1}}^n}_{m_n} \\
 &= \frac{n!}{\underbrace{1! \dots 1!}_{m_1} \underbrace{2! \dots 2!}_{m_2} \dots \underbrace{n! \dots n!}_{m_n}} = \frac{n!}{(1!)^{m_1} (2!)^{m_2} \dots (n!)^{m_n}}
 \end{aligned}$$

Вспользуемся интерпретацией формирования упорядоченных разбиений как разложения n различных шаров по различным $m_1 + m_2 + \dots + m_n$ 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 m_i корзин 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 $m_1! m_2! \dots m_n!$ раз меньше, чем упорядоченных. Следовательно,

$$N(m_1, m_2, \dots, m_n) = \frac{n!}{(1!)^{m_1} (2!)^{m_2} \dots (n!)^{m_n} m_1! m_2! \dots m_n!}$$

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. Откуда искомое число равно $N(0, 6, 0, 3, 0, 4, 1, 5, 0, 6, 0, 7, \dots, 0) = \frac{17!}{(2!)^6 (5!)^1 6! 1!}$.

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 $\frac{4!}{(2!)^2 2!} = 3$ 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 $\frac{32!}{16! 16!}$. Согласно правилу прямого произведения, общее число вариантов разделить колоду пополам равно $\frac{4!}{(2!)^2 2!} \cdot \frac{32!}{16! 16!} = \frac{3 \cdot 32!}{16! 16!}$.

1.11. Полиномиальная формула

Формула

$$(x_1 + x_2 + \dots + x_k)^n = \sum_{n_1 + n_2 + \dots + n_k = n} \frac{n!}{n_1! n_2! \dots n_k!} x_1^{n_1} x_2^{n_2} \dots x_k^{n_k} \quad (1.11.1)$$

называется полиномиальной, где суммирование выполняется по всем решениям уравнения $n_1 + n_2 + \dots + n_k = n$ в целых неотрицательных числах, $n_i > 0$, $i = 1, 2, \dots, k$. Для доказательства выполним умножение

$$\underbrace{(x_1 + x_2 + \dots + x_k)(x_1 + x_2 + \dots + x_k) \dots (x_1 + x_2 + \dots + x_k)}_n = (x_1 + x_2 + \dots + x_k)^n.$$

Чтобы привести подобные в полученном выражении, необходимо подсчитать количество одночленов вида $x_1^{n_1} x_2^{n_2} \dots x_k^{n_k}$ каждого разбиения $n_1 + n_2 + \dots + n_k = n$. Для получения же одночлена $x_1^{n_1} x_2^{n_2} \dots x_k^{n_k}$ необходимо выбрать x_1 в качестве множителя в n_1

скобках при раскрытии выражения $(x_1 + x_2 + \dots + x_k)^n$. 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 одночленов $x_1^{n_1} x_2^{n_2} \dots x_k^{n_k}$ при раскрытии выражения

$$\underbrace{(x_1 + x_2 + \dots + x_k)}_n (x_1 + x_2 + \dots + x_k) \dots (x_1 + x_2 + \dots + x_k)$$

будет равно числу $C_n^{n_1} C_{n-n_1}^{n_2} \dots C_{n-n_1-n_2-\dots-n_{k-1}}^{n_k} = \frac{n!}{n_1! \dots n_k!}$ упорядоченных разбиений.

1.12. Бином Ньютона

Частный вид полиномиальной формулы (1.11.1):

$$(a+b)^n = \sum_{k=0}^n C_n^k a^k b^{n-k} \quad (1.12.1)$$

называется биномом Ньютона.

Рассмотрим несколько задач, в основе решения которых лежит бином Ньютона.

Задача 1. Доказать тождество $\sum_{k=0}^n C_n^k = 2^n$.

Решение. Воспользуемся формулой (1.12.1) биннома Ньютона, в которой положим $a = 1$ и $b = 1$, тогда $(1+1)^n = \sum_{k=0}^n C_n^k \cdot 1^k \cdot 1^{n-k}$.

Задача 2. Доказать тождество $\sum_{k=0}^n C_n^k (m-1)^{n-k} = m^n$.

Решение. Воспользуемся формулой (1.12.1), где положим $a = 1$ и $b = m - 1$.

Задача 3. Доказать тождество $\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} C_n^{2k} = \sum_{k=1}^{\lceil \frac{n}{2} \rceil} C_n^{2k-1} = 2^{n-1}$.

Решение. Воспользуемся формулой (1.12.1), 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

$\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} C_n^{2k} = \sum_{k=1}^{\lceil \frac{n}{2} \rceil} C_n^{2k}$. Так как $\sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} C_n^{2k} = \sum_{k=1}^{\lceil \frac{n}{2} \rceil} C_n^{2k} = \sum_{k=0}^n C_n^k = 2^n$, то каждая из сумм составляет половину числа 2^n .

Задача 4. Доказать тождество $\sum_{k=0}^n k \cdot C_n^k = n2^{n-1}$.

Решение. Воспользуемся формулой (1.12.1), 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 $n(1+x)^{n-1} = \sum_{k=0}^n k \cdot C_n^k x^{k-1}$. Пусть $x = 1$, тогда $n(1+1)^{n-1} = \sum_{k=0}^n k \cdot C_n^k 1^{k-1}$, что доказывает искомое тождество.

1.13. 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, \dots, a_n) — перестановка элементов множества $\{1, 2, \dots, n\}$. Если $i < j$, а $a_i > a_j$, то пара (a_i, a_j) называется инверсией перестановки. Например, перестановка 3142 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 $(1, 2, \dots, n)$.

Таблицей инверсии перестановки (a_1, a_2, \dots, a_n) называется последовательность $d_1 d_2 \dots d_n$, где d_j — число элементов, больших j и расположенных левее. 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

$$0 < d_1 < n - 1, 0 \leq d_2 \leq n - 2, \dots, 0 < d_{n-1} < 1, d_n = 0.$$

М. Холл установил, что таблица инверсий единственным образом определяет соответствующую перестановку. Из любой таблицы инверсий $d_1 d_2 \dots d_n$ можно однозначно восстановить перестановку, которая порождает данную таблицу, путем последовательного определения относительного расположения элементов $n, n-1, \dots, 1$ (в этом порядке). Например, перестановку, соответствующую таблице инверсий $(2, 3, 6, 4, 0, 2, 2, 1, 0) = d_1 d_2 d_3 d_4 d_5 d_6 d_7 d_8 d_9$, 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 что 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 $\{1, 2, \dots, n\}$? Ответ должен быть равен числу всевозможных таблиц инверсий, а их легко пересчитать, так как d_1 можно выбрать n различными способами, d_2 можно независимо от d_1 выбрать $n-1$ способами и т. д., d_n — одним способом. Тогда различных таблиц инверсий $n(n-1)\dots 1 = n!$. Таблицы инверсий пересчитать легко, потому что все d_j независимые, в то время как элементы a_j перестановки должны все быть различными.

1.14. Обратные перестановки

Не следует путать «инверсии» перестановок с обратными перестановками. Пусть a_1, a_2, \dots, a_n — различные шары, индексы которых свяжем с номерами шаров. Тогда исходное расположение шаров однозначно определяется тождественной перестановкой $e = (1, 2, \dots, n)$. Пусть $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ — произвольная перестановка номеров $1, 2, \dots, n$, где π_k — номер шара на k -м месте. Такая перестановка отвечает расположению шаров $a_{\pi_1}, a_{\pi_2}, \dots, a_{\pi_n}$. Вспомним (см. п. 1.5), что перестановка $\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix}$ может быть 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 $л = (\pi_1, \pi_2, \dots, \pi_n)$ и $ст = (\sigma_1, \sigma_2, \dots, \sigma_n)$ исходной последовательности $1, 2, \dots, n$ номеров шаров можно рассматривать как операцию умножения перестановок $р = \pi\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix} \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma_1 & \sigma_2 & \dots & \sigma_n \end{pmatrix}$.

Упорядочим столбцы перестановки $а$ в соответствии с перестановкой $л = (\pi_1, \pi_2, \dots, \pi_n)$, т.е. $ст = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma_1 & \sigma_2 & \dots & \sigma_n \end{pmatrix} = \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \dots & \sigma_{\pi_n} \end{pmatrix}$.

Тогда можно записать, что $р = \pi\sigma = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix} \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \dots & \sigma_{\pi_n} \end{pmatrix} = \begin{pmatrix} 1 & 2 & \dots & n \\ \sigma_{\pi_1} & \sigma_{\pi_2} & \dots & \sigma_{\pi_n} \end{pmatrix} \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_n \\ \rho_1 & \rho_2 & \dots & \rho_n \end{pmatrix}$. Этой перестановке отвечает

расположение шаров $a_{\rho_1}, a_{\rho_2}, \dots, a_{\rho_n}$ где значение $\rho_k = \sigma_{\pi_k}$ — это номер шара на k -м месте.

Обратной к перестановке $я = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix}$ называется перестановка $\pi^{-1} = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1^{-1} & \pi_2^{-1} & \dots & \pi_n^{-1} \end{pmatrix}$ которая получается, если в исходной перестановке поменять местами строки, а затем упорядочить столбцы в возрастающем порядке по верхним элементам, т.е. $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \dots, \pi_n^{-1})$. Ясно, что последовательное изменение порядка шаров согласно перестановкам $л = (\pi_1, \pi_2, \dots, \pi_n)$ и обратной $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \dots, \pi_n^{-1})$ приводит к исходному их расположению, т.е. к тождественной перестановке $е = \begin{pmatrix} 1 & 2 & \dots & n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix} \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_n \\ 1 & 2 & \dots & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & \dots & n \\ 1 & 2 & \dots & n \end{pmatrix}$. 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 $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} \begin{pmatrix} 5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2 \end{pmatrix}$.

Сортированную последовательность элементов перестановки $л = (\pi_1, \pi_2, \dots, \pi_n)$ можно получить, заполнив в цикле вектор e_1, e_2, \dots, e_n :

for $k = 1$ *to* n *do* $e_{\pi_k} = \pi_k$ или $e[\pi_k] = \pi_k$.

Ясно, что результирующие значения e_1, e_2, \dots, e_n будут соответственно $1, 2, \dots, n$. В цикле каждый элемент π_k встает на свое упорядоченное место e_{π_k} (см. п.5.7). Подобным образом выполним заполнение элементов перестановки $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \dots, \pi_n^{-1})$, однако в упорядоченное место π_k^{-1} элемента π_k будем размещать его номер в исходной перестановке $y = (\pi_1, \pi_2, \dots, \pi_n)$:

for $k=1$ **to** n **do** $\pi_k^{-1} = k$ или $\pi^{-1}[\pi_k] = k$.

Результирующий вектор $\pi^{-1} = (\pi_1^{-1}, \pi_2^{-1}, \dots, \pi_n^{-1})$ будет обратной перестановкой к $y = (\pi_1, \pi_2, \dots, \pi_n)$.

Х. А. Роте впервые установил связь между обратными перестановками и инверсиями: *обратная перестановка содержит ровно столько же инверсий, сколько исходная.*

Глава 2

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 ее 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.

2.1. 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.

2.1.1. 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, \dots, s_n является точный список ее членов, расположенных по порядку в смежных ячейках памяти. В языках высокого уровня — это одномерные, двумерные и т. д. массивы данных. Наряду с очевидными преимуществами последовательное представление имеет и некоторые значительные недостатки. Смежное представление становится неудоб-

ным, если требуется изменить последовательность путем включения новых и исключения имеющихся там элементов. Включение между s_i и s_{i+1} нового элемента требует сдвига $s_{i+1}, s_{i+2}, \dots, s_n$ вправо на одну позицию; аналогично исключение s_i требует сдвига тех же элементов на одну позицию влево, как показано в алгоритме 2.1.

Алгоритм 2.1. Включение и исключение элементов при последовательном размещении

{ Включить элемент z на i -е место }

$n = n + 1;$

for $j = n - 1$ to i by -1 do $s_{j+1} = s_j;$

$s_i = z.$

{ Исключить элемент с i -го места }

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 операций включения и удаления $O(n)$.

2.1.2. Характеристические векторы

Важной разновидностью смежного размещения является случай, когда такому представлению подвергается подпоследовательность $s_{k_1}, s_{k_2}, \dots, s_{k_r}$ некоторой основной последовательности s_1, s_2, \dots, 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.

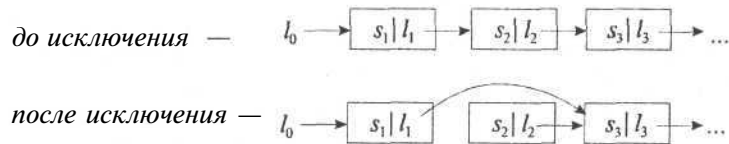
2.1.3. Связанное размещение

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.

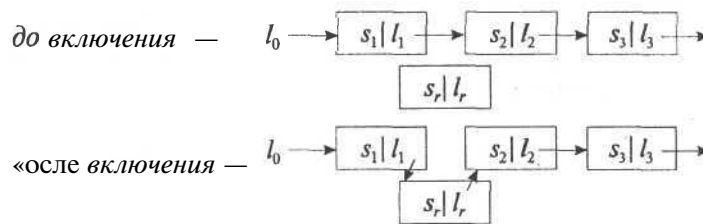
Если требование последовательного размещения элементов опущено, то операции включения и исключения можно выполнить без того, чтобы передвигать элементы. *При любом размещении элементов необходимо сохранять информацию о способе их упорядочения.* При связанном размещении последовательности s_1, s_2, \dots, 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 s_1 последовательности. Последний указатель l_n в списке является пустым или нулевым, это признак конца списка. Графическое представление связанного списка можно изобразить следующим образом:



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 исключения второго элемента достаточно переустановить указатели $NEXT(l_1) = NEXT(l_2)$. Графически это изображается следующим образом:



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 установлено на новый включаемый элемент. Графически включение нового элемента изображается так:



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 2.2 и 2.3 приводятся программы, реализованные на языках Pascal и C, связанного формирования списка элементов последовательности. В программы 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.

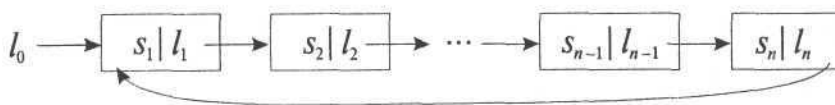


Рис. 2.1. Циклический список

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. 2.2
Дважды
связанный
список



Алгоритм 2.2. Программа на Pascal'e включения и исключения элементов из списка

```

Program Number_List; {Связанный список данных}
uses CRT;

type
  NodePointer=^Node;
  Node= RECORD {Элемент связанного списка}
    s :Integer; {Элемент последовательности}
    next :NodePointer;
           {Указатель на следующий элемент}
  END;
const first :NodePointer=NIL;
           {Указатель начала списка}

{Генерация нового элемента списка}
function InitNode: NodePointer;
var newNode :NodePointer;
begin
  New(newNode); {Выделить память новому элементу}
  newNode^.s:=Random(99)+1;
                {Значение нового элемента}
  newNode^.next:=NIL;
  InitNode:=newNode;
end;

{Включить новый элемент в начало списка}
procedure IncludeNode( newNode: NodePointer );
begin
  newNode^.next:=first;
  first:=newNode;
end;

{Удалить из списка k-й элемент}

```

```
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 { k-й элемент найден}
            if first=current then first:=current^.next
            else previos^.next:=current^.next;
                {Удаление из списка}
            dispose(current);
            break;
        end;
        previos:=current;
        current:=current^.next;
    end;
end;

procedure PrintNodeList; {Печать элементов списка}
var p :NodePointer;
begin
    WriteLn;
    p:=first;
    while p<>nil do begin
        Write (p^.s:3, ' ');
        p:=p^.next
    end;
end;

Var {Main} i,m,n : Integer;
begin {Main}
    ClrScr;
    Randomize;
    n:=17; {Список из n элементов}
    for i:=1 to n do IncludeNode (InitNode) ;
    PrintNodeList;
    WriteLn;
    m:=17; {Удалить из списка m-й элемент}
    DeleteNode (m);
    PrintNodeList;
    ReadKey;
end. {Main}
```

Алгоритм 2.3. Программа на Си включения и исключения элементов из списка

```

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

typedef struct tagNode{ //Элемент связанного списка
    int s; //Элемент последовательности
    tagNode *next; //Указатель на следующий
                //элемент
}Node;
typedef Node *NodePointer;
Node *first=NULL; //Указатель начала списка
NodePointer InitNode( void ){//Генерация нового
                //элемента списка
    NodePointer newNode;
    newNode=new Node; //Выделение памяти
                //новому элементу
    newNode->s=random(99)+1; //Значение нового
                //элемента списка
    newNode->next=NULL;
    return newNode;
}
//Включить новый элемент в начало списка
void IncludeNode( NodePointer newNode ){
    newNode->next=first;
    first=newNode;
}
void DeleteNode( int k ){ //Удалить из списка
                //k-й элемент
    NodePointer previos, current;
    int i;
    i=0;
    current=first;
    while ( current!=NULL ){
        i++;
        if( i==k ){ //k-й элемент найден
            if( first==current ) first=current->next;
            else previos->next=current->next; //Удалить
                //из списка
            delete current;
            break;
        }
    }
}

```

```

    previos=current;
    current=current->next;
}
}
void PrintNodeList ( void ) { //Печать элементов
                                // списка
    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; //Список из n элементов
    for( i=0; i<n; i++ ) IncludeNode ( InitNode ( ) );
    PrintNodeList();
    printf ("\n");
    m=17; //Удалить m-й элемент
    DeleteNode(m);
    PrintNodeList();
    getch();
}

```

Связанное представление предпочтительнее лишь в том случае, если в значительной степени используются операции включения и исключения элементов.

2.2. 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 $m > 0$ поддеревьев T_1, T_2, \dots, T_m .

Корневое дерево на рис. 2.3 содержит 9 узлов, помеченных буквами от a до i . 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.

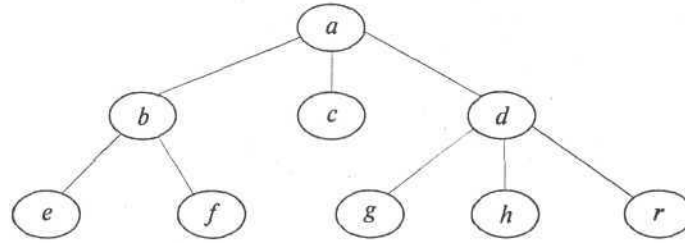


Рис. 2.3. Корневое дерево с тремя поддеревьями

Важной разновидностью корневых деревьев являются *бинарные деревья*. Бинарное дерево T либо пустое, либо состоит из выделенного узла, называемого корнем, и двух бинарных поддеревьев: левого T_1 и правого T_2 .

Лесом называют упорядоченное множество деревьев. Тогда дерево можно определить как непустое множество узлов, такое, что существует один выделенный узел, называемый корнем дерева, а оставшиеся узлы образуют лес с поддеревьями корня.

2.2.1. Представление деревьев на связанной памяти

Почти все машинные представления деревьев основаны на связанных распределениях. Каждый узел состоит из поля данных и некоторых полей для указателей. В следующем примере представления дерева каждый узел имеет по три поля указателей.

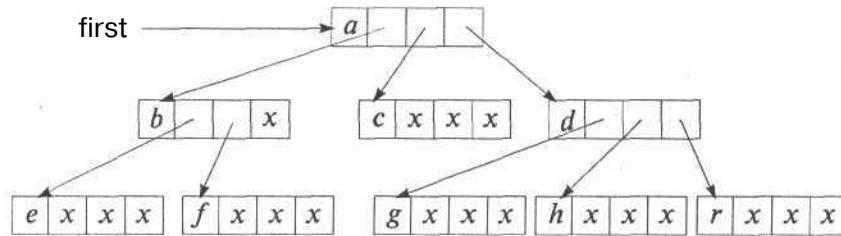


Рис. 2.4. Регулярная связанная структура представления дерева

Произвольное дерево с переменным числом поддеревьев всегда можно представить с помощью односторонних списков с использованием двухкомпонентных звеньев, в которых в первом поле находится либо указатель, либо данные, а во втором — всегда указатель.

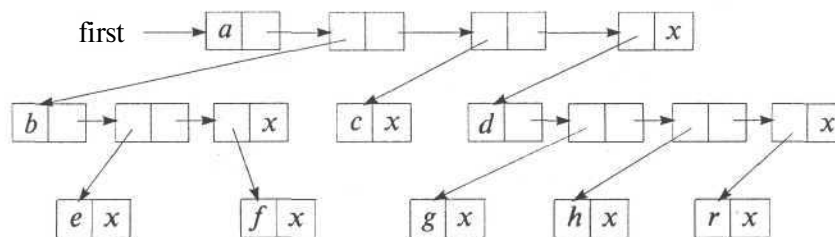


Рис. 2.5. Универсальная связанная структура представления дерева

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).

2.2.2. 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, \dots, a_n . Неявная структура двоичного дерева определяется как на рис. 2.6.

По дереву на рис. 2.6 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 $a[t]$ можно выполнить, разделив t пополам и отбросив дробную часть. Рассмотренная структура применима к любому дереву с постоянным количеством ребер, выходящих из вершин.

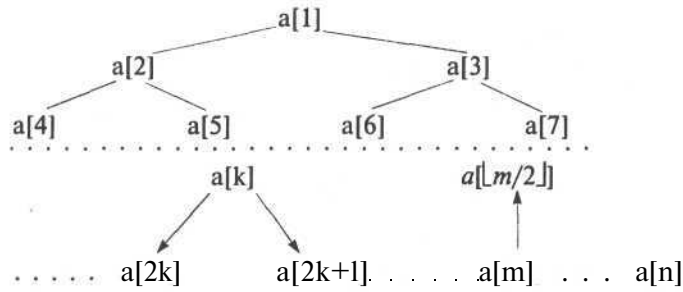
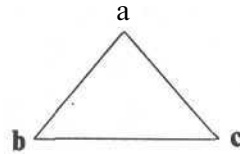


Рис. 2.6. Двоичное дерево на смежной памяти с последовательной нумерацией вершин

Другой способ, основанный на индексной арифметике, применим только для *двоичных деревьев*. Пусть для представления дерева используется одномерный массив a_i, a_{i+1}, \dots, 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 $a_i, a_{i+1}, \dots, a_{m-1}$, а правое поддерево — в массиве $a_{m+1}, a_{m+2}, \dots, a_j$. 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.

Задача. Написать программу поиска всех замкнутых маршрутов длины $n < 15$ по ребрам треугольника abc. Длину ребра принять равной 1. Начальная и конечная точка искомого маршрута — 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.

Пример файла исходных данных:

Выходной файл для данного примера:

```
abcba 1
ababa 2
acaba 3
abaca 4
acaca 5
acbca 6
```

В алгоритме 2.4 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. 2.7.

Алгоритм 2.4. Программа на Pascal'e поиска замкнутых маршрутов по треугольнику

```
Program Tree_Abc; {Движение по треугольнику Abc}
uses CRT,DOS;

const n_max=$fc00; {Максимальная память для дерева}

type Vector=array[1..n_max] of Char;

var f :Text; {Текстовый файл}
    z :Vector; {Двоичное дерево движения по треугольнику}

Procedure CreateTreeAbc( n:Integer );
    {Формирование дерева}
var
    k,level,m,m1,m2 :LongInt;
begin
    z[1]:='a'; {Вершина a}
    level:=1; {Номер уровня}
    m1:=1; {Индекс первой вешины уровня}
    m2:=1; {Индекс последней вешины уровня}
    while level<=n do begin
        for k:=m1 to m2 do begin {Заполнить следующий уровень дерева}
            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;
    end;
```

```

    m1:=2*m1;
    m2:=2*m2+1;
end;
end;
Procedure RouteTreeAbc( n:Integer ); {Формирование
                                         маршрутов}
var
    i,k,m1,m2,r :LongInt;
begin
    r:=0; {Количество маршрутов}
    k:=1;
    for i:=1 to n do k:=2*k;
    m1:=k; {Индекс первой вершины на последнем уровне}
    m2:=2*k-1; {Индекс последней вершины
                на последнем уровне}
    for i:=m1 to m2 do begin{Проход от листьев
                            к вершинам дерева}
        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 {Main} n :Integer; {Длина маршрута}
begin {Main}
    Assign(f,'treeabc.in');
    Reset (f); {Файл открыт для чтения}
    Read(f,n); {Ввод данных}
    Close(f);
    Assign(f,'treeabc.out');
    Rewrite(f); {Файл открыт для записи}
    CreateTreeAbc(n); {Формировать дерево Abc
                      сверху вниз}
    RouteTreeAbc(n); {Формировать маршруты
                     от листьев к вершине}
    Close(f);
end. {Main}

```

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. 2.7 показывают соответствующее их место в массиве данных (смежной памяти).

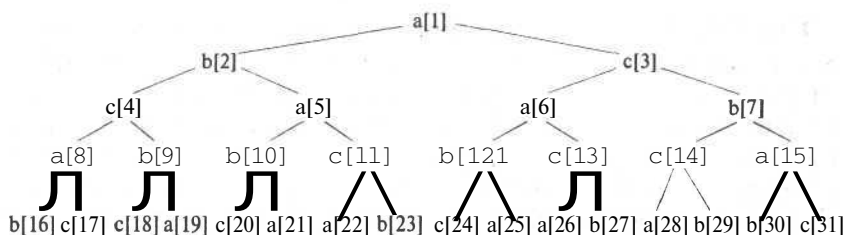


Рис. 2.7. Двоичное дерево маршрутов по треугольнику на смежной памяти

Во второй части алгоритма выполняется формирование искоемых маршрутов (процедура `RouteTreeAbc`), основой для построения которых служит дерево прохода на рис. 2.7. Для формирования всех маршрутов теперь достаточно подняться по нему от листьев с метками **a** вершины треугольника к корню, запоминая пройденные метки. Ясно, что число маршрутов будет равно числу вершин на последнем уровне (количество листьев) с меткой **a**.

2.3. 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. 2.8). Данные методы размещения подробно рассмотрены в п. 2.1 представления последовательностей.

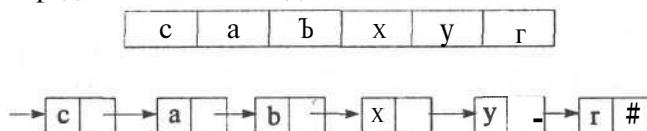


Рис. 2.8. Смежное и связанное представление множества в памяти. Как и для последовательностей, наилучший метод представления множеств существенно зависит от операций, которые мы собираемся выполнять над ними. Типичные операции над множествами: выяснить, имеется ли конкретный элемент в данном множестве; 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 он устанавливается равным 0 (рис. 2.9).

Представление в виде 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 объединения и пересечения 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.

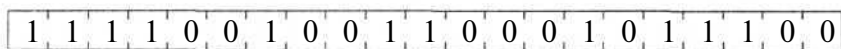


Рис. 2.9. Представление множества характеристическим вектором

Глава 3

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.

3.1. Производящие функции

Пусть a_0, a_1, a_2, \dots — произвольная последовательность. Сопоставим последовательности функцию действительного или комплексного переменного:

$$A(x) = \sum_{k=0}^{\infty} a_k x^k. \quad (3.1.1)$$

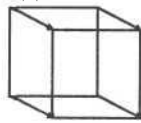
Функция $A(x)$ называется производящей функцией последовательности a_0, a_1, a_2, \dots . Как правило, поиск функции $A(x)$ по формуле (3.1.1) прямыми методами является сложной задачей. Однако заметим, что последовательность $\{a_k\}$ может быть восстановлена по $A(x)$. Выражение (3.1.1) является разложением $A(x)$ в ряд Тейлора в окрестности точки $x = 0$. Воспользуемся этим замечанием и приведем некоторые наиболее распространенные производящие функции и соответствующие им последовательности.

Производящие функции $A(x)$	Последовательности $\{a_k\}$
$\frac{1}{1-x} = \sum_{k=0}^{\infty} 1 \cdot x^k$	$a_k = 1, k \geq 0$ (3.1.2)
$\frac{1}{(1-x)^2} = \sum_{k=0}^{\infty} (k+1)x^k$	$a_k = k+1, k \geq 0$ (3.1.3)
$\ln \frac{1}{1-x} = \sum_{k=1}^{\infty} \frac{1}{k} x^k$	$a_0 = 0, a_k = \frac{1}{k}, k \geq 1$ (3.1.4)
$\ln(1-x) = -\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} x^k$	$a_0 = 0, a_k = \frac{(-1)^{k+1}}{k}, k \geq 1$ (3.1.5)
$(1+x)^r = \sum_{k=0}^{\infty} \binom{r}{k} x^k$	$a_k = \binom{r}{k}, k \geq 0, r - \text{любое}$ (3.1.6)
$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$	$a_k = \frac{1}{k!}, k \geq 0$ (3.1.7)
$e^{rx} = \sum_{k=0}^{\infty} \frac{r^k}{k!} x^k$	$a_k = \frac{r^k}{k!}, k \geq 0, r - \text{любое}$ (3.1.8)
$(1-x)^r = \sum_{k=0}^{\infty} \binom{r+k-1}{k} x^k$	$a_k = \binom{r+k-1}{k}, k \geq 0, r - \text{любое}$ (3.1.9)

Задача. Найти число k -мерных граней в n -мерном кубе.

Решение. Обозначим через a_k число k -мерных граней в n -мерном кубе, где $0 < k < n$. Тогда производящую функцию последовательности $\{a_k\}$ можно записать как $A_n(x) = \sum_{k=0}^n a_k x^k$. 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



$$A_{n+1}(x) = 2A_n(x) + x \cdot A_n(x), \text{ где } A_0(x) = 1.$$

Отсюда

$$A_{n+1}(x) = (2+x)A_n(x) = (2+x)^{n+1}.$$

Воспользуемся разложением биннома Ньютона:

$$A_n(x) = (2+x)^n = \sum_{k=0}^n C_n^k x^k 2^{n-k} = \sum_{k=0}^n a_k x^k.$$

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, $A_3(x) = 2^{3-0} C_3^0 x^0 + 2^{3-1} C_3^1 x^1 + 2^{3-2} C_3^2 x^2 + 2^{3-3} C_3^3 x^3 = 8 + 12x + 6x^2 + x^3$.

Простейшие производящие функции (3.1.2)–(3.1.9) 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 $A(x)$, $B(x)$, $C(x)$.

3.1.1. Линейные операции

Если α и β константы, то последовательность $c_k = \alpha \cdot a_k + \beta \cdot b_k$ имеет производящую функцию $C(x) = \alpha \cdot A(x) + \beta \cdot B(x)$.

Например, последовательность $\{1\}$ соответствует производящей функции $\frac{1}{1-x}$, а последовательность $\left\{\frac{1}{k!}\right\}$ соответствует производящей функции e^x , тогда последовательность $\left\{100 + \frac{1}{k!}\right\}$ соответствует производящей функции $\frac{100}{1-x} + 5e^x$.

3.1.2. Сдвиг начала вправо

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = 0 \text{ для } k = 0, 1, \dots, i-1 \text{ и}$$

$$b_k = a_{k-i} \text{ для } k = i, i+1, \dots, \text{ тогда } B(x) = x^i A(x).$$

Действительно

$$B(x) = \sum_{k=0}^{\infty} b_k x^k = \sum_{k=i}^{\infty} a_{k-i} x^k = x^i \sum_{k=i}^{\infty} a_{k-i} x^{k-i} = x^i \sum_{k=0}^{\infty} a_k x^k = x^i A(x).$$

3.1.3. Сдвиг начала влево

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом: $b_k = a_{k+i}$, $k = 0, 1, \dots$, тогда

$$B(x) = \left[A(x) - \sum_{k=0}^{i-1} a_k x^k \right] x^{-i}.$$

Действительно

$$\begin{aligned} B(x) &= \sum_{k=0}^{\infty} b_k x^k = \sum_{k=0}^{\infty} a_{k+i} x^k = x^{-i} \sum_{k=0}^{\infty} a_{k+i} x^{k+i} = x^{-i} \sum_{k=i}^{\infty} a_k x^k = \\ &= \left[\sum_{k=0}^{\infty} a_k x^k - \sum_{k=0}^{i-1} a_k x^k \right] x^{-i} = \left[A(x) - \sum_{k=0}^{i-1} a_k x^k \right] x^{-i}. \end{aligned}$$

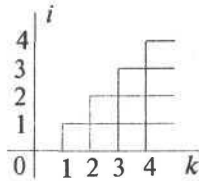
3.1.4. Частичные суммы

Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = \sum_{i=0}^k a_i, \quad k = 0, 1, \dots, \text{тогда } B(x) = \frac{A(x)}{1-x}.$$

Действительно

$$B(x) = \sum_{k=0}^{\infty} b_k x^k = \sum_{k=0}^{\infty} \left(\sum_{i=0}^k a_i \right) x^k.$$



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

$$B(x) = \sum_{k=0}^{\infty} \sum_{i=0}^k a_i x^k = \sum_{i=0}^{\infty} a_i \left(\sum_{k=i}^{\infty} x^k \right) = \left(\sum_{i=0}^{\infty} a_i x^i \right) \left(\sum_{k=0}^{\infty} x^k \right) = A(x) \frac{1}{1-x}.$$

3.1.5. Дополнительные частичные суммы

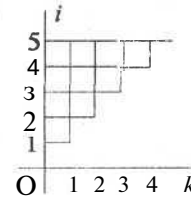
Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = \sum_{i=k}^{\infty} a_i, \quad k = 0, 1, \dots, \text{тогда } B(x) = \frac{A(1) - x \cdot A(x)}{1-x}.$$

Действительно

$$B(x) = \sum_{k=0}^{\infty} b_k x^k = \sum_{k=0}^{\infty} \left(\sum_{i=k}^{\infty} a_i \right) x^k.$$

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



$$\begin{aligned} B(x) &= \sum_{i=0}^{\infty} \sum_{k=0}^i a_i x^k = \sum_{i=0}^{\infty} a_i \left(\sum_{k=0}^i x^k \right) = \sum_{i=0}^{\infty} a_i \frac{1-x^{i+1}}{1-x} = \\ &= \frac{1}{1-x} \left(\sum_{i=0}^{\infty} a_i - x \cdot \sum_{i=0}^{\infty} a_i x^i \right) = \frac{A(1) - x \cdot A(x)}{1-x}. \end{aligned}$$

3.1.6. Изменение масштаба

1. Пусть последовательность $\{b_k\}$ определяется через последовательность $\{a_k\}$ следующим образом:

$$b_k = k \cdot a_k, \text{ тогда } B(x) = x \cdot A'(x).$$

Действительно

$$A(x) = \sum_{k=0}^{\infty} a_k x^k \quad \text{и} \quad A'(x) = \sum_{k=0}^{\infty} k a_k x^{k-1}$$

или

$$x \cdot A'(x) = \sum_{k=0}^{\infty} (k a_k) x^k = B(x).$$

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 $A(x) = \sum_{k=0}^{\infty} a_k x^k$, то

$$\int_0^x A(x) dx = \sum_{k=0}^{\infty} \int_0^x a_k x^k dx = \sum_{k=0}^{\infty} \frac{a_k}{k+1} x^{k+1} = x \sum_{k=0}^{\infty} b_k x^k = x \cdot B(x).$$

3.1.7. Свертка

Последовательность

$$c_k = \sum_{i=0}^k a_i b_{k-i}, k = 0, 1, \dots, \text{ тогда } \text{ОД} = A(x) \cdot B(x).$$

Действительно, $A(x) = \sum_{k=0}^{\infty} a_k x^k$ и $B(x) = \sum_{k=0}^{\infty} b_k x^k$, тогда

$$\begin{aligned} C(x) &= \sum_{k=0}^{\infty} c_k x^k = \sum_{k=0}^{\infty} \left(\sum_{i=0}^k a_i b_{k-i} \right) x^k = \sum_{i=0}^{\infty} \sum_{k=i}^{\infty} a_i b_{k-i} x^k = \\ &= \sum_{i=0}^{\infty} a_i x^i \sum_{k=i}^{\infty} b_{k-i} x^{k-i} = \left(\sum_{i=0}^{\infty} a_i x^i \right) \cdot \left(\sum_{k=0}^{\infty} b_k x^k \right) = A(x) \cdot B(x). \end{aligned}$$

Далее обсудим наиболее общие приемы использования производящих функций на примере решения следующих задач.

Задача. 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

$$\begin{aligned} \binom{-r}{k} &= \frac{(-r)(-r-1)\dots(-r-k+1)}{k!} = \\ &= (-1)^k \cdot \frac{r(r+1)\dots(r+k-1)}{k!} = (-1)^k \binom{r+k-1}{k}. \end{aligned}$$

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 $r = -\frac{1}{2}$.

$$\begin{aligned} \sqrt{1-x} &= (1-x)^{1/2} = \sum_{k=0}^{\infty} \binom{1/2}{k} (-x)^k = \\ &= 1 + \sum_{k=1}^{\infty} \frac{\frac{1}{2}(\frac{1}{2}-1)\dots(\frac{1}{2}-k+1)}{k!} (-x)^k = 1 + \sum_{k=1}^{\infty} \frac{1(-1)(-3)\dots(-2k+3)}{2^k k!} (-x)^k = \end{aligned}$$

$$\begin{aligned}
&= 1 - \sum_{k=1}^{\infty} \binom{2k-1}{k} \frac{1 \cdot 3 \cdot 5 \cdot 7 \dots (2k-3)}{2^k k!} x^k = \sum_{k=1}^{\infty} \frac{1 \cdot 3 \cdot 5 \dots (2k-3)}{2^k k!} x^k = \\
&= 1 - \sum_{k=1}^{\infty} \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \dots (2k-3)(2k-2)}{2^k k! \cdot 2 \cdot 4 \cdot 6 \dots (2k-2)} x^k = \\
&= 1 - \sum_{k=1}^{\infty} \frac{1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \dots (2k-3)(2k-2)}{2^k k! 2^{k-1} (k-1)!} x^k = \\
&= 1 - \sum_{k=1}^{\infty} \frac{2}{4^k k} \frac{(2k-2)!}{(k-1)! (k-1)!} x^k = 1 - \sum_{k=1}^{\infty} \frac{2}{4^k k} \binom{2k-2}{k-1} x^k.
\end{aligned}$$

Таким образом,

$$\sqrt{1-x} = (1-x)^{1/2} = 1 - \sum_{k=1}^{\infty} \frac{2}{4^k k} \binom{2k-2}{k-1} x^k.$$

Рассмотрим выражение $\frac{1}{(1-x)^r} = (1-x)^{-r}$ при $r = 1$.

$$\frac{1}{1-x} = (1-x)^{-1} = \sum_{k=0}^{\infty} \binom{-1}{k} (-x)^k,$$

где

$$\binom{-1}{k} = \frac{(-1)(-1-1)\dots(-1-k+1)}{k!} = (-1)^k \frac{1 \cdot 2 \cdot \dots \cdot k}{k!} = (-1)^k.$$

Следовательно, $\frac{1}{1-x} = (1-x)^{-1} = \sum_{k=0}^{\infty} (-1)^k (-x)^k = \sum_{k=0}^{\infty} x^k$.

Задача. Сколькими способами можно разбить выпуклый $(n+3)$ -угольник ($n > 0$) 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 $1, 2, \dots, n+1$. 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 $k = 1, 2, \dots, n + 1$, получим (согласно правилам прямого произведения и суммы) искомое число разбиений исходного $(n + 3)$ -угольника на треугольники:

$$u_{n+3} = \llcorner 2 \cdot u_{n+2} + u_3 \cdot u_{n+1} + \llcorner 4 \cdot u_{n-0} + \dots + u_{n+1} \cdot u_3 + u_{n+2} \cdot u_2,$$

где $n > 0$ и положили $u_2 = 1$.

Получили нелинейное рекуррентное соотношение для последовательности $\{u_{n+2}\}, n > 0$, для поиска которой удобно ввести новую последовательность $\{v_n\}, n > 0$, такую, что $v_n = u_{n+2}, n \geq 0$.

Тогда рекуррентное соотношение переписывается в виде

$$v_{n+1} = n_0 \cdot v_{n+0} + v_1 \cdot v_{n-1} + v_2 \cdot v_{n-2} + \dots + v_{n-1} \cdot v_1 + v_{n-0} \cdot v_0, \text{ ИЛИ}$$

$$v_{n+1} = \sum_{k=0}^n v_k v_{n-k}.$$

Заметим, что правая часть является сверткой двух одинаковых последовательностей $\{v_n\}$ и $\{v_n\}$ (см. п.3.1.7 операций с производящими функциями). Ввиду этого, составим производящую функцию правой части

$$\sum_{n=0}^{\infty} v_{n+1} x^n = \sum_{n=0}^{\infty} \left(\sum_{k=0}^n v_k v_{n-k} \right) x^n.$$

Пусть $V(x)$ — производящая функция последовательности $\{v_n\}, n > 0$, тогда последнее соотношение запишем как

$$\frac{1}{x}(V(x) - v_0) = V(x)V(x) \text{ или } x \cdot V^2(x) - V(x) + 1 = 0.$$

Отсюда $V(x) = \frac{1}{2x}(1 \pm \sqrt{1-4x})$.

Ранее рассмотренное разложение обобщенного бинома

$$\sqrt{1-x} = 1 - \sum_{k=1}^{\infty} \frac{2 \cdot C_{2k-2}^{k-1}}{4^k k} x^k$$

запишем для случая

$$\sqrt{1-4x} = 1 - \sum_{k=1}^{\infty} \frac{2}{k} C_{2k-2}^{k-1} x^k.$$

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

$$V(x) = \frac{1}{2x}(1 - \sqrt{1-4x}) = \sum_{k=1}^{\infty} \frac{1}{k} C_{2k-2}^{k-1} x^{k-1} = \sum_{k=0}^{\infty} \frac{1}{k+1} C_{2k}^k x^k.$$

Отсюда $u_{n+2} = v_n = \frac{C_{2n}^n}{n+1}$, $n > 0$.

Ответ: число способов разбить выпуклый $(n+2)$ -угольник на треугольники непересекающимися диагоналями равно $\frac{C_{2n}^n}{n+1}$, $n > 0$.

Задача. Найти сумму $1^2 + 2^2 + \dots + k^2 = \sum_{i=1}^k i^2$.

Определим четыре последовательности и их производящие функции:

$$a_k = 1, b_k = ka_k, c_k = kb_k, d_k = \sum_{i=0}^k i^2 \text{ и } A(x), B(x), C(x), D(x).$$

Для решения задачи необходимо найти d_k . С этой целью определим $D(x)$, что позволит нам установить значения d_k , $D(x) = \sum_{k=0}^{\infty} d_k x^k$. Последовательности b_k и a_k связаны «изменением масштаба», значит $B(x) = x \cdot A'(x)$. Последовательности c_k и b_k также связаны «изменением масштаба», следовательно,

$$C(x) = x \cdot B'(x) = x(x \cdot A'(x))' = x \cdot A'(x) + x^2 A''(x).$$

Последовательности d_k и c_k связаны «частичной суммой», тогда

$$D(x) = \frac{C(x)}{1-x} = \frac{x \cdot A'(x) + x^2 A''(x)}{1-x},$$

$$A(x) = \sum_{k=0}^{\infty} 1 \cdot x^k = \frac{1}{1-x}, \quad A'(x) = \frac{1}{(1-x)^2}, \quad A''(x) = \frac{2}{(1-x)^3}.$$

Окончательно $D(x) = \frac{1}{(1-x)^4}$.

Для получения коэффициентов d_k воспользуемся разложением (3.1.9):

$$\frac{1}{(1-x)^4} = \sum_{k=0}^{\infty} \binom{-4}{k} (-x)^k = \sum_{k=0}^{\infty} C_{k+3}^3 \cdot x^k.$$

Теперь можно записать, что

$$\begin{aligned} D(x) &= (x+x^2) \sum_{k=0}^{\infty} C_{k+3}^3 x^k = \sum_{k=0}^{\infty} C_{k+3}^3 x^{k+1} = \\ &= C_3^3 x + \sum_{k=0}^{\infty} (C_{k+4}^3 + C_{k+3}^3) x^{k+2} = C_3^3 x + \sum_{k=2}^{\infty} (C_{k+2}^3 + C_{k+1}^3) x^k = \sum_{k=0}^{\infty} d_k x^k. \end{aligned}$$

Сравнивая коэффициенты при одинаковых степенях x , получим

$$d_0 = 0, \quad d_1 = C_3^3, \quad d_k = C_{k+1}^3 + C_{k+2}^3, \quad k = 2, 3, \dots$$

Таким образом,

$$1^2 + 2^2 + \dots + k^2 = \sum_{i=0}^k i^2 = C_{k+1}^3 + C_{k+2}^3 - \frac{(k+1)(k+1)k}{6}.$$

Задача. Показать, что $\sum_{i=0}^k C_{r+i}^i = C_{r+k+1}^k$ или

$$C_{r+0}^0 + C_{r+1}^1 + \dots + C_{r+k}^k = C_{r+k+1}^k.$$

Заметим, что $\frac{1}{(1+x)^{r+1}} = \sum_{k=0}^{\infty} C_{r+k}^k x^k$ является производящей

функцией последовательности $a_k = C_{r+k}^k$. Следовательно, иско-

мая сумма равна $\sum_{i=0}^k C_{r+i}^i = \sum_{i=0}^k a_i$.

Рассмотрим последовательность $b_k = \sum_{i=0}^k a_i$. Так как b_k и a_k свя-

заны частичной суммой, то $B(x) = \frac{A(x)}{1-x} = \frac{1}{(1-x)^{r+2}}$. Разложение

(3.1.9) позволяет записать последнее выражение в следующем виде:

$$B(x) = \frac{1}{(1-x)^{r+2}} = \sum_{k=0}^{\infty} C_{r+k+1}^k x^k, \text{ откуда } b_k = C_{r+k+1}^k = \sum_{i=0}^k a_i = \sum_{i=0}^k C_{r+i}^i.$$

Задача. Пусть X и Y — целочисленные случайные величины и определены их ряды распределений. Характеристической функцией распределения случайной величины (с.в.) X называется функция

$$g_X(s) = \sum_{k=0}^{\infty} P(X=k) s^k \text{ и } g_Y(s) = \sum_{k=0}^{\infty} P(Y=k) s^k.$$

Таким образом, $g_X(s)$ — это производящая функция последовательности чисел $P(X=k)$, где $k=0, 1, \dots$. Будем полагать X и Y независимыми случайными величинами (с.в.). Рассмотрим с.в. $Z = X + Y$. Очевидно, что

$$P(Z=k) = P(X+Y=k) = \sum_{i=0}^k P(X=i)P(Y=k-i).$$

Ввиду свойства свертки для производящих функций характеристическая функция с.в. Z может быть записана таким образом:

$$g_Z(s) = \sum_{k=0}^{\infty} P(Z=k)s^k = g_X(s)g_Y(s).$$

3.2. Линейные рекуррентные соотношения

Рассмотрим последовательность $\{u_n\}$, $n=0, 1, 2, \dots$. Будем говорить, что задано однородное линейное рекуррентное соотношение с постоянными коэффициентами порядка r , если для членов последовательности $\{u_n\}$ выполняется равенство

$$u_{n+r} = c_1 u_{n+r-1} + c_2 u_{n+r-2} + \dots + c_r u_n, \quad (3.2.1)$$

где c_1, c_2, \dots, c_r — постоянные величины. Выражение (3.2.1) позволяет вычислить очередной член последовательности по предыдущим r членам. Ясно, что, задав начальные значения u_0, u_1, \dots, 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 (3.2.1).

Для решения задачи достаточно найти производящую функцию

$$ВД = \sum_{k=0}^{\infty} u_k x^k \quad (3.2.2)$$

последовательности $\{u_n\}$. Введем обозначение для полинома

$$K(x) = 1 - c_1 x - c_2 x^2 - \dots - c_r x^r$$

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} ($n=0, 1, \dots$) в $U(x)K(x)$ согласно уравнению (3.2.1), равны $u_{n+r} - (c_1 u_{n+r-1} + c_2 u_{n+r-2} + \dots + c_r u_n) = 0$.

Характеристическим полиномом соотношения (3.2.1) называется

$$F(x) = x^r - c_1 x^{r-1} - c_2 x^{r-2} - \dots - c_{r-1} x - c_r. \quad (3.2.3)$$

Выполним разложение $F(x)$ на линейные множители

$$F(x) = (x - \alpha_1)^{e_1} (x - \alpha_2)^{e_2} \dots (x - \alpha_r)^{e_r},$$

где $e_1 + e_2 + \dots + e_r = r$.

Сравнивая $K(x)$ и $F(x)$, запишем $K(x) = x^r \prod_{i=1}^r \left(\frac{1}{x} - \alpha_i \right)^{e_i}$. Отсюда

$$\begin{aligned} K(x) &= x^r \left(\frac{1}{x} - \alpha_1 \right)^{e_1} \left(\frac{1}{x} - \alpha_2 \right)^{e_2} \dots \left(\frac{1}{x} - \alpha_r \right)^{e_r} = \\ &= (1 - \alpha_1 x)^{e_1} (1 - \alpha_2 x)^{e_2} \dots (1 - \alpha_r x)^{e_r}, \quad e_1 + e_2 + \dots + e_r = r. \end{aligned}$$

Данное разложение на множители используем для представления

$$U(x) = \frac{C(x)}{K(x)}$$

в виде суммы простых дробей:

$$U(x) = \frac{C(x)}{K(x)} = \sum_{i=1}^r \sum_{n=1}^{e_i} \frac{\beta_{in}}{(1 - \alpha_i x)^n}. \quad (3.2.4)$$

Таким образом, $U(x)$ является суммой функций вида

$$\frac{\beta}{(1 - \alpha x)^n} = \beta \cdot \sum_{k=0}^{\infty} \binom{n+k-1}{k} \alpha^k x^k.$$

Тогда выражение (3.2.4) примет вид

$$U(x) = \frac{C(x)}{K(x)} = \sum_{i=1}^r \sum_{n=1}^{e_i} \beta_{in} \cdot \sum_{k=0}^{\infty} \binom{n+k-1}{k} \alpha_i^k x^k. \quad (3.2.5)$$

Данное разложение является производящей функцией

$U(x) = \sum_{n=0}^{\infty} u_n x^n$ последовательности $\{u_n\}$. Для определения u_n необходимо найти коэффициент при x^n в разложении (3.2.5).

Задача. Найти члены последовательности $\{u_n\}$, удовлетворяющие рекуррентному соотношению $u_{n+2} = 5u_{n+1} - 6u_n$, $u_0 = u_1 = 1$.

Решение. $K(x) = 1 - 5x + 6x^2$, $K(x)U(x) = C(x)$.

Выполним данное умножение: $(1 - 5x + 6x^2)(u_0 + u_1x + u_2x^2 + \dots) = u_0 + (u_1 - 5u_0)x + (u_2 - 5u_1 + 6u_0)x^2 + \dots = \text{НО} + (u_1 - 5u_0)x = 1 - 4x$. Таким образом, $C(x) = 1 - 4x$.

Характеристический полином $F(x) = x^2 - 5x + 6 = (x - 2)(x - 3)$. Отсюда $U(x) = \frac{C(x)}{K(x)} = \frac{1 - 4x}{(1 - 2x)(1 - 3x)} = \frac{A}{1 - 2x} + \frac{B}{1 - 3x}$. Значения величин A и B находим методом неопределенных коэффициентов: $A = 2$, $B = -1$. Наконец, принимая во внимание (3.1.2),

$$U(x) = \frac{2}{1 - 2x} + \frac{-1}{1 - 3x} = 2 \cdot \sum_{k=0}^{\infty} 2^k x^k - \sum_{k=0}^{\infty} 3^k x^k = \sum_{k=0}^{\infty} (2^{k+1} - 3^k) \cdot x^k.$$

С другой стороны, $U(x) = \sum_{k=0}^{\infty} u_k x^k$. Поэтому, сравнивая коэффициенты при одинаковых степенях x^n , заключаем, что $u_n = 2^{n+1} - 3^n$.

3.3. Неоднородные линейные рекуррентные соотношения

Неоднородное линейное рекуррентное соотношение имеет вид

$$u_{n+r} = c_1 u_{n+r-1} + c_2 u_{n+r-2} + \dots + c_r u_n + b_n, \quad (3.3.1)$$

где величина b_n в общем случае является функцией от n . Общее решение соотношения (3.3.1) представляет собой сумму частного решения неоднородного уравнения (3.3.1) (т.е. любого решения, которое ему удовлетворяет) и общего решения соответствующего ему однородного соотношения (3.2.1), которое находится рассмотренным способом. Общих способов определения частного решения нет, однако для специальных значений B существуют стандартные приемы определения u_n . Рассмотрим на примере в некотором роде универсальную процедуру, которая позволяет сразу находить общее решение неоднородного уравнения (3.3.1).

Задача. Найти $\{u_n\}$, если известно, что $u_{n+1} = u_n + (n + 1)$ и $u_0 = 1$.

Умножив левую и правую части рекуррентного соотношения на x^n , получим

$$u_{n+1}x^n = u_nx^n + (n + 1)x^n.$$

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

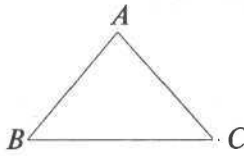
$$\frac{U(x) - u_0}{x} = U(x) + \frac{1}{(1-x)^2}.$$

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:

$$\begin{cases} a_{n+1} = b_n + c_n, \\ b_{n+1} = a_n + b_n, \\ c_{n+1} = a_n + c_n, \\ b_n = c_n. \end{cases}$$

С учетом последнего равенства $b_n = c_n$ ($n = 0, 1, 2, \dots$) 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 $u_n = a_n$:

$$u_{n+2} = u_{n+1} + 2u_n, \quad u_0 = 1, \quad u_1 = 0, \quad \text{где } n = 0, 1, 2, \dots$$

Решение данного соотношения получим согласно изложенной выше теории.

$$K(x) = 1 - x - 2x^2, \quad U(x)K(x) = C(x) = 1 - x.$$

Характеристический полином

$$F(x) = x^2 - x - 2 = (x - 2)(x + 1).$$

$$\text{Отсюда } U(x) = \frac{1-x}{(1-2x)(1+x)} = \frac{1/3}{1-2x} + \frac{2/3}{1+x}.$$

Перепишем $U(x)$ в развернутом виде по степеням x :

$$U(x) = \sum_{n=0}^{\infty} \frac{1}{3} 2^n x^n + \sum_{n=0}^{\infty} \frac{2}{3} (-1)^n x^n = \sum_{n=0}^{\infty} \frac{2^n + (-1)^n 2}{3} x^n.$$

Сравнивая коэффициенты при x^n , заключаем, что число замкнутых маршрутов длины n равно $u_n = \frac{2^n + (-1)^n 2}{3}$.

3.4. Обобщенное правило произведения

Пусть S_1, S_2, \dots, S_n — произвольные множества.

$\Omega = \{\omega_1, \omega_2, \omega_3, \dots\}$ — множество весов, где под ω_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 число либо переменная, которая может принимать любые числовые значения. Назначим каждому элементу $s \in S_i$ ($i = 1, 2, \dots, n$) 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 ω , тогда $A(S_i) = \sum_{\omega \in \Omega} C_{i\omega} \omega$ — сумма весов элементов множества S_i .

Рассмотрим прямое произведение множеств

$$S = S_1 \times S_2 \times \dots \times S_n = \{s = (s_1 s_2 \dots s_n) \mid s_i \in S_i, i = \overline{1, n}\}.$$

Положим вес элемента множества равным

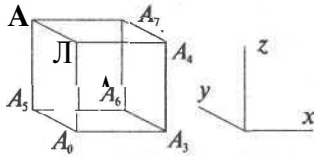
$$\omega(\varepsilon) = \omega(s_1 s_2 \dots s_n) = \omega(s_1) \cdot \omega(s_2) \dots \omega(s_n) = \prod_{i=1}^n \omega(s_i) \in \Omega.$$

Пусть E_ω — количество элементов $\varepsilon \in S$ с весом ω , тогда $A(S) = \sum_{\omega \in \Omega} E_\omega \cdot \omega$ — сумма весов элементов множества.

Теорема. $A(S) = A(S_1)A(S_2)\dots A(S_n)$.

Доказательство.

$$\begin{aligned} A(S_i) &= \sum_{\omega \in \Omega} C_{i\omega} \cdot \omega = \sum_{s \in S_i} \omega(s). \\ A(S) &= \sum_{\omega \in \Omega} E_\omega \cdot \omega = \sum_{s_1 \in S_1} \sum_{s_2 \in S_2} \dots \sum_{s_n \in S_n} \omega(s_1 s_2 \dots s_n) = \\ &= \sum_{s_1 \in S_1} \sum_{s_2 \in S_2} \dots \sum_{s_n \in S_n} \omega(s_1) \cdot \omega(s_2) \dots \omega(s_n) = \\ &= \left(\sum_{s_1 \in S_1} \omega(s_1) \right) \left(\sum_{s_2 \in S_2} \omega(s_2) \right) \dots \left(\sum_{s_n \in S_n} \omega(s_n) \right) = A(S_1) \cdot A(S_2) \dots A(S_n). \end{aligned}$$



Задача. Найти количество замкнутых маршрутов длины $2n$ по ребрам трехмерного куба. Начальное и конечное положение — вершина A_0 куба.

Решение. Исходное положение — вершина A_0 . Каждый шаг движения по кубу — это выбор одного из трех ребер. Пусть 1 обозначает выбор ребра вдоль оси OX , 2 — вдоль оси OY , 3 — вдоль оси OZ . В соответствии с этим введем множества S_1, S_2, \dots, S_{2n} такие, что $S_1 = S_2 = \dots = S_{2n} = \{1, 2, 3\}$. Тогда все маршруты длины $2n$ составят множество $S = S_1 \times S_2 \times \dots \times S_{2n}$. 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 $A(S) = A(S_1)A(S_2)\dots A(S_{2n}) = (x + y + z)^{2n}$, т.к. $A(S_i) = x + y + z$.

После возведения в степень получим, что

$$A(S) = (x + y + z)^{2n} = \sum_i \sum_j \sum_k a_{ijk} \cdot x^i y^j z^k, \quad (3.4.1)$$

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 , т. к. в замкнутом маршруте, сделав шаг вдоль оси, необходимо вернуться по этой оси. Для выделения таких маршрутов положим $x = y = z = 1$ (x, y, z — произвольные). Тогда выражение (3.4.1) примет вид

$$(x + y + z)^{2n} = C_0 + C_2 yz + C_4 xz + C_6 xy, \quad (3.4.2)$$

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 (3.4.2) следует, что

$$(x + y + z)^{2n} = C_0 + C(yz + xz + xy). \quad (3.4.3)$$

Уравнение (3.4.3) содержит два неизвестных C_0 и C при $x^2 = y^2 = z^2 = 1$. Для их определения составим систему уравнений из выражения (3.4.3), полагая $x = y = z = 1$ и $x = y = 1, z = -1$. Это возможно, т. к. выполняется условие $x^2 = y^2 = z^2 = 1$. Получим систему

$$\begin{cases} (1+1+1)^{2n} = C_0 + C(1+1+1) \\ (1+1-1)^{2n} = C_0 + C(-1+1+1) \end{cases} \quad \text{или} \quad \begin{cases} 3^{2n} = C_0 + 3C \\ 1 = C_0 - C \end{cases}.$$

Отсюда $C_0 = \frac{3^{2n} + 3}{4}$ — число искомых маршрутов.

Задача. Найти число решений p, q уравнения

$$2p + 3q = n, \quad \text{где } n, p, q \in \{0, 1, 2, \dots\}.$$

Решение. Введем множества: $S_1 = \{2p \mid p = 0, 1, 2, \dots\}$, $S_2 = \{3q \mid q = 0, 1, 2, \dots\}$ и $S = S_1 \times S_2 = \{(2p, 3q) \mid p, q = 0, 1, 2, \dots\}$.

Назначим веса элементам данных множеств следующим образом:

$$\omega(2p) = x^{2p}, \quad \omega(3q) = x^{3q}, \quad \text{и}$$

$$\omega(\varepsilon) = \omega((2p, 3q)) = x^{2p+3q=n} = x^{2p} \cdot x^{3q} = \omega(2p)\omega(3q).$$

Веса элементов определены таким образом, что выполняется правило обобщенного произведения $A(S) = A(S_1) \cdot A(S_2)$. Легко заметить, что степень веса $x^{2p+3q=n}$ любого элемента $E = (2p, 3q) \in S$ дает одно из решений исходного уравнения $2p + 3q = n$.

Раскроем выражение $A(S) = A(S_1) \cdot A(S_2)$:

$$\begin{aligned} & \text{НО} + u_1x + u_2x^2 + \dots + u_nx^n + \dots = \\ & = (x^{2 \cdot 0} + x^{2 \cdot 1} + x^{2 \cdot 2} + \dots + x^{2 \cdot p} + \dots)(x^{3 \cdot 0} + x^{3 \cdot 1} + x^{3 \cdot 2} + \dots + x^{3 \cdot q} + \dots) = \\ & = (1 + x^2 + x^4 + \dots)(1 + x^3 + x^6 + \dots) = \frac{1}{1-x^2} \cdot \frac{1}{1-x^3}, \end{aligned}$$

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, $A(S) = \frac{1}{1-x} \cdot \frac{1}{1-x^3}$ у Разложим данное выражение на множители:

$$\begin{aligned} A(S) &= \frac{1/4}{1+x} + \frac{1/6}{(1-x)^2} + \frac{7/12 - x/12 + x^2/4}{1-x^3} = \\ &= \frac{1}{12} \cdot \left(\frac{3}{1+x} + \frac{2}{(1-x)^2} + \frac{7-x+3x^2}{1-x^3} \right) = \\ &= \frac{1}{12} \left(3 \sum_{k=0}^{\infty} (-1)^k x^k + 2 \sum_{k=0}^{\infty} (k+1)x^k + (7-x+3x^2) \sum_{k=0}^{\infty} x^{3k} \right) = \sum_{k=0}^{\infty} u_k x^k, \end{aligned}$$

где u_n — число решений уравнения $2p + 3q = n$.

Сравнивая коэффициенты при одинаковых степенях, получим:

$$\begin{aligned} u_{3n} &= \frac{3 \cdot (-1)^{3n} + 2 \cdot (3n+1) + 7}{12} = \frac{2n+3+(-1)^{3n}}{4}, \\ u_{3n+1} &= \frac{3 \cdot (-1)^{3n+1} + 2 \cdot (3n+2) - 1}{12} = \frac{2n+1-(-1)^{3n}}{4}, \\ u_{3n+2} &= \frac{3 \cdot (-1)^{3n+2} + 2 \cdot (3n+3) + 3}{12} = \frac{2n+3+(-1)^{3n}}{4}, \end{aligned}$$

где $y \in \{0, 1, 2, \dots\}$.

3.5. Принцип включения и исключения

Пусть $S = \{s_1, s_2, s_3, \dots\}$ ~ произвольное множество элементов.
 $\Omega = \{\omega_1, \omega_2, \omega_3, \dots\}$ — множество весов, $\Omega(s_k)$ — вес элемента $s_k \in S$.
 $P_1, P_2, P_3, \dots, P_n$ — свойства элементов или индикаторы свойств элементов.

$$P_i(s_k) = \begin{cases} 1, & \text{если элемент } s_k \text{ обладает свойством } P_i, \\ 0, & \text{если элемент } s_k \text{ не обладает свойством } P_i. \end{cases}$$

$W(P_i) = \sum_{s_k \in S} P_i(s_k) \cdot \text{га}(s_k)$ — сумма весов элементов со свойством P_i .

$W(P_{i_1} P_{i_2} \dots P_{i_m})$ — сумма весов элементов множества S , которые обладают каждым из свойств $P_{i_1}, P_{i_2}, \dots, P_{i_m}$.

$W(m) = \sum_{(i_1 i_2 \dots i_m)} W(P_{i_1} P_{i_2} \dots P_{i_m})$, суммирование выполняется по всем сочетаниям $(i_1 i_2 \dots i_m)$ 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 $k \geq m$, тогда его вес $\omega(s)$ в $W(m)$ войдет C_k^m раз.

Так, $W(1) = \sum_{(i_1)} W(P_{i_1}) = W(P_1) + W(P_2) + \dots + W(P_n)$ содержит $C_n^1 =$

$= n$ членов и $W(2) = \sum_{(i_1 i_2)} W(P_{i_1} P_{i_2}) = W(P_1 P_2) + W(P_1 P_3) + \dots + W(P_{n-1} P_n)$

содержит $C_n^2 = n(n+1)/2$ членов.

Распространим определение $W(m)$ на $m = 0$, положив $W(0) = \sum_{s \in S} \omega(s)$ — сумма весов элементов исходного множества S .

Данное определение $W(0)$ выполнено корректно, так как сумма $W(0)$ должна включать элементы, обладающие нуль свойствами и более. Действительно, любой из элементов множества S удовлетворяет этим условиям.

Положим:

$E(m)$ — сумма весов элементов, обладающих ровно m свойствами;

$E(0)$ — сумма весов элементов, которые не имеют ни одного из указанных свойств.

Теорема. Сумма весов элементов, обладающих точно m свойствами из n свойств P_1, P_2, \dots, P_n , равна

$$E(m) = W(m) - C_{m+1}^m W(m+1) + \dots + (-1)^{n-m} C_{m+(n-m)}^m W(m+(n-m))$$

или

$$E(m) = \sum_{i=0}^{n-m} (-1)^i C_{m+i}^m W(m+i). \quad (3.5.1)$$

Доказательство. Для доказательства достаточно показать, что вклад веса $\omega(s)$ произвольного элемента $s \in S$ в правую и левую

части (3.5.1) одинаковый. Пусть $s \in S$ обладает точно / свойствами. Возможны следующие соотношения между / и m :

- 1) $/ < m$, тогда $\omega(s)$ не входит в $E(m)$ и не входит в $W(m+i)$ для $i = 0, 1, \dots, n - m$. Равенство (3.5.1) примет вид $0 = 0$.
- 2) $/ = m$, тогда $\omega(s)$ входит один раз в $E(m)$ и один раз в $W(m)$.
- 3) $/ > m$, тогда $\omega(s)$ не входит в $E(m)$ и левая часть равна 0. Покажем, что в этом случае и правая часть выражения (3.5.1) равна нулю.

Вес $\omega(s)$ в $W(m+i)$ входит C_t^{m+i} раз, $m+i < /$. Правая часть для веса $\omega(s)$ одного элемента s примет вид

$$\begin{aligned} C_m^m \omega(s) C_t^m - C_{m+1}^m \omega(s) C_t^{m+1} + \dots + (-1)^{t-m} C_t^m \omega(s) C_t^t &= \\ = \omega(s) \sum_{i=0}^{t-m} (-1)^i C_{m+i}^m C_t^{m+i}. \end{aligned}$$

Заметим, что

$$C_{m+i}^m C_t^{m+i} = \frac{(m+i)!}{m! i!} \cdot \frac{t!}{(m+i)!(t-(m+i))!} = \frac{/!}{m!(t-m)!} \cdot \frac{(t-m)!}{i!((t-m)-i)!}.$$

Следовательно, $C_{m+i}^m C_t^{m+i} = C_t^m C_{t-m}^i$, а исходная сумма составит

$$\begin{aligned} \omega(s) \sum_{i=0}^{t-m} (-1)^i C_{m+i}^m C_t^{m+i} &= \omega(s) \sum_{i=0}^{t-m} (-1)^i C_t^m C_{t-m}^i = \\ &= \omega(s) C_t^m \sum_{i=0}^{t-m} (-1)^i C_{t-m}^i = \omega(s) C_t^m (1-1)^{t-m} = 0. \end{aligned}$$

И в третьем случае выражение (3.5.1) справедливо. Теорема доказана.

Следствие.

$$E(0) = W(0) - W(1) + W(2) - W(3) + \dots + (-1)^n W(n).$$

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. $D(0)$ — студенты, не знающие языков (не обладают свойствами).

$$\begin{aligned}
 E(0) &= W(0) - W(1) + W(2). \\
 W(0) &= \sum_{s \in S} \omega(s) = 23. \\
 W(1) &= W(P_1) + W(P_2) = 18 + 9 = 27. \\
 W(2) &= W(P_1 P_2) = 6.
 \end{aligned}$$

Тогда $E(0) = 23 - 27 + 6 = 2$.

$E(1)$ — студенты со знанием одного языка (обладают ровно одним свойством).

$$E(1) = C_1^1 W(1) - C_2^1 W(2) = 1 \cdot (18 + 9) - 2 \cdot 6 = 15.$$

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, $i = 1, 2, \dots, m$. $E(r)$ — количество перестановок, обладающих ровно r свойствами, т.е. при перестановке шаров ровно r из них остаются на своих местах. Тогда по формуле включений и исключений запишем:

$$E(r) = \sum_{i=0}^{m-r} (-1)^i C_{r+i}^i W(r+i). \text{ Рассмотрим } W(r) = \sum_{(i_1 i_2 \dots i_r)} W(P_{i_1} P_{i_2} \dots P_{i_r}),$$

где суммирование выполняется по всем сочетаниям $(i_1 i_2 \dots i_r)$ длины r из m свойств, количество сочетаний равно C_m^r .

$W(P_{i_1} P_{i_2} \dots P_{i_r}) = (m-r)!$ — количество перестановок из $m-r$ шаров, так как r шаров должны оставаться на месте. Тогда значение

$$W(r) = C_m^r (m-r)! \text{ Следовательно, и } E(r) = \sum_{i=0}^{m-r} (-1)^i C_{r+i}^r C_m^{r+i} (m-r-i)!$$

$$\text{или } E(r) = \frac{m! \sum_{i=0}^{m-r} (-1)^i}{r! \sum_{i=0}^{m-r} i!} \approx \frac{m! e^{-1}}{r!}.$$

3.6. Ладейные многочлены и многочлены попаданий

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.

3.6.1. 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. 3.1.

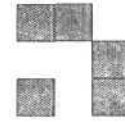


Рис. 3.1

- **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

$$R(x, C) = \sum_{k=0}^{\infty} r_k x^k. \quad (3.6.1)$$



Задача. Найти ладейный многочлен для доски C_1 на рис. 3.2.

Рис. 3.2 **Решение.** Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 1$ и $r_i = 0$, $i > 2$. Тогда $R(x, C_1) = 1 + x$.



Задача. Найти ладейный многочлен доски C_2 на рис. 3.3.

Рис. 3.3 **Решение.** Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 2$, $r_2 = 1$ и $r_i = 0$, $i > 3$. Тогда $R(x, C_2) = 1 + 2x + x^2$.



Задача. Найти ладейный многочлен доски C_3 на рис. 3.4.

Рис. 3.4 **Решение.** Непосредственным подсчетом можно установить, что $r_0 = 1$, $r_1 = 3$, $r_2 = 1$ и $r_i = 0$, $i > 3$. 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. 3.5.

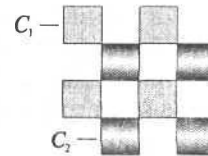


Рис. 3.5

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 π_1 — расстановка ладей на доске C_1 , π_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_{π_1} и S_{π_2} — множество расстановок ладей на доске, соответственно C_1 и C_2 . Тогда прямое произведение $S_{\pi} = S_{\pi_1} \times S_{\pi_2}$ — множество расстановок ладей на доске C .

Обозначим веса расстановок: $\omega(\pi_1) = x^{k_1}$, где k_1 — число ладей в перестановке π_1 ; $\omega(\pi_2) = x^{k_2}$, где k_2 — число ладей в перестановке π_2 . Тогда вес перестановки π равен $\omega(\pi) = x^{k_1+k_2} = x^{k_1} \cdot x^{k_2} = \omega(\pi_1)\omega(\pi_2)$.

В соответствии с тем, как введены веса перестановок, ладейные многочлены можно записать как сумму весов элементов множеств:

$$R(x, C_1) = \sum_{\pi_1 \in S_{\pi_1}} \omega(\pi_1), \quad R(x, C_2) = \sum_{\pi_2 \in S_{\pi_2}} \omega(\pi_2) \quad \text{и} \quad R(x, C) = \sum_{\pi \in S_{\pi}} \omega(\pi).$$

Многочлены $R(x, C_1)$, $R(x, C_2)$ и $R(x, C)$ удовлетворяют всем условиям правила обобщенного произведения (см. п. 3.4), в соответствии с которым $R(x, C) = R(x, C_1)R(x, C_2)$.

Задача. Найти ладейный многочлен доски C на рис. 3.6.

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 $R(x, C) = [R(x, C_1)]^n = (1 + x)^n$.

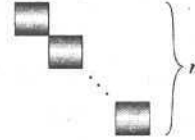


РИС. 3.6

- *Свойство 2. Правило суммы.* Пусть C — доска запрещенных позиций. Введем обозначения:

$C_{i_{\alpha}}$ — доска с ладьей в клетке α (i — index); $C_{e_{\alpha}}$ — доска с удаленной клеткой α (e — erase). Тогда

$$R(x, C) = xR(x, C_{i_{\alpha}}) + R(x, C_{e_{\alpha}}) \quad \text{или} \quad R(xi_{\alpha} + e_{\alpha}) = xR(i_{\alpha}) + R(e_{\alpha}).$$

Для доказательства данного свойства вновь рассмотрим ладейный многочлен как сумму весов перестановок. Все расстановки множества разобьем на два подмножества $S_{\pi_{i_{\alpha}}}$ и $S_{\pi_{e_{\alpha}}}$, $S_{\pi_{i_{\alpha}}}$ — перестановки с ладьей на клетке α ; $S_{\pi_{e_{\alpha}}}$ — перестановки, которые не занимают клетку α .

Тогда

$$\begin{aligned}
 R(x, C) &= \sum_{\pi \in S_x} \omega(\pi) = \sum_{\pi_{i_\alpha} \in S_{\pi_{i_\alpha}}} x \omega(\pi_{i_\alpha}) + \sum_{\pi_{e_\alpha} \in S_{\pi_{e_\alpha}}} \omega(\pi_{e_\alpha}) = \\
 &= x \left(\sum_{\pi_{i_\alpha} \in S_{\pi_{i_\alpha}}} \omega(\pi_{i_\alpha}) \right) + \sum_{\pi_{e_\alpha} \in S_{\pi_{e_\alpha}}} \omega(\pi_{e_\alpha}) = x \cdot R(x, C_{i_\alpha}) + R(x, C_{e_\alpha}).
 \end{aligned}$$

Множитель x перед скобкой — это вес ладьи в перестановке π_{i_α} , которая поставлена на выделенную клетку a .

Задача. Найти $R(x, C)$ для доски на рис. 3.7.

Решение. Воспользуемся правилом суммы, по которому

$$\begin{aligned}
 R(x, C) &= x \cdot R(\square) + R(\square) = x(1 + x + x^2 + x^3) + 1 = \\
 &= 1 + 4x + 4x^2 + x^3.
 \end{aligned}$$

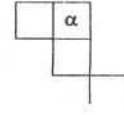


Рис. 3.7

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. 3.8).

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 $r_k = C_m^k C_n^k \cdot k!$ — число способов расставить k ладей на исходной прямоугольной доске. Тогда $R(x, C) = \sum_{k=0}^{\min(m,n)} C_m^k C_n^k k! x^k$.

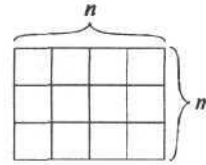


Рис. 3.8

Задача. Найти $R(x, C)$ для прямоугольной доски C размером 2×3 .

Решение.

$$\begin{aligned}
 R(x, C) &= \sum_{k=0}^2 C_2^k C_3^k k! x^k = C_2^0 C_3^0 0! x^0 + C_2^1 C_3^1 1! x^1 + C_2^2 C_3^2 2! x^2 = \\
 &= 1 + 6x + 6x^2.
 \end{aligned}$$

Коэффициент при x^2 показывает, что число способов поставить две ладьи на такую доску равно 6.

3.6.2. Многочлены попаданий

- Определение.** Пусть дана квадратная доска размером $l \times l$ с запрещенными позициями (рис. 3.9). Обозначим через $E_n(k)$ количество перестановок на квадратной доске l ладей, k из которых занимают запрещенные позиции. Многочленом попадания называется производящая функция $E(t) = \sum_{k=0}^n E_n(k)t^k$ последовательности $\{E_n(k)\}$.

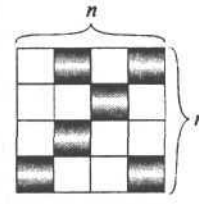


Рис. 3.9

Установим связь между многочленом попаданий и ладейным многочленом. Введем обозначения:

$S_n = \{\pi_1, \pi_2, \dots, \pi_n\}$ — множество всех перестановок l ладей на доске $l \times n$;

$\Pi = \{\omega_1, \omega_2, \dots\}$ — множество весов;

$\omega: S_n \rightarrow \Omega$ — весовая функция, $\gamma \in \Omega$ — вес перестановки $\gamma \in S_n$;

$\{1, 2, \dots, m\}$ — номера запрещенных клеток на доске $l \times l$;

P_1, P_2, \dots, P_m — свойства перестановок; $\gamma \in S_n$ обладает свойством P_j , если ее ладья занимает запрещенную позицию j .

$P_j(\gamma) = \begin{cases} 1, & \text{если } \gamma \text{ обладает свойством } P_j, \\ 0, & \text{в противном случае.} \end{cases}$

$W(P_j) = \sum_{\pi \in S_n} P_j(\pi)\omega(\pi)$ — сумма весов перестановок со свойством P_j ;

$W(P_{j_1} P_{j_2} \dots P_{j_k}) = \sum_{\pi \in S_n} P_{j_1}(\pi)P_{j_2}(\pi) \dots P_{j_k}(\pi)\omega(\pi)$ — сумма весов перестановок со свойствами $P_{j_1} P_{j_2} \dots P_{j_k}$;

$W(k) = \sum_{(j_1, j_2, \dots, j_k)} W(P_{j_1} P_{j_2} \dots P_{j_k})$, суммирование выполняется по всем сочетаниям $(j_1 j_2 \dots j_k)$ длины k из m свойств, число сочетаний C_m^k .

Принцип включения и исключения позволяет определить

$$E_n(k) = C_k^k W(k) - C_{k+1}^k W(k+1) + \dots + (-1)^{m-k} C_{k+(m-k)}^k W(k+(m-k)).$$

Так как нас интересует лишь количество перестановок, то будем полагать $\omega(\pi) = 1$. Нетрудно установить, что $W(k) = r_k(n-k)!$, где r_k — количество способов расставить k ладей на запрещенных позициях доски $l \times l$. Тогда

$$E_n(k) = C_k^k r_k(n-k)! - C_{k+1}^k r_{k+1}(n-k+1)! + \dots + (-1)^{m-k} C_m^k r_m(n-m)!.$$

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 $E_n(k) = C_k^k r_k (n-k)! - C_{k+1}^k r_{k+1} (n-k+1)! + \dots + (-1)^{n-k} C_n^k r_n (n-n)!$ или

$$E_n(k) = \sum_{i=0}^{n-k} (-1)^i C_{k+i}^i r_{k+i} (n-(k+i))!. \text{ Найденное выражение позво-}$$

ляет записать многочлен попаданий 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. 3.10. Замена переменных суммирования $k' = k + i$ и $i' = i$ в последнем выражении позволяет упростить многочлен попаданий и приводит его к виду $E(t) = \sum_{k=0}^n \sum_{i=0}^k (-1)^i C_k^i r_k (n-k)! t^k$, где суммирование уже выполняется по узлам сетки на рис. 3.11. Таким образом,

$$E(t) = \sum_{k=0}^n r_k \left(\sum_{i=0}^k (-1)^i C_k^i \cdot t^{k-i} \right) (n-k)! \text{ или}$$

$$E(t) = \sum_{k=0}^n r_k (t-1)^k (n-k)!.$$

Для более компактной записи последнего выражения введем оператор E^{-1} , действие которого распространим на функции целочисленного аргумента: $\varepsilon^{-1}(f(n)) = f(n-1)$ и $\varepsilon^{-k}(f(n)) = f(n-k)$. Например, $\varepsilon^{-1}(n!) = (n-1)!$ 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

$$\begin{aligned} E(t) &= \sum_{k=0}^n r_k (t-1)^k (n-k)! = \sum_{k=0}^n r_k [(t-1)\varepsilon^{-1}]^k n! = \\ &= \left(\sum_{k=0}^n r_k [(t-1)\varepsilon^{-1}]^k \right) n! = \sum_{k=0}^n r_k x^k = R(x) n!, \end{aligned}$$

где $x = (t-1)\varepsilon^{-1}$. Таким образом, $E(t) = R((t-1)\varepsilon^{-1}) \cdot n!$.

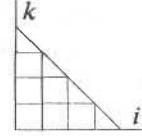


Рис. 3.10

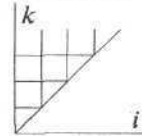


Рис. 3.11

Задача. Найти многочлен попаданий для доски 3×3 с запрещенными позициями на рис. 3.12. Запрещенные позиции отмечены темным цветом.

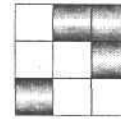


Рис. 3.12

Решение. Найдем ладейный многочлен доски запрещенных позиций, которая состоит из двух независимых досок. Тогда

$$R(x) = R(\text{доска 1}) \cdot R(\text{доска 2}) = (1+x)(1+3x+x^2) = 1+4x+4x^2+x^3.$$

Значит

$$\begin{aligned} E(t) &= R((t-1)\varepsilon^{-1})n! = (1+4(t-1)\varepsilon^{-1}+4[(t-1)\varepsilon^{-1}]^2+[(t-1)\varepsilon^{-1}]^3)3! = \\ &= 1 \cdot 3! + 4(t-1)\varepsilon^{-1}3! + 4[(t-1)\varepsilon^{-1}]^2 3! + [(t-1)\varepsilon^{-1}]^3 3! = \\ &= 3! + 4(t-1)2! + 4(t-1)^2 1! + (t-1)^3 0!. \end{aligned}$$

$$\text{Итак, } E(t) = 1 + 3t + t^2 + t^3.$$

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).

Глава 4

Генерация комбинаторных объектов



О 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

$$\sum f(x_1, x_2, \dots, x_n),$$

где суммирование выполняется по всем последовательностям x_1, x_2, \dots, x_n , удовлетворяющим некоторым ограничениям.

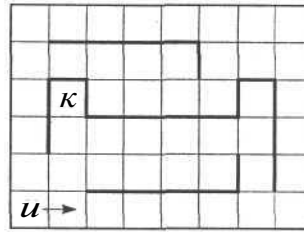
В алгоритмах порождения комбинаторных объектов нас прежде всего будет интересовать сложность алгоритмов, т.е. общее количество времени, требующегося для порождения всего множества объектов.

4.1. 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 H в другой заданный квадрат K путем последо-

вательного перемещения по квадратам. Трудность состоит 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 не исследованный путь;
- если из исследуемого в данный момент квадрата не ведут не исследованные 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, \dots) 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 $(a_1, a_2, a_3, \dots, a_i) \in A_1 \times A_2 \times \dots \times A_i$, для $i = 0, 1, 2, \dots$ в качестве возможных решений. В качестве исходного частичного решения примем пустой вектор $()$ и на основании имеющихся ограничений выясним, какие элементы из A_1 являются кандидатами в a_1 . Обозначим это подмножество кандидатов через $S_1 \subset A_1$. В результате имеем частичное решение (a_1) . В общем случае для расширения частичного решения от $(a_1, a_2, \dots, a_{k-1})$ до $(a_1, a_2, \dots, a_{k-1}, a_k)$ кандидаты на роль a_k выбираются из $S_k \subset A_k$. Если частичное решение $(a_1, a_2, \dots, a_{k-1})$ не представляет возможности для выбора элемента a_k , то $S_k = \emptyset$; 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. Процедура поиска с возвращением для нахождения всех решений формально представлена в алгоритме 4.1.

Алгоритм 4.1. *Общий алгоритм поиска с возвращениями*

```

 $S_1 = A_1$ ; {Выделить кандидатов}
 $k = 1$ ; {Длина частичного решения}
while  $k > 0$  do begin
  while  $S_k \neq \emptyset$  do begin
     $a_k \in S_k$ ; {Расширить частичное решение }
     $S_k = S_k - \{a_k\}$ ; {Удалить выбранного кандидата}
    if  $(a_1, a_2, \dots, a_k)$  — решение then Сохранить решение;
     $k = k + 1$ ; {Расширить частичное решение}
     $S_k \leftarrow S_k \cup \{a_k\}$ ; {Выделить кандидатов}
  end;
   $k = k - 1$ ; {Вернуться, уменьшить частичное решение}
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 $|A_k| = C$ — констан-

ты, 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.

4.2. Перестановки различных элементов

Перестановки множества $\{a_1, a_2, \dots, a_n\}$ 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 $1, 2, \dots, n$.

Порождение перестановок на основе метода поиска с возвращениями выполняется в алгоритме 4.2. Заметим, что в процессе приспособления общего алгоритма 4.1 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 $S_k \neq 0$ соответствует условию $s_k < n$, поскольку алгоритм устроен так, что перебор значений элемента s_k выполняется в порядке их возрастания. Поэтому неравенство $s_k > n$ соответствует пустому множеству кандидатов $S_k = 0$.

Алгоритм 4.2. Порождение перестановок методом поиска с возвращением

```

 $s_1 = 1$ ; {Первый кандидат}
 $k = 1$ ; {Длина частичного решения}
while  $k > 0$  do begin
  while  $s_k \leq n$  do begin
     $a_k = s_k$ ; {Расширить частичное решение }
     $s_k = s_k + 1$ ; {Удалить выбранного кандидата}
    while  $s_k \leq n$  and not flag( $s_k$ ) do  $s_k = s_k + 1$ ;
    if  $k = n$  then Перестановка  $(a_1, a_2, \dots, a_n)$  — решение;
    else begin
       $k = k + 1$ ; {Расширить частичное решение}
       $s_k = 1$ ;
      while  $s_k \leq n$  and not flag( $s_k$ ) do  $s_k = s_k + 1$ ;
    end;
  end;
   $k = k - 1$ ; {Вернуться, уменьшить частичное решение}
end;
Function flag( $s_k$ ) {Поиск элемента  $s_k$  в перестановке  $(a_1, a_2, \dots, a_{k-1})$ }
flag = TRUE;
 $i = 1$ ;
while  $i < k$  and flag do begin
  if  $a_i = s_k$  then flag = FALSE;
   $i = i + 1$ ;
end.

```

Программа на языке Pascal реализации рассмотренного метода генерации перестановок приводится в алгоритме 4.3. Следует отметить, что в программе размерность (длина) и перестановок читается из файла и порождаемые перестановки также сохраняются в файле. Попутно программа вычисляет время порождения всех перестановок с точностью до сотых долей секунды, которое сохраняется в конце файла сгенерированных перестановок.

Алгоритм 4.3. Программа на Pascal'e порождения перестановок методом поиска с возвратом

```

Program Start_BackTrack; {Порождение перестановок}
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;
  { Поиск элемента sk в перестановке a[1],a[2],...,a[k-1] }
  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;{Flag}

Procedure BackTrack (var f:Text; Var a:Vector; n:Integer);
{ Генерация перестановок a[1],a[2],...,a[n] }
Const m :LongInt=0; {Количество перестановок}
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 {Поиск следующего кандидата на место a[k]}
        s[k]:=s[k]+1;
      until (s[k]>n) or Flag(a,s[k],k);
      if k=n then begin {Перестановка найдена}
        m:=m+1;
      end;
    end;
  end;
end;

```

```

        Write(f,m,' ');
        for i:=1 to n do Write(f,a[i],' '); WriteLn(f);
    end
    else begin{Поиск первого кандидата на место a[k+1]}
        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;{BackTrack}
Var {Main}
    a :Vector;
    n :Integer;
    f :Text; {Текстовый файл}
    Hour,Minute,Second,Sec100,rHour,rMinute,rSecond,
    rSec100 :Word;
    delta :LongInt;
begin{Main}
    Assign(f,'BkTrack.in');
    Reset(f);{Файл открыт для чтения}
    ReadLn(f,n);{Ввод длины перестановки}
    Close(f);
    Assign(f,'BkTrack.out');
    Rewrite(f); {Файл открыт для записи}
    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{Main}.

```

4.3. Эффективное порождение перестановок

Последовательность я! перестановок на множестве $\{1, 2, \dots, n\}$, в которой соседние перестановки различаются так мало, как только возможно, — лучшее, на что можно надеяться с точки зрения мини-

мизации 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 $\pi_1, \pi_2, \pi_3, \dots$ на множестве $\{1, 2, \dots, n\}$, 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:

$$\begin{array}{l}
 \left. \begin{array}{l} \{1234 \\ 1243 \\ 1423 \\ 4123 \} \\ \{12, 132\} \left\{ \begin{array}{l} 4132 \\ 1432 \\ 1342 \\ 1324 \end{array} \right. \\ \{312\} \left\{ \begin{array}{l} 3124 \\ 3142 \\ 3412 \\ 4312 \end{array} \right. \\ \{321\} \left\{ \begin{array}{l} 4321 \\ 3421 \\ 3241 \\ 3214 \end{array} \right. \\ \{21, 231\} \left\{ \begin{array}{l} 2314 \\ 2341 \\ 2431 \\ 4231 \end{array} \right. \\ \{213\} \left\{ \begin{array}{l} 4213 \\ 2413 \\ 2143 \\ 2134 \end{array} \right. \end{array} \right\}
 \end{array}$$

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 $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, обратной к ней перестановки $p = (p_1, p_2, \dots, p_n)$ 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 я, то в л легко найти место следующего меньшего элемента. Алгоритм 4.4 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 4.4 — один из наиболее эффективных алгоритмов для порождения перестановок.

Рабочая программа на языке Pascal реализации эффективного метода генерации перестановок приводится в алгоритме 4.5.

В качестве примера в алгоритме 4.6 приводится программа реализации эффективного метода генерации перестановок, написанная на языке Си.

Алгоритм 4.4. Метод эффективного порождения перестановок

```

for i = 1 to n do {  $\pi_i = p_i = i;$ 
                   $d_i = -1;$ 
                  }
d1 = 0;
 $\pi_0 = \pi_{n+1} = m = n + 1;$  {Меткиграницы}
Print  $\pi = (\pi_1, \pi_2, \dots, \pi_n);$ 
m = n;
while m ≠ 1 do • while  $\pi_{p_m : d_m} > m$  do {  $m \sim m$ 
                                                 $[m = m - 1$ 
                                                }
 $\pi_{p_m} \leftrightarrow \pi_{p_m + d_m};$  {Изменить $\pi$ }
 $p_{r_m} \leftrightarrow p_m;$  {Изменить  $p = \pi$ ,  $\pi_{p_m + d_m} = m$ }

```

Алгоритм 4.5. Программа на Pascal'e порождения перестановок эффективным методом

```

Program Start_Effect; {Эффективная генерация
                      перестановок}
uses CRT, DOS;
Const max_n=20; { n<=max_n }
Type Vector=array [0..max_n+1] of Integer;
Var f :Text;
{ Генерация перестановок z[1], z[2], .../ z[n] }

```

```

Procedure Effect( Var z:Vector; n:Integer );
Const k :LongInt=0; {Количество перестановок}
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
    { Печать перестановки }
    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;(Effect)
Var {Main}
  z :Vector;
  n :Integer; {Длина перестановки}
  Hour,Minute,Second,Sec100 :Word;
  rHour,rMinute,rSecond,rSec100 :Word;
  delta :LongInt;
begin
  Assign(f,'Effect.in');
  Reset(f); {Файл открыт для чтения}
  ReadLn(f,n); {Чтение длины перестановки}
  Close(f);
  Assign(f,'Effect.out');
  Rewrite(f); {Файл открыт для записи}
  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 4.6. Программа на Си порождения перестановок
эффективным методом*

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <dos.h>

void main () { //Генерация перестановок
    // z[1], z[2], ..., z[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)); //Перестановка
    p=(int*)malloc((n+2)*sizeof(int)); //Обратная
    d=(int*)malloc((n+2)*sizeof(int)); //Смещение
    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 ) {
        //Печать перестановки
        k++; fprintf(f, "\n%d", 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);
}

```

4.4. Порождение подмножеств множества

Порождение подмножеств множества $\{a_1, a_2, \dots, a_n\}$ 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 4.7.

Перевод этого алгоритма на язык подмножеств множества $\{a_1, a_2, \dots, a_n\}$ осуществляется согласно алгоритму 4.8, где добавлен фиктивный элемент a_{n+1} .

Алгоритм 4.7. Счет в системе счисления с основанием 2 для порождения всех n -разрядных наборов

```

for i = 0 to n do  $b_i = 0$ ;
while  $b_n \neq 1$  do
  {
    Print( $b_{n-1}, b_{n-2}, \dots, b_0$ );
    i = 0;
    while  $b_i = 1$  do
      {
         $b_i = 0$ ;
        i = i + 1;
      }
     $b_i = 1$ .
  }

```

Алгоритм 4.8. Порождение подмножеств счетом в двоичной системе счисления

```

S =  $\emptyset$ ;
while  $a_{n+1} \notin S$  do
  {
    Print(S),
    i = 0;
    while  $a_i \in S$  do
      {
         $i = i + 1$ ;
      }
    S =  $S \cup \{a_i\}$ .
  }

```

Задача «Счастливый билет». Дано n ($n \geq 2$) произвольных цифр: a_1, a_2, \dots, a_n , где $a_i \in \{1, 2, \dots, 9\}$, и произвольное целое число m . Написать программу, которая расставляла бы между каждой парой цифр a_1, a_2, \dots, a_n , записанных именно в таком порядке, знаки $+$, $-$ так, чтобы значением получившегося выражения было число m . Например, если a_1, a_2, \dots, a_n соответственно равны 1, 2, ..., 9 и $m = 5$, то подойдет следующая расстановка знаков: $1-2+3-4+5-6+7-8+9$. Если требуемая расстановка невозможна, то сообщить об этом.

Исходные данные вводятся из текстового файла, имеющего следующую структуру:

Первая строка — целое число n .

Вторая строка — целое число m .

Третья строка — цифры a_1, a_2, \dots, a_n через пробелы.

Результаты расчетов сохранить в текстовом файле.

Пример файла исходных данных:

```
14
71
6 5 8 8 4 7 5 2 3 4 5 7 8 9
```

Пример файла выходных данных:

```
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 « \pm » между каждой парой цифр a_1, a_2, \dots, 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 « $-$ » — 0.

Программа решения задачи «счастливый билет» представлена алгоритмом 4.9. Процедура SubSet (подмножество) этой программы реализует рассмотренный выше алгоритм 4.7 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 « $-$ » — 0.

**Алгоритм 4.9. Программа на Pascal'решения задачи
«Счастливый билет»**

```

Program Lucky_ticket; {Счастливый билет}
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 );

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 {Main}
  a      :Vector;
  n,m,i  :Integer;
  f      :Text; {Текстовый файл}
  Hour,Minute,Second,Sec100 :Word;
  rHour,rMinute,rSecond,rSec100 :Word;
  delta  :LongInt;
begin{Main}
  Assign(f,'Number.in');
  Reset(f); {Файл открыт для чтения}
  Read(f,n); {Ввод количества цифр}
  Read(f,m); {Ввод счастливой суммы}
  for i:=1 to n do Read(f,a[i]);
  Close(f);
  Assign(f,'Number.out');
  Rewrite(f); {Файл открыт для записи}
  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{Main}.

```

4.5. Генерация размещений с повторениями

Порождение множества всех размещений с повторениями длины k из элементов $\{a_0, a_1, \dots, a_{n-1}\}$ 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 4.10 использует фиктивный элемент b_k при порождении наборов длины k в системе счисления с основанием n , где

$b_i \in \{0, 1, \dots, n-1\}$, $i = 0, 1, \dots, k$, т.е. b_i — это цифры генерируемого числа в системе счисления с основанием n .

Алгоритм 4.10. Счет в системе счисления с основанием n для порождения всех k -разрядных наборов

```

for i = 0 to k do  $b_i = 0$ ;
while  $b_k \neq 1$  do
  Print( $b_{k-1}, b_{k-2}, \dots, b_0$ );
  i = 0;
  while  $b_i = n-1$  do
    {  $i = i - 1$ ;
       $i = i + 1$ ;
    }
   $b_i = b_i + 1$ .

```

4.6. Порождение сочетаний

Обычно требуются не все подмножества множества $\{a_1, a_2, \dots, a_n\}$, 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 $\{1, 2, \dots, n\}$; таким образом, будем порождать все сочетания длины k из целых чисел $\{1, 2, \dots, n\}$. 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 $\{1, 2, \dots, k\}$ следующее сочетание находится просмотром текущего сочетания справа налево с тем, чтобы определить место самого правого элемента, который еще не достиг своего максимального значения. Этот элемент увеличивается на единицу, и всем элементам справа от него присваиваются новые возможные наименьшие значения, как показано в алгоритме 4.11. Алгоритм порождения всех C_n^k сочетаний линеен, его сложность $C_n^k + o(C_n^k)$.

Алгоритм 4.11. Порождение сочетаний

```

 $c_0 = -1;$ 
for  $i = 0$  to  $k$  do  $c_i = i;$ 
 $j = 1;$ 
while  $j \neq 0$  do
   $\left\{ \begin{array}{l} \text{Print}(c_1, c_2, \dots, c_k); \\ j = k; \\ \text{while } C_j = n - k + j \text{ do } j = j - 1; \\ C_j = C_j + 1; \\ \text{for } i = j + 1 \text{ to } k \text{ do } c_i = c_{i-1} + 1. \end{array} \right.$ 

```

Задача. Выпуклый многоугольник. Дано множество пар целых чисел $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ — 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, вершинами которых являются исходные точки $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, без рассматриваемой точки (x_i, y_i) . Всего треугольников из n точек можно составить $C_n^3 + O(n^3)$. 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 4.12.

Алгоритм 4.12. Программа поиска точек выпуклой оболочки

```

Program Start_Envelope; {Выпуклая оболочка}
uses CRT, DOS;
const n_max=100;

type Vector=array[1..n_max] of Integer;
      Combine=array[0..3] of Integer;

var f :Text; {Текстовый файл}
    x,y :Vector; {Координаты точек}
    z :Vector; {Характеристический вектор выпуклой обол-
очки}

Procedure Envelope( var c:Combine; n:Integer );
                                {Формирование оболочки}
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 );
    {Генерация сочетаний из n по 3}
Const k :Integer=3; {Длина сочетания}
Var c   :Combine; {Сочетание}
    i, j :Integer;
begin
    for i:=1 to k do c[i]:=i;
    j:=1;
    while j<>0 do begin
        {Print(c, k);}
        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 {Main}
    i, n :Integer; {Число исходных точек}
begin {Main}
    Assign(f, 'Envelope.in');
    Reset(f); {Файл открыт для чтения}
    Read(f, n); {Ввод данных}
    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); {Файл открыт для записи}
    Combination(n);
    WriteLn(f); {Результаты: номера точек
                выпуклой оболочки}
    for i:=1 to n do Write(f, z[i], ' ');
    Close(f);
end. {Main}

```

4.7. Порождение композиций и разбиений

Рассмотрим задачу порождения разбиений положительного числа n в последовательность неотрицательных целых чисел $\{z_1, z_2, \dots, z_k\}$, так что $z_1 + z_2 + \dots + z_k = n$.

Разбиение $\{z_1, z_2, \dots, z_k\}$ называется *композицией* числа n , если учитывается порядок чисел z_i . Как правило, представляют интерес композиции, в которых либо все $z_i > 0$, либо все $z_i > 0$.

Разбиение $\{z_1, z_2, \dots, z_k\}$ называется *разбиением* числа n , если все $z_i > 0$ и порядок чисел z_i не важен. По сути *разбиение* $\{z_1, z_2, \dots, z_k\}$ числа n является *мультимножеством* (см. п. 1.8).

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, $z_i \geq 0$.
 (3), (1, 2), (1, 1, 1) — все разбиения числа три.

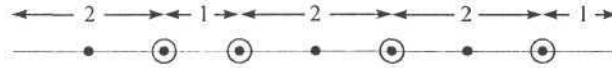
Композиции $z_i > 0$

Композицию $\{z_1, z_2, \dots, z_k\}$, где $z_i > 0$, числа $z_1 + z_2 + \dots + z_k = n$ можно интерпретировать следующим образом. Каждое значение $z_i = 1_i + 1_i + \dots + 1_i$ 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 $\{1_1, 1_2, \dots, 1_k\}$, значение каждого из них равно единице. Теперь любую композицию можно представить как сумму, составленную из n произвольных единиц множества $\{1_1, 1_2, \dots, 1_k\}$. Суммируя подобные единицы 1_i с одинаковыми индексами, получим соответствующие значения z_i композиции. Данное соответствие является взаимно однозначным, откуда и следует, что число композиций равно числу сочетаний с повторениями $\bar{C}_k^n = C_{n+k-1}^{k-1} = C_{n+k-1}^n$. Каждое из сочетаний C_{n+k-1}^n 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 обратное. Суммируя в таком числе слева направо единицы между нулями (их $k - 1$), будем получать соответствующие значения членов z_i (их k) композиции. Например, одному из $C_{n+k-1}^n = C_{17+7-1}^{17}$ сочетаний 1101110011110111111010 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 $z_i \geq 0$.

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 $z_1 + z_2 + \dots + z_k = n$ каждый из $z_i > 0$, тогда для новых переменных $r_i = z_i - 1$, ($r_i > 0$) соответствующая композиция (r_1, r_2, \dots, r_k) будет для числа $n - k = r_1 + r_2 + \dots + r_k$, $r_i \geq 0$. Генерация слагаемых $r_i \geq 0$ композиции (r_1, r_2, \dots, r_k) подробно разобрана в предыдущем разделе, добавляя к каждому из r_i по единице, получим слагаемые $z_i > 0$ композиции $\{z_1, z_2, \dots, z_k\}$.

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:

$$\{m_1 \bullet z_1, m_2 \bullet z_2, \dots, m_k \bullet z_k\},$$

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 условию

$z_1 > z_2 > \dots > z_k$. 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 ве-

либо ($m_k > 1$), 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 $z_k + 1$. Все, что остается, превращается в соответствующее число единиц и формируется новое разбиение.

{7*1}	= {1, 1, 1, 1, 1, 1, 1}
{1*2, 5*1}	= {2, 1, 1, 1, 1, 1}
{2*2, 3*1}	= {2, 2, 1, 1, 1}
{3*2, 1*1}	= {2, 2, 2, 1}
{1*3, 4*1}	= {3, 1, 1, 1, 1}
{1*3, 1*2, 2*1}	= {3, 2, 1, 1}
{1*3, 2*2}	= {3, 2, 2}
{2*3, 1*1}	= {3, 3, 1}
{1*4, 3*1}	= {4, 1, 1, 1}
{1*4, 1*2, 1*1}	= {4, 2, 1}
{1*4, 1*3}	= {4, 3}
{1*5, 2*1}	= {5, 1, 1}
{1*5, 1*2}	= {5, 2}
{1*6, 1*1}	= {6, 1}
{1*7}	= {7}

Алгоритм 4.13 использует рассмотренный процесс для порождения всех разбиений числа n .

Алгоритм 4.13. Генерация разбиений числа n

```

k = 1;
z-1 = 0;
m-1 = 0;
z0 = n + 1;
m0 = 0;
z1 = 1;
m1 = n;
while k ≠ 0 do
  Print{m1 • z1, m2 • z2, ..., mk • zk};
  Summa = mk zk;
  if mk = 1 then {k = k - 1;
                  Summa = Summa + mk zk;
  if zk-1 = zk + 1 then {k = k - 1;
                          mk = mk + 1;
  else {mk = 1;
  if Summa > zk then {zk+1 = 1;
                      mk+1 = Summa - zk;
                      k = k + 1;

```

Алгоритм 4.13 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 4.14. 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 секунды, которое сохраняется в конце файла сгенерированных разбиений.

Алгоритм 4.14. Программа на Pascal'e генерации разбиений числа n

```

Program Start_Divide; {Разбиение числа n}
uses CRT,DOS;
Const
  max_n=100; { n<=max_n }
Type
  Vector=array[-1..max_n] of Integer;
Var
  f      :Text;
  z,m   :Vector;

Procedure Print( var m,z:Vector; k:Integer );
                                {Печать разбиения}
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; {Print}

Procedure Divide( n:Integer ); {Разбиение числа n}
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(t, 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; {Divide}

Var {Main}
n : Integer; {Число для разбиения}
Hour, Minute, Second, Sec100 : Word;
rHour, rMinute, rSecond, rSec100 : Word;
delta : LongInt;
begin
Assign(f, 'Divide.in');
Reset(f); {Файл открыт для чтения}
ReadLn(f, n); {Чтение числа n}
Close(f);
Assign(f, 'Divide.out');
Rewrite(f); {Файл открыт для записи}
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.

```

4.8. Генерация случайных перестановок

Пусть $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ — произвольно выбранная перестановка целых чисел $1, 2, \dots, n$, например, $\pi = (1, 2, \dots, n)$ — тождественная. Случайную перестановку можно получить за линейное время $O(n)$ из выбранной перестановки $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, выполнив в ней n транспозиций (см. п.7.4).

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 $rand(k, l)$, которая порождает независимые случайные целые числа на отрезке $[k, l]$ с равномерным распределением.

Положим $\pi^{(0)} = (\pi_1^{(0)}, \pi_2^{(0)}, \dots, \pi_n^{(0)})$ равной исходной перестановке $\pi = (\pi_1, \pi_2, \dots, \pi_n)$. Каждая следующая перестановка $\pi^{(k)} = (\pi_1^{(k)}, \pi_2^{(k)}, \dots, \pi_n^{(k)})$ получается из предыдущей перестановки $\pi^{(k-1)} = (\pi_1^{(k-1)}, \pi_2^{(k-1)}, \dots, \pi_n^{(k-1)})$ транспозицией элементов $\pi_k^{(k-1)}$ и $\pi_{rand(k, n)}^{(k-1)}$ где $k = 1, 2, \dots, n$. Между элементами перестановок $\pi^{(0)}, \pi^{(1)}, \dots, \pi^{(n)}$ выполняются равенства $\pi_k^{(n)} = \pi_k^{(n-1)} = \dots = \pi_k^{(k)}$, где $k = 1, 2, \dots, n$.

Покажем, что после выполнения всех указанных n транспозиций равновероятно получение любой из $n!$ возможных перестановок $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_n)$ исходных чисел $\pi_1, \pi_2, \dots, \pi_n$. Для этого достаточно проверить, что $\Pr(\pi^{(n)} = \sigma) = 1/n!$. С этой целью введем события A_1, A_2, \dots, A_n .

$$\begin{aligned}
 A_1 &= \{\pi_1^{(1)} = \sigma_1\} = \{\pi_{rand(1, n)}^{(0)} = \sigma_1\}, \\
 A_k &= \{\pi_k^{(k)} = \sigma_k \mid \pi_1^{(k-1)} = \sigma_1 \& \pi_2^{(k-1)} = \sigma_2 \& \dots \& \pi_{k-1}^{(k-1)} = \sigma_{k-1}\} = \\
 &= \{\pi_{rand(k, n)}^{(k-1)} = \sigma_k \mid \pi_1^{(k-1)} = \sigma_1 \& \pi_2^{(k-1)} = \sigma_2 \& \dots \& \pi_{k-1}^{(k-1)} = \sigma_{k-1}\} = \\
 &= \{\pi_k^{(k)} = \sigma_k \mid \pi_1^{(1)} = \sigma_1 \& \pi_2^{(2)} = \sigma_2 \& \dots \& \pi_{k-1}^{(k-1)} = \sigma_{k-1}\}, \quad k = 2, \dots, n.
 \end{aligned}$$

Вероятности данных событий, согласно схеме формирования перестановок $\pi^{(0)}, \pi^{(1)}, \dots, \pi^{(n)}$, равны

$$\Pr(A_k) = 1/(n - k + 1), \quad \text{где } k = 1, 2, \dots, n.$$

Условие $\pi_1^{(k-1)} = \sigma_1 \& \pi_2^{(k-1)} = \sigma_2 \& \dots \& \pi_{k-1}^{(k-1)} = \sigma_{k-1}$ в событии A_k , $k = 2, \dots, n$, обеспечивает выбор элемента $\pi_{rand(k,n)}^{(k-1)}$ из множества $\{\pi_k^{(k-1)}, \pi_{k+1}^{(k-1)}, \dots, \pi_n^{(k-1)}\}$, которое совпадает с множеством $\{\sigma_k, \sigma_{k+1}, \dots, \sigma_n\}$. Индекс же $rand(k, n)$ элемента $\pi_{rand(k,n)}^{(k-1)}$ является независимой случайной величиной с равномерным распределением на отрезке $[k, n]$ целых чисел.

Теперь заметим, что

$$\begin{aligned} \text{Pr}(\pi^{(n)} = \sigma) &= \text{Pr}(\pi_1^{(n)} = \sigma_1 \& \pi_2^{(n)} = \sigma_2 \& \dots \& \pi_n^{(n)} = \sigma_n) = \\ &= \text{Pr}(\pi_1^{(1)} = \sigma_1 \& \pi_2^{(2)} = \sigma_2 \& \dots \& \pi_n^{(n)} = \sigma_n) = \\ &= \text{Pr}(A_n) \times \text{Pr}(A_{n-1}) \times \dots \times \text{Pr}(A_1) = \frac{1}{1} \times \frac{1}{2} \times \dots \times \frac{1}{n} = \frac{1}{n!}. \end{aligned}$$

Рассмотренный метод генерации случайной перестановки представлен в алгоритме 4.15.

Алгоритм 4.15. Генерация случайной перестановки

for $k = 1$ *to* n *do* $\pi_k = k$; {Начальная перестановка}
for $k = 1$ *to* $n - 1$ *do* $\pi_k \leftrightarrow \pi_{rand(k,n)}$; {Случайная перестановка}

или, если генерацию перестановки вести с конца,

for $k = 1$ *to* n *do* $\pi_k = k$; {Начальная перестановка}
for $k = n$ *to* 2 *do* $\pi_k \leftrightarrow \pi_{rand(1,k)}$; {Случайная перестановка}.

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 требуемых 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, \dots, 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 $a_i = a_j$, либо $a_i > a_j$. Требуется найти перестановку $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ этих n элементов, которая отобразит данную последовательность в неубывающую последовательность $a_{\pi_1} \leq a_{\pi_2} \leq \dots \leq a_{\pi_n}$. Как правило, далее будем получать саму упорядоченную последовательность, а не упорядочивающую перестановку π .

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 при-

менимы 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 данных. 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.

5.1. Сортировка вставками

Сортировка вставками элементов a_1, a_2, \dots, a_n относится к наиболее очевидным методам (алгоритм 5.1). Для компактности алгоритма вводится фиктивный элемент a_0 , значение которого устанавливается равным $-\infty$. Сортировка проходит цикл для $j = 2, 3, \dots, n$; для каждого j элемент a_j вставляется в свое правильное место среди a_1, a_2, \dots, a_{j-1} . При вставке элемент a_j временно размещается в w и просматриваются имена $a_{j-1}, a_{j-2}, \dots, a_1$; 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 5.1. Сортировка вставками

$$a_0 = -\infty;$$

$$\text{for } j = 2 \text{ to } n \text{ do } \left\{ \begin{array}{l} i = j - 1; \\ w = a_j; \\ \text{while } w < a_i \text{ do } \left\{ \begin{array}{l} a_{i+1} = a_i; \\ i = i + 1; \end{array} \right. \\ a_{i+1} = w. \end{array} \right.$$

Сложность алгоритма определяется числом проверок условия $w < a_i$ в цикле. Сравнение $w < a_i$ для конкретного $w = a_j$ ($j > 2$) выполняется $1 + d_j$ раз, где d_j — число элементов, больших a_j и стоящих слева от него, т.е. d_j — это число инверсий, у которых второй элемент a_j . Числа d_j составляют таблицу инверсий $d_1 d_2 \dots d_n$, а так как $0 < d_1 < n - 1$, $0 < d_2 < n - 2, \dots$, $0 < d_{n-1} < 1$, $d_n = 0$, то в худшем случае сортировка элементов a_1, a_2, \dots, a_n потребует $\sum_{j=2}^n (1 + d_j) \leq \sum_{j=2}^n (1 + n - j) = \frac{n(n-1)}{2} \sim \frac{1}{2}n^2$ сравнении. Сложность сортировки вставками является квадратичной.

5.2. Пузырьковая сортировка

Рассматриваемый метод пузырьковой сортировки последовательности a_1, a_2, \dots, a_n представляет собой наиболее очевидный метод систематического обмена местами слева направо смежных элементов, не отвечающих выбранному порядку, до тех пор пока каждый элемент 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 5.2. В алгоритме используется переменная b , значение которой при каждом проходе цикла устанавливается равным наибольшему индексу t , такому, что все элементы $a_{t+1}, a_{t+2}, \dots, a_n$ уже находятся на своих окончательных позициях. Ясно, что не имеет смысла продолжать просмотр для указанных элементов.

Алгоритм 5.2. Пузырьковая сортировка

```

b = n;
while b ≠ 0 do {
  t = 0;
  for j = 1 to b - 1 do if  $a_j > a_{j+1}$  then {
     $a_j \leftrightarrow a_{j+1}$ ;
    t = j;
  }
  b = t.
}

```

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, \dots, 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 $\sum_{b=n}^1 (b-1) = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} = O(n^2)$. Сложность пузырьковой сортировки является квадратичной.

В алгоритме 5.3 представлена «полная» пузырьковая сортировка. Это наиболее популярный и упрощенный вариант алгоритма 5.2. Ясно, что основным достоинством алгоритма полной пузырьковой сортировки является легкость программирования. Сложность же алгоритма 5.3 остается постоянной, равной $\sum_{i=1}^n (n-i) = (n-1) + (n-2) + \dots + 1 = \frac{n(n-1)}{2} = O(n^2)$, и не зависит от расположения исходных данных.

Алгоритм 5.3. Полная пузырьковая сортировка

```

for i = 1 to n do begin
  for j = 1 to n - i do begin
    if aj > aj+1 then aj ↔ aj+1
  end;
end.

```

5.3. Сортировка перечислением

Идея сортировки последовательности a_1, a_2, \dots, a_n данных перечислением состоит в том, чтобы сравнить попарно все элементы a_1, a_2, \dots, a_n и подсчитать, сколько из них меньше каждого отдельного элемента (алгоритм 5.4). Для подсчета числа элементов, меньших данного, в алгоритме используется вспомогательный вектор c_1, c_2, \dots, c_n . После завершения алгоритма значения $c_j + 1, j = 1, 2, \dots, n$ определяют окончательное положение элементов a_j в сортированной последовательности r_1, r_2, \dots, r_n .

Алгоритм 5.4. Сортировка перечислением

```

for i = 1 to n do ci = 0 { Сбросить счетчики }
for i = n to 1 by -1 do begin
  for j = i - 1 to 1 by -1 do begin
    if ai > aj then ci = ci + 1
  else C = cj + 1
  end;
end;

```

```

end;
for i = 1 to n do begin;
  rci+1 = ai { ri — сортированные элементы }
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, \dots, a_n .

Пусть перестановка $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, где $\pi_j = c_j + 1, j = 1, 2, \dots, n$. Алгоритм 5.4 сортировки перечислением определяет перестановку π^{-1} , которая соответствует расположению $a_{\pi_1^{-1}} \leq a_{\pi_2^{-1}} \leq \dots \leq a_{\pi_n^{-1}}$ исходных данных (см. п. 1.14).

5.4. Сортировка всплытием Флойда

Все ранее упомянутые методы сортировки последовательности a_1, a_2, \dots, a_n требовали сравнений порядка $O(n^2)$, и «это никуда не годится». Рассмотрим один из наиболее элегантных и эффективных методов сортировки сложности $O(n \log n)$, предложенный Флойдом. До сих пор он остается самым оптимальным из существующих методов. В алгоритме активно используется *упорядоченное двоичное дерево*, пример которого представлен на рис. 5.1.

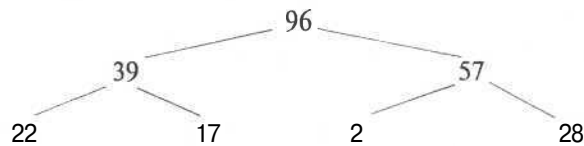


Рис. 5.1. Пример упорядоченного двоичного дерева

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 нарушается. Пример частично упорядоченного дерева приведен на рис. 5.2.

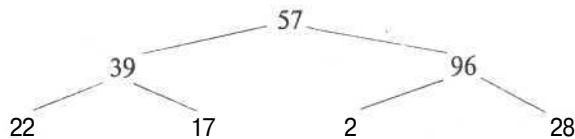


Рис. 5.2. Пример частично упорядоченного двоичного дерева

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. Структура же дерева позволяет сохранить состояние процесса сортировки последовательности a_1, a_2, \dots, 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.

Метод сортировки Флойда представлен в алгоритме 5.5, где исходная последовательность a_1, a_2, \dots, 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. 5.3. 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 5.5.

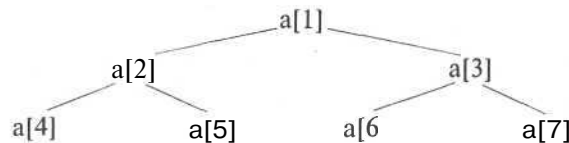


Рис. 5.3. Пример двоичного дерева на смежной памяти

Основу алгоритма 5.5 составляет процедура *SURFACE* ($a[i..k]$) всплытия Флойда, которая за $O(\log_2 n)$ сравнений преобразует почти упорядоченное поддереву в упорядоченное. Поддерево представляется на одномерном массиве $a[i..k]$, рис. 5.4, где $a[i]$ — корень поддерева, $a[k]$ — максимальный элемент массива, который еще может принадлежать поддереву.

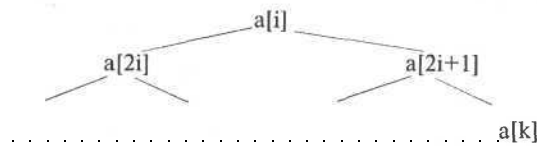


Рис. 5.4. Двоичное поддереву на смежной памяти $a[i..k]$

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 $n < 2^0 + 2^1 + \dots + 2^{h-1}$, где 2^{i-1} — количество вершин на i -м уровне дерева, $i = 1, 2, \dots, h$. 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, \dots, a_n из них по алгоритму 5.5 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 5.5 осуществляется формирование исходного почти упорядоченного дерева.

После того как дерево упорядочено, наибольший (наименьший) элемент оказывается в его корне. По алгоритму 5.5 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. 5.5 показана полная последовательность перестановок и всплытий, которые происходят после формирования из исходно-

го множества почти упорядоченного дерева и вплоть до того, как в этом дереве останется всего одна вершина, а исходное множество окажется отсортированным.

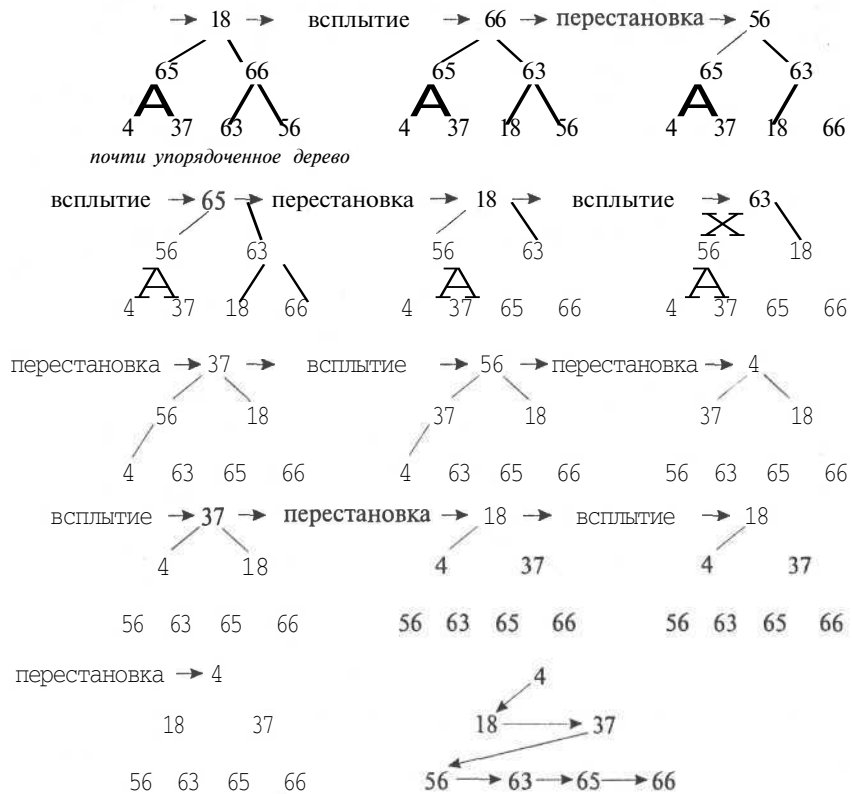


Рис. 5.5. Пример сортировки чисел 18,4,56,65,37,63,66 методом Флойда

Алгоритм 5.5. Сортировка всплытием Флойда сложности $O(n \log_2 n)$

```

Program Floyd; { Сортировка по возрастанию
                всплытием Флойда }
uses CRT;
const n = 1000; {Размер массива данных для сортировки}
type
  Vector = array[1..n] of Integer;
Var
  f :Text; {Текстовый файл для результатов сортировки}

```



```
procedure Init( var a: vector; n: integer);
{ Заполнить вектор a[1..n] случайными числами }
var
  i: integer;
begin
  Randomize;
  for i:=1 to n do a[i]:=Random(100);
end;

procedure Surface( var a: Vector; i,k: integer);
{ Процедура всплытия Флойда по дереву a[i..k] }
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; {выход из цикла}
  end;
  a[i]:=copy;
end;

procedure Sort( var a: Vector; n: integer);
{ Сортировка вектора a[1..n] методом Флойда }
var
  i,k,w :Integer;
begin
  {Формировать исходное частично упорядоченное дерево}
  for i:=n div 2 downto 2 do Surface(a,i,n) ;
  {Выполнить процедуру всплытия Флойда
    для каждого поддерева}
  for k:=n downto 2 do begin
    Surface(a,1,k) ;
    {Поместить найденный максимальный элемент
      в конец списка}
    w:=a[k]; a[k]:=a[1]; a[1]:=w;
  end
end;
```

```

Var {Main}
  a : Vector; {Вектор исходных данных для сортировки}
  i : Integer;
begin; {Main}
  Assign(f, 'sort.out');
  Rewrite(f); {Файл открыт для записи}
  Init(a, n);
  {Сохранить исходные данные}
  for i:=1 to n do WriteLn(f, 'a[ ', i:1, ']=' , a[i]:3);
  Sort(a, n);
  {Сохранить сортированные данные}
  WriteLn(f);
  for i:=1 to n do WriteLn(f, 's[ ', i:1, ']=' , a[i]:3);
  Close(f);
end. {Main}

```

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, \dots, 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, \dots, 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 $\log_2 n! \sim \sqrt{2\pi n} n^n e^{-n} = O(n \log_2 n)$.

Задача. Длина объединения отрезков. Текстовый файл содержит целые числа: $a_1, b_1, a_2, b_2, \dots, a_n, b_n$. Данная последовательность чисел определяет на прямой n отрезков $[a_i, b_i]$, $i = 1, 2, \dots, n$. 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
o 2
-1 1
o 1
```

Пример файла выходных данных:

```
3
```

Решение. Алгоритм 5.6 решения задачи основывается на предварительной сортировке абсцисс $a_1, b_1, a_2, b_2, \dots, a_n, b_n$ 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 5.6. Программа расчета длины объединения отрезков

```
Program MeasureLength; {Длина объединения отрезков}
uses CRT, DOS;
const n_max=100;
type
  Vector=array[0..2*n_max] of Integer;
var
  f :Text; {Текстовый файл}
  ab :Vector; {Границы отрезков a[i], b[i]}
  g :Vector; {Признаки границ: 1-левая, 0-правая}

Procedure Measure( Var m: LongInt; n:Integer );
{Расчет объединения}
Var
  i, c : Integer;
begin
  ab[0]:=ab[1];
  m:=0; {Длина объединения}
  c:=0; {Число перекрывающихся интервалов}
  for i:=1 to n do begin
    if c<>0 then m:=m+ab[i]-ab[i-1];
    if g[i]=1 {-левая граница} then c:=c+1 else c:=c-1;
  end;
end;
```

```
Procedure SortBubble( n:Integer );
    {Сортировка границ a[i],b[i] и g[i]}
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 {Main}
i, k :Integer;
n :Integer; {Число исходных точек}
m :LongInt; {Длина объединения отрезков}
begin {Main}
Assign(f, 'Measure.in');
Reset(f); {Файл открыт для чтения}
Read(f,n); {Ввод данных}
for i:=1 to n do begin
    k:=2*i;
    Read(f,ab[k-1],ab[k]);
    g[k-1]:=1; {Левая граница}
    g[k]:=0; {Правая граница}
end;
Close(f);
Assign(f, 'Measure.out');
Rewrite(f); {Файл открыт для записи}
SortBubble (2*n);
Measure (m, 2*n);
WriteLn(f,m); {Длина объединения}
Close (f);
end. {Main}
```

5.5. 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, \dots, 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 5.7 выполняет последовательный поиск элемента z в множестве a_1, a_2, \dots, a_n . Несмотря на свою простоту, последовательный поиск содержит ряд очень интересных идей.

Алгоритм 5.7. Последовательный поиск

```

c = 0; {Признакпоиска записи z}
for i = 1 to n do if z = ai then {c = 1;
                                break;}

```

if c = 1 then Запись найдена **else** Запись не найдена.

Оценим среднюю сложность поиска элементов множества a_1, a_2, \dots, 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 p_i обозначает частоту (распределение вероятностей) обращения к элементу a_i , где $p_i \geq 0$ и $\sum_{i=1}^n p_i = 1$. В этом случае средняя сложность (математическое ожидание) поиска элемента будет равна $\sum_{i=1}^n i p_i$. Хорошим приближением распределения частот к действительности является закон Зипфа: $p_i = \frac{c}{i}$, для $i = 1, 2, \dots, n$. (Дж. К. Зипф заметил, что n -е наиболее употребительное в тексте на естественном языке слово встречается с частотой, приблизительно обратно пропорциональной i .) Нормирующая константа выбирается так, что $\sum_{i=1}^n p_i = 1$. Пусть элементы множества a_1, a_2, \dots, a_n упорядочены согласно указанным частотам.

Тогда $c = \frac{1}{\sum_{i=1}^n 1/i} = \frac{1}{H_n} \approx \frac{1}{\ln n}$ и среднее время успешного поиска составит $\sum_{i=1}^n i \cdot \frac{c}{i} \cdot n = \frac{1}{H_n} \approx \frac{n}{\ln n}$, что много меньше $\frac{n+1}{2}$.

Последний пример показывает, что даже простой последовательный поиск требует выбора разумной структуры данных множества, который бы повышал эффективность работы алгоритма. Более того, это общепринятая стратегия, 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, \dots, a_n на смежной или связанной памяти.

5.6. Логарифмический поиск

Логарифмический (бинарный или метод деления пополам) поиск данных применим к сортированному множеству элементов $a_1 < a_2 < \dots < a_n$, 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 $a_1, a_2, \dots, a_{\lfloor \frac{1+n}{2} \rfloor}, \dots, a_n$ 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 $a_1 < a_2 < \dots < a_n$ в виде двоичного дерева сравнений, которое отвечает бинарному поиску.

Двоичное дерево называется *деревом сравнений*, если для любой его вершины (корня дерева или корня поддеревы) выполняется условие:

$$\{\text{Вершины левого поддерева}\} < \text{Вершина корня} < \{\text{Вершины правого поддерева}\}.$$

Пусть на очередном шаге деления пополам оказалось, что необходимо выполнить поиск среди элементов $a_i < a_{i+1} < \dots < a_j$. В качестве корня принимается элемент $a_{\lfloor \frac{i+j}{2} \rfloor}$, где $\lfloor \frac{i+j}{2} \rfloor$ — наибольшее целое, меньшее или равное $(i+j)/2$. Левое поддерево располагается в векторе $a_i, a_{i+1}, \dots, a_{\lfloor \frac{i+j}{2} \rfloor - 1}$, а правое поддерево — в векторе $a_{\lfloor \frac{i+j}{2} \rfloor + 1}, \dots, a_{j-1}, a_j$. На рис. 5.6 показан пример двоичного дерева сравнений, ребра которого неявно выражаются рассмотренными выше отношениями между индексами элементов a_1, a_2, \dots, a_n .

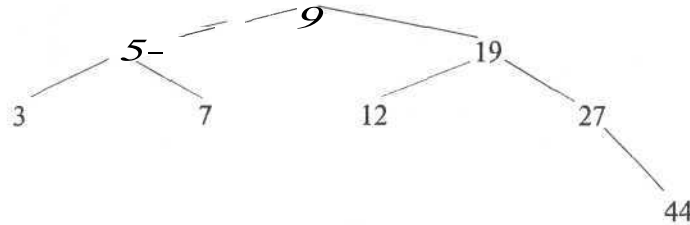


Рис. 5.6. Пример дерева сравнений, отвечающего бинарному поиску среди сортированных элементов: 3,5,7,9,12,19,27,44

Поиск элемента z среди $a_1 < a_2 < \dots < a_n$ методом деления пополам представлен в алгоритме 5.8.

Алгоритм 5.8. Логарифмический поиск z в $a_1 < a_2 < \dots < a_n$

```

find = 0; {Признак поиска записи}
i = 1; {Левая граница поддерева}
j = n; {Правая граница поддерева}
while i ≤ j do begin
  m = ⌊ (i+j)/2 ⌋; {Корень текущего поддерева}
  if z = am then {find = 1} {Элемент найден}
  else if z > am then i = m + 1 {Новая левая граница}
  else j = m - 1 {Новая правая граница}
end;
if find = 1 then Запись найдена;
else Запись отсутствует.
  
```

Средняя сложность бинарного поиска среди элементов $a_1 < a_2 < \dots < a_n$ сравнима с высотой двоичного дерева (рис 5.6). 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.5.4 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.

Необходимо отметить, что рассмотренный метод бинарного поиска предназначен главным образом для сортированных элементов $a_1 < a_2 < \dots < a_n$ на смежной памяти фиксированного n размера. Если же размерность вектора динамически меняется, то экономия от использования бинарного поиска не покрывает затрат на поддержание упорядоченного расположения $a_1 < a_2 < \dots < a_n$.

5.7. Сортировка с вычисляемыми адресами

Пусть a_1, a_2, \dots, a_n — исходная последовательность сортируемых целых чисел и $l = (\pi_1, \pi_2, \dots, \pi_n)$ — упорядочивающая перестановка этих элементов $a_{\pi_1} \leq a_{\pi_2} \leq \dots \leq a_{\pi_n}$. 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}, \dots, b_s , где $b_{a_i} = a_i$. Получим упорядоченную последовательность $b_r < b_{r+1} < \dots < b_s$, а значит, и упорядоченную последовательность исходных данных a_1, a_2, \dots, a_n .

В качестве примера рассмотрим частный случай сортировки элементов последовательности a_1, a_2, \dots, 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 $b_1 < b_2 < \dots < b_7$, где b_i определяются в цикле

$$\text{for } i = 1 \text{ to } 7 \text{ do } b_{a_i} = a_i.$$

Сложность данного метода сортировки является линейной $O(n)$, где $n = 7$. Фактически, значения a_i определяют перестановку $l = (\pi_1, \pi_2, \dots, \pi_n) = (7, 5, 2, 6, 3, 1, 4)$, которая упорядочивает элементы $a_{\pi_1} < a_{\pi_2} < \dots < a_{\pi_n}$. Значения π_i устанавливаются в цикле

$$\text{for } i = 1 \text{ to } 7 \text{ do } \pi_{a_i} = i.$$

Сортировка 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, \dots, a_n . Алгоритм сортировки существенно упрощается, если все значения a_1, a_2, \dots, a_n различные.

Значения a_1, a_2, \dots, a_n — различные

Реализация метода сортировки представлена в алгоритме 5.9. Временный массив b_r, b_{r+1}, \dots, b_s , где $r = \min(a_1, a_2, \dots, a_n)$ и $s = \max(a_1, a_2, \dots, a_n)$, используется для сортированной упаковки элементов a_1, a_2, \dots, a_n . Свободные места в массиве b_r, b_{r+1}, \dots, b_s инициализируются значением $s + 1$, отличным от значений элементов a_1, a_2, \dots, a_n .

**Алгоритм 5.9. Сортировка с вычисляемыми адресами
для различных a_1, a_2, \dots, a_n**

```
{Поиск  $\min$  и  $\max$  значений среди  $a_1, a_2, \dots, a_n$ }
 $r = s = a_1$ ;
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;
{Инициализация  $b_i$  значением  $s+1$ , отличным от  $a_i$ }
for  $i = r$  to  $s$  do  $b_i = s + 1$ ;
{Сортированная упаковка элементов  $a_1, a_2, \dots, a_n$ }
for  $i = 1$  to  $n$  do  $b_{a_i} = a_i$ ;
{Выделение сортированных  $a_1, a_2, \dots, a_n$  из  $b_r, b_{r+1}, \dots, b_s$ }
 $k = 0$ ;
for  $i = r$  to  $s$  do begin
  if  $b_i \neq s + 1$  then begin
     $k = k + 1$ ;
     $a_k = b_i$ ;
  end;
end.
```

Сортированный вектор $a_1 < a_2 < \dots < a_n$ является результатом последовательного просмотра массива b_r, b_{r+1}, \dots, 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, \dots, a_n ,

Реализация метода представлена в алгоритме 5.10. Наличие одинаковых элементов среди a_1, a_2, \dots, a_n , например, $a_i = a_j$, при упаковке $b_a = a_i$ и $b_a = a_j$ ведет к потере данных в исходном множестве $\{j, a_2, \dots, a_n\}$. Ситуация, когда одновременно несколько элементов претендуют на одно место, называется *коллизией*. Такие элементы необходимо перерасмещать на свободные места, сохраняя свойства сортировки с вычисляемыми адресами. С этой целью на основании величин a_1, a_2, \dots, a_n рассчитывается вектор индексов d_r, d_{r+1}, \dots, d_s сортированной упаковки данных a_1, a_2, \dots, a_n в массиве b_1, b_2, \dots, b_n . В алгоритме 5.9 роль индексов упаковки могли выполнять непосредственно сами значения элементов a_1, a_2, \dots, a_n , так как они были различные. В данном случае значения d_r, d_{r+1}, \dots, d_s настраиваются таким образом, что одинаковые по величине элементы из a_1, a_2, \dots, a_n оказываются смежными при их упаковке в массиве b_1, b_2, \dots, b_n . Вектор c_r, c_{r+1}, \dots, c_s используется для подсчета количества элементов каждого значения среди $\{j, a_2, \dots, a_n\}$ и формирования индексов упаковки d_r, d_{r+1}, \dots, d_s .

**Алгоритм 5.10. Сортировка с вычисляемыми адресами
для произвольных a_1, a_2, \dots, a_n**

```

{Поиск min и max значений среди  $a_1, a_2, \dots, a_n$ }
 $r = s = a_1$ ;
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;
{Расчет количества элементов каждого значения  $a_i$ }
for  $i = r$  to  $s$  do  $c_i = 0$ ;
for  $i = 1$  to  $n$  do  $c_{a_i} = c_{a_i} + 1$ ;
{Расчет индексов упаковки  $d_r, d_{r+1}, \dots, d_s$ }
 $d_r = 1$ ;
for  $i = r + 1$  to  $s$  do  $d_i = \text{flf}(\dots) + c_{i-1}$ ;
{Сортированная упаковка элементов  $a_1, a_2, \dots, a_n$  в  $b_1, b_2, \dots, b_n$ }
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, \dots, a_n располагается в массиве $b_1 \leq b_2 \leq \dots < b_n$. 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}, \dots, c_s и d_r, d_{r+1}, \dots, d_s .

Глава 6

Введение в теорию 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.

6.1. Основные понятия и определения

- *Определение.* Конечным графом называется тройка $G = (X, U, \Phi)$, где X — конечное множество вершин; U — конечное множество ребер (дуг); Φ — отношение инцидентности; $X \cap U = \emptyset$. Отношение инцидентности Φ 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 всегда 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 u соответствует не более чем одной паре вершин x, y .

- *Графическое представление графов.*

Элементы графов	Геометрические элементы
1. $x \in X$ — вершина.	1. \bullet — точка в пространстве.
2. $\Phi(x, u, y) \wedge \neg \Phi(y, u, x)$ — ориентированное ребро, дуга.	2. $x \leftarrow \rightarrow y$ — направленный отрезок.
3. $\Phi(x, u, y) \wedge \Phi(y, u, x)$ — неориентированное ребро.	3. $x \longleftrightarrow y$ — отрезок.
4. $\Phi(x, u, x)$ — петля.	4. $x \text{ } \textcircled{\curvearrowright}$ — замкнутый отрезок.

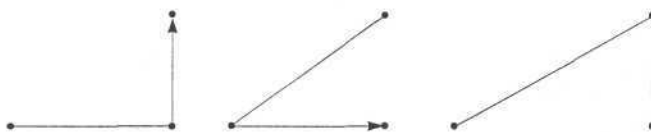


Рис. 6.1. Графы с тремя вершинами и двумя ребрами

- *Определение.* $\Gamma_1 = (X_1, U_1, \Phi_1)$ и $\Gamma_2 = (X_2, U_2, \Phi_2)$ называются изоморфными ($\Gamma_1 = \Gamma_2$), если существуют два взаимно однозначных соответствия $\varphi : X_1 \rightarrow X_2$ и $\psi : U_1 \rightarrow U_2$, сохраняющие 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. 6.2).

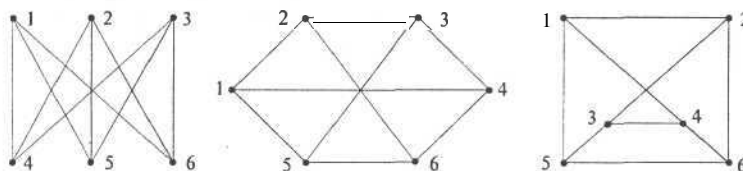


Рис. 6.2. Три изоморфных графа

- *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 противоположной ориентацией.
- *Определение.* Подграфом графа $\Gamma = (X, U, \Phi)$ называется такой граф $\Gamma' = (X', U', \Phi)$, что $X' \subseteq X$, $U' \subset U$. Обозначают Γ' с Γ .
- *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. 6.3).

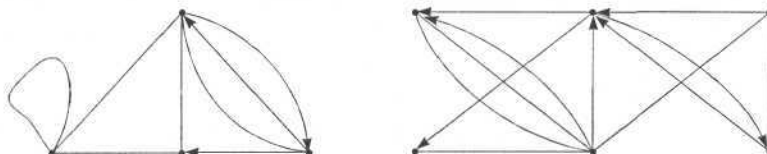


Рис. 6.3. Псевдограф (слева) и мультиграф (справа)

- *Определение.* Неориентированный граф называется *простым*, если он не имеет петель и любая пара вершин соединена не более чем одним ребром.
- *Определение.* Простой граф называется *полным*, если каждая пара вершин соединена ребром. Такой граф с n вершинами содержит C_n^2 ребер (рис. 6.4).

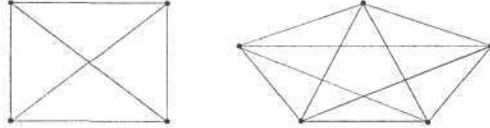


Рис. 6.4. Полные неориентированные графы

- *Определение.* Дополнением простого графа Γ называется граф $\bar{\Gamma}$, имеющий те же вершины, а его ребра являются дополнением Γ до полного графа (рис. 6.5).

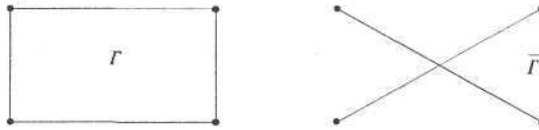


Рис. 6.5. Исходный граф Γ и дополнительный $\bar{\Gamma}$

- *Определение.* Граф называется *плоским (планарным)*, если он *может* быть изображен на плоскости так, что все пересечения ребер являются его вершинами.

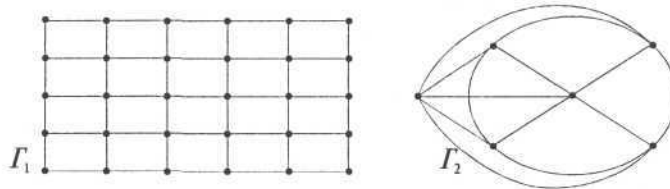


Рис. 6.6. Граф Γ_1 — плоский, а граф Γ_2 — неплоский

- *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 пометками (рис. 6.7).

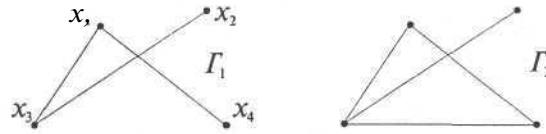


Рис. 6.7. Граф G_1 — помеченный, а граф G_2 — непомеченный

- *Определение.* Путь (маршрут) на графе $G=(X, U, \Phi)$ определяется последовательностью вершин и ребер $x_1 u_1 x_2 u_2 x_3 \dots x_n u_n x_{n+1}$, 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 $G=(X, U, \Phi)$ 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. 6.8).

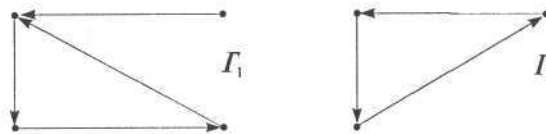


Рис. 6.8. G_1 — слабо связный, G_2 — сильно связный

- *Определение.* Связный неориентированный ациклический граф называется *деревом*, множество деревьев называется *лесом*.

Рис. 6.9. G_1 — дерево, G_2 — не дерево

Большинство задач на графах касается определения 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.

6.2. 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.

6.2.1. Матрица смежности графа

- *Определение.* Матрицей смежности *ориентированного* помеченного графа с n вершинами называется матрица $A = [a_{ij}]$, $i, j = 1, 2, \dots, n$, 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. 6.10 и рис. 6.11. Отметим, что петля в матрице 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.

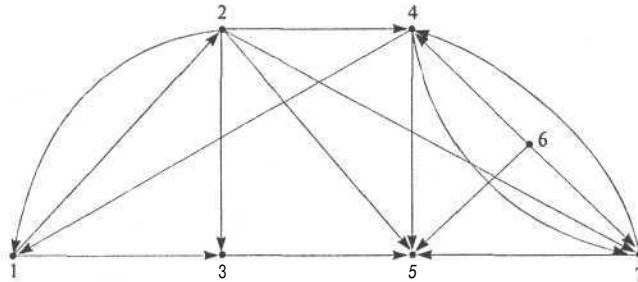


Рис. 6.10. Ориентированный граф

$$A = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Рис. 6.11. Матрица смежности ориентированного графа рис. 6.10

6.2.2. Матрица инцидентности графа

- *Определение.* Матрицей инцидентности для неориентированного графа с n вершинами и m ребрами называется матрица $B = [b_{ij}]$, $i = 1, 2, \dots, n$, $j = 1, 2, \dots, m$, строки которой соответствуют вершинам, а столбцы — ребрам. 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 $B[b_{ij}]$,

$i = 1, 2, \dots, n, j = 1, 2, \dots, m$, строки которой соответствуют вершинам, а столбцы — ребрам. 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 структуру графа. На рис. 6.12 представлена такая матрица для орграфа на рис. 6.10.

$$B = \begin{matrix} & \begin{matrix} \frac{1}{2} & \frac{1}{3} & \frac{2}{1} & \frac{2}{3} & \frac{2}{4} & \frac{2}{5} & \frac{2}{7} & \frac{3}{5} & \frac{4}{1} & \frac{4}{5} & \frac{4}{7} & \frac{6}{4} & \frac{6}{5} & \frac{6}{7} & \frac{7}{4} & \frac{7}{5} \end{matrix} \\ \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{matrix} & \begin{bmatrix} +1 & +1 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & +1 & +1 & +1 & +1 & +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 0 & 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & +1 & +1 & +1 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & +1 & +1 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & +1 & +1 \end{bmatrix} \end{matrix}$$

Рис. 6.12. Матрица инцидентности ориентированного графа

6.2.3. 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 $i, j = 1, 2, \dots, n$. Веса несуществующих ребер полагают равными ∞ или 0 в зависимости от приложений. Заметим, что матрица весов является простым обобщением матрицы смежности.

6.2.4. Список ребер графа

При описании графа списком его ребер каждое ребро представляется парой инцидентных ему вершин. Это представление можно реализовать двумя массивами $r = (r_1, r_2, \dots, r_m)$ и $t = (t_1, t_2, \dots, t_m)$, где m — количество 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. 6.10 будут иметь вид:

$$r = (1, 1, 2, 2, 2, 2, 2, 3, 4, 4, 4, 6, 6, 6, 7, 7),$$

$$t = (2, 3, 1, 3, 4, 5, 7, 5, 1, 5, 7, 4, 5, 7, 4, 5).$$

- *Интересно, что данное представление позволяет легко описать петли и кратные ребра.*

6.2.5. 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 опишем структуру смежности графа, представленного на рис. 6.10.

x	$Adj[x]$
1	2, 3
2	1, 3, 4, 5, 7
3	5
4	1, 5, 7
5	—
6	4, 5, 7
7	4, 5

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.

6.3. Метод поиска в глубину

Один из наиболее естественных способов систематического исследования всех вершин графа исходит из процедуры прохождения графа методом поиска с возвратом, который исследует граф в глубину (см. п.4.1). На неориентированном графе $G = (X, U, \Phi)$ 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 u не пройдена, то заходим в нее и применяем процесс прохождения рекурсивно уже с вершиной u . 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 u , возвращаемся в x даже тогда, когда в u 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 6.1. Рекурсивная процедура $Depth(x, w)$ осуществляет поиск в глубину на графе $G = (X, U, \Phi)$, содержащем $x \in X$, и строит для графа дерево T поиска, которое является ориентированным остовным деревом $G_0 = (X, T, \Phi)$ (если исходный граф не связан, то G_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 G_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 6.1. Поиск в глубину
на простом неориентированном графе*

```

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
            T = T ∪ {(x, v)}; { Включить ребро дерева }
            Depth(v, x);
        else if Mark[v] < Mark[x] and v ≠ w then
            B = B ∪ {(x, v)}; { Включить обратное ребро }
    end;
end.

```

Программная реализация алгоритма 6.1 представлена алгоритмом 6.2. Реализация близко соответствует основному алгоритму 6.1. 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 6.1; *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 ∈ Adj[x] do ...* в программной реализации пред-

ставляется 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 графа $Adj[x]$:

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 6.2 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 6.2 обхода графа, представленного на рис. 6.13. Сплошными линиями отмечены ребра, которые были пройдены во время обхода графа в глубину, пунктирными — обратные ребра.

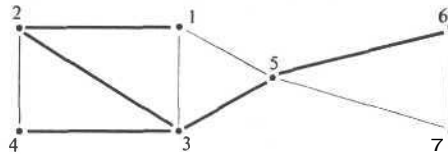


Рис. 6.13. Пример обхода графа в глубину

Исходные данные структуры смежности графа рис. 6.13 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 6.2. Программа поиска в глубину
на простом неориентированном графе*

```
Program PgmWayDepth; {Проход графа в глубину}
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;      { Текстовый файл }
  n      :Integer;   { Количество вершин }
  nT     :Integer;   { Количество ребер прохода в глубину}
  Adj    :TypeAdjacent; { Список смежности графа }
  Fst    :TypeVertex; {Указатели вершин списка смежности }
  Nbr    :TypeVertex; { Количество вершин в списке
                       смежности}
  Vtx    :TypeVertex; { Список вершин графа }
  Mark   :TypeVertex; { Номера компонент для вершин графа}
  T      :TypeVertex; { Последовательность ребер прохода
                       в глубину }
```

```

В      :TypeVertex; { Признаки основных и обратных ребер
                прохода в глубину }

Procedure Init( Var yes :Boolean );
{ Переназначение меток вершин }
{ их порядковыми номерами в списке смежности }
{ yes - признак правильной структуры списка смежности }
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 {Обратное ребро}
        nT:=nT+2; T[nT-1]:=x; T[nT]:=v;
        B[nT div 2]:=0; {Обратное ребро}
      end;
  end;
end;

```



```
Procedure WayDepth; {Проход в глубину}
Var
  v,count :Integer;
begin
  nT:=0; {T - пустое дерево}
  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 {Main}
  i,j :Integer;
  yes :Boolean;
begin {Main}
  Assign(f,'Depth.in');
  Reset(f);{Файл открыт для чтения}
  ( Ввод списка смежности )
  Read(f,n); {Количество строк в списке}
  Fst[1]:=0; {Указатель начала первой строки списка}
  for i:=1 to n do begin
    Read (f,Vtx[i]); {Метка вершины}
    Read(f,Nbr[i]); {Количество вершин в списке}
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]);
      {Список смежных вершин}
    Fst[i+1]:=Fst[i]+Nbr[i]; {Указатель начала
      следующей строки в списке}
  end;
  Close (f);
  Assign(f,'Depth.out');
  Rewrite(f); {Файл открыт для записи}
  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. {Main}
```

6.4. Отношение эквивалентности

Бинарное отношение \sim , определенное на множестве S , называется *отношением эквивалентности*, если оно удовлетворяет свойствам *рефлексивности*, *симметричности* и *транзитивности*:

1. $\forall s_1 \in S \quad s_1 \sim s_1$.
2. $\forall s_1, s_2 \in S \quad s_1 \sim s_2 \rightarrow s_2 \sim s_1$.
3. $\forall s_1, s_2, s_3 \in S \quad s_1 \sim s_2 \wedge s_2 \sim s_3 \rightarrow s_1 \sim s_3$.

Все элементы из множества S , эквивалентные данному элементу s_j , образуют множество S_j , которое называется *классом эквивалентности*. Два различных класса эквивалентности не могут иметь какого-либо общего элемента, в противном случае такие классы совпадают, что следует из свойств 1–3. Таким образом, определенное на множестве S отношение эквивалентности выполняет разложение его на непересекающиеся классы S_i эквивалентности, т.е. $S = \bigcup_i S_i$, где $S_i \cap S_j = \emptyset$, $i \neq j$.

Пример. Пусть S — множество треугольников. Определим на S бинарное \sim отношение. Будем считать, что для треугольников $\Delta a, \Delta b \in S$ выполняется отношение $\Delta a \sim \Delta b$, если они подобные. Ясно, 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 $\vec{a}, \vec{b} \in S$ выполняется отношение $\vec{a} \sim \vec{b}$, если они коллинеарные. Данное отношение является отношением эквивалентности, так как свойства 1–3 выполняются для коллинеарных векторов. Множество векторов под действием введенного отношения разбивается на классы эквивалентности коллинеарных векторов.

Пример. Пусть $S = \{1, 2, \dots, n\}$. 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. Пусть S — множество треугольников. Определим на S бинарное \sim отношение. Будем считать, что для треугольников $\Delta a, \Delta b \in S$ выполняется отношение $\Delta a \sim \Delta b$, если их площади равны. Данное отношение является отношением эквивалентности.

6.5. 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 связаны. 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

$$\Gamma = \bigcup_i \Gamma(X_i, U_i, \Phi)$$

графа $\Gamma = (X, U, \Phi)$ на непересекающиеся связные подграфы $\Gamma = (X_i, U_i, \Phi)$. Вследствие попарного непересечения подграфов, разложение называется *прямым*, а сами подграфы называются *компонентами связности графа Γ* . Таким образом, справедливо следующее утверждение.

- *Утверждение 6.5.1.* Каждый неориентированный граф распадается единственным образом в прямую сумму своих компонент связности.

Количество компонент связности находится в определенном отношении с основными параметрами графа — числом его вершин и ребер.

- *Утверждение 6.5.2.* Пусть $\Gamma = (X, U, \Phi)$ является простым графом с n вершинами и k компонентами связности. Число ребер в таком графе не может превосходить величины $C_{n-k+1}^2 = (n-k+1)(n-k)/2$.

Доказательство. Рассмотрим прямое разложение $\Gamma = \bigcup_{i=1}^k \Gamma(X_i, U_i, \Phi)$ исходного графа на компоненты связности.

Если положить, что число вершин в компоненте X_i связности равно n_i , то число ребер в таком графе не превосходит $\sum_{i=1}^k C_{n_i}^2$. Данная

величина достигается в том случае, когда каждая из компонент связности является полным подграфом. Допустим, что среди компонент связности $\Gamma(X_i, U_i, \Phi)$ найдутся хотя бы две, которые имеют более одной вершины, например $n_2 \geq n_1 > 1$. Перенесем одну вершину из Γ_1 в Γ_2 . Легко видеть, что это увеличивает число ребер в модифицируемом полном 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 $\frac{(n-1)(n-2)}{2}$, связан.

6.6. 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 6.3 выделения компонент связности положена описанная ранее техника поиска в глубину на графе $\Gamma(X, U, \Phi)$. Структура алгоритма 6.3 является модификацией в сторону упрощения основного алгоритма 6.1 поиска в глубину. Работа алгоритма 6.3 направлена на формирование вектора $Mark[x]$ меток вершин $x \in X$ графа. Элементу $Mark[x]$ присваивается общий номер той компоненты, которой принадлежит вершина $x \in X$. Сложность алгоритма 6.3, как и алгоритма 6.1, составляет $O(|X| + |U|)$.

Алгоритм 6.3. Выделение связных компонент неориентированного графа

```

for v ∈ X do Mark[v] = 0; {Начальная установка}
count = 0; {Счетчик числа компонент}
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
    Component(v, count);
end;

```

Программная реализация выделения компонент связности представлена в алгоритме 6.4, который близко соотносится с соответствующим множественным описанием алгоритма 6.3. Рассмотрим пример расчета по программе алгоритма 6.4 выделения компонент связности графа, представленного на рис. 6.14.

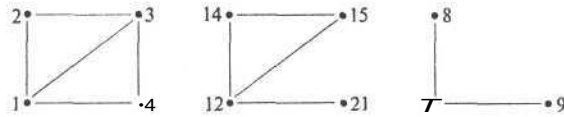


Рис. 6.14. Пример выделения компонент связности графа

Для программы алгоритма 6.4 исходные данные структуры смежности $Adj[x]$ графа на рис. 6.14 задаются в текстовом файле Connect.in. Структура (правило) заполнения файла одинакова с той, которая описана в рассмотренном примере поиска в глубину при расчете по программе алгоритма 6.2.

Данные файла Connect.in для примера на рис. 6.14:

```

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 6.4. Программа выделения связных компонент
неориентированного графа**

```

Program ConnectComponent; {Выделение компонент
                           связности графа}
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;           { Текстовый файл }
  n      :Integer;       { Количество вершин }
  Adj    :TypeAdjacent; { Список смежности графа }
  Fst    :TypeVertex;   { Указатели вершин списка смежности}
  Nbr    :TypeVertex;   { Количество вершин в списке
                           смежности}
  Vtx    :TypeVertex;   { Список вершин графа }
  Mark   :TypeVertex;   { Номера компонент для вершин графа}
Procedure Init( Var yes :Boolean );
{ Переназначение меток вершин }
{ их порядковыми номерами в списке смежности }
{ yes - признак правильной структуры списка смежности }
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;
end;

Procedure Connect; {Выделение компонент связности}
Var
  v,count :Integer;
begin
  for v:=1 to n do Mark[v]:=0;
  count:=0; {Номер компоненты связности}
  for v:=1 to n do begin
    if Mark[v]=0 then begin
      count:=count+1;
      Component(v,count);
    end;
  end;
end;

Var {Main}
  i,j :Integer;
  yes :Boolean;
begin {Main}
  Assign(f,'Connect.in');
  Reset(f);{Файл открыт для чтения}
  { Ввод списка смежности }
  Read(f,n); {Количество строк в списке}
  Fst[1]:=0; {Указатель начала первой строки списка}
  for i:=1 to n do begin
    Read(f,Vtx[i]); { Метка вершины}
    Read(f,Nbr[i]); { Количество вершин в списке}
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]);
      { Список смежных вершин}
    Fst [i+1] :=Fst[i]+Nbr[i];{ Указатель начала следующей
      строки в списке}
  end;
  Close(f);
  Assign(f,'Connect.out');
  Rewrite(f); {Файл открыт для записи}
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа!');
    Close(f);
    exit;
  end;
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. {Main}

```

6.7. Эйлеровы графы

Классической в теории графов является следующая задача. Имеются два острова, соединенных семью мостами с берегами реки и друг с другом, как показано на рис. 6.15. 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.

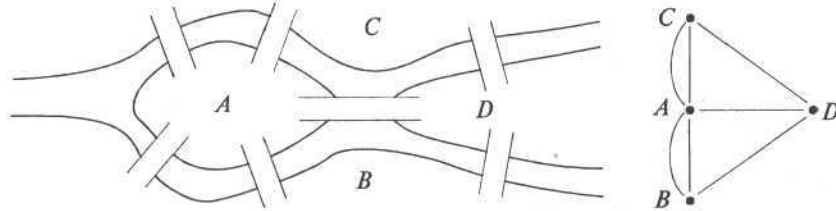


Рис. 6.15. План расположения мостов и соответствующий ему мультиграф

- **Определение.** Пусть $\Gamma = (X, U, \Phi)$ — неориентированный псевдограф. Цепь в Γ называется *эйлеровой*, если она проходит по одному разу через каждое ребро псевдографа Γ . Такой граф называется *эйлеровым*. Замкнутая эйлерова цепь называется *эйлеровым циклом*.

Поставим в соответствие плану расположения суши и мостов, приведенному на рис. 6.15, мультиграф на рис. 6.15, 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.

- **Теорема Л. Эйлера.** Эйлерова цепь в псевдографе $\Gamma = (X, U, \Phi)$ 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.

Доказательство. (\Rightarrow) Дано, что существует эйлерова цепь $au_1x_2u_2\dots x_nu_nb$, 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, если 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 $u_{i-1}x_iu_i$ приносит вершине x_i степень два, а так как все ребра u_i в цепи различные, то степени внутренних вершин четные; 3) и 4) доказательства повторяют доказательство пункта 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. 6.16). Из нее будем строить цепь, выбирая в качестве продолжения пути ребро, которое еще не пройдено. Эта цепь (цикл Γ_1) может закончиться только в вершине A , так как, при входе в любую другую вершину, всегда существует ребро, по которому можно выйти из нее (степени вершин четные).

Возможны два случая: 1) построенный цикл Γ_1 содержит все ребра графа, тогда теорема доказана; 2) Γ_1 содержит не все ребра графа. Во втором случае рассмотрим граф $G \setminus \Gamma_1$, полученный удалением из G всех ребер, входящих в Γ_1 . Граф $G \setminus \Gamma_1$ вновь содержит вершины только с четными степенями (у каждой вершины удалили по четному числу ребер). Так как G — связный граф, то существует вершина в Γ_1 , инцидентная ребру из $G \setminus \Gamma_1$. Пусть это вершина $B \in X$. Построим из нее цикл Γ_2 так же, как строили цикл Γ_1 . Построим общий Γ_{12} цикл из Γ_1 и Γ_2 так, как это сделано на рис. 6.16. Для Γ_{12} вновь проверяем рассмотренные выше два случая, как для Γ_1 .

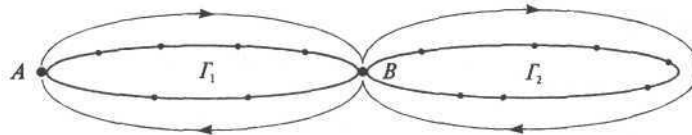


Рис. 6.16. Объединение двух циклов в один цикл

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 $flw_1/2\dots/nw_nf$, где $t_i \in X$, $w_i \in U$.

Конструктивный характер доказательства теоремы позволяет на формальном уровне в алгоритме 6.5 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 6.5. Алгоритм поиска эйлеровой цепи графа

```

 $z \in X$ ; {Вершина начала эйлеровой цепи}
 $Z = \{z\}$ ; {Начало строящейся эйлеровой цепи}
 $R = \emptyset$ ; {Частный расширяющийся цикл эйлеровой цепи}
repeat
   $Cycle(z, R)$ ;
   $Z = Z \cup R$ , {Объединение циклов в один цикл}
until Not  $\exists v \in Z |Adj[v]| > 0$ 
Procedure  $Cycle(z, R)$ 
 $R = \{z\}$ ; {Построение отдельного цикла эйлеровой цепи}
repeat
   $w = Adj[z]$ ;
   $R = R \cup \{w\}$ ;
   $Adj[z] = Adj[z] \setminus \{w\}$ ; {Удалить пройденное ребро  $(z, w)$ }
  if  $v \neq w$  then  $Adj[w] = Adj[w] \setminus \{z\}$ ; {Удалить ребро  $(z, w)$ }
   $v = w$ ;
until Not  $|Adj[z]| > 0$ ; {Пока все ребра не пройдены}
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 $Cycle$ не более, чем число вершин $|X|$. При каждом обращении количество производимых действий пропорционально числу ребер, входящих в выделенный цикл. Сложность суммарной работы процедуры $Cycle$ пропорциона-

льна 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 6.6. Программа поиска эйлеровой цепи графа

```

Program EulerWay; {Эйлерова цепь в псевдографе}
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;           { Текстовый файл }
  ks     :Integer;        { Начальная вершина эйлеровой цепи }
  n      :Integer;        { Количество вершин }
  Adj    :TypeAdjacent;  { Список смежности графа }
  Fst    :TypeVertex;    { Указатели вершин списка смежности }
  Nbr    :TypeVertex;    { Количество вершин в списке
                          смежности }
  Vtx    :TypeVertex;    { Список вершин графа }
  Deg    :TypeVertex;    { Степени вершин графа }
  kz     :Integer;        { Количество вершин в эйлеровой цепи }
  z      :TypeAdjacent;  { Последовательность вершин
                          эйлеровой цепи }
  r      :TypeAdjacent;  { Отдельный расширяющийся цикл }

Procedure Init( Var yes :Boolean );
Var
  i,j,k,m :Integer;
begin
  { Переназначение меток вершин }
  { их порядковыми номерами в списке смежности }
  { yes - признак правильной структуры списка смежности }
  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;
        end
      end
    end
  end

```

```

        break;
    end;
    if not yes then exit;
end;
{ Разместить петли в начале списка смежности }
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;
{ Степени вершин графа }
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; {Петля}
    end;
end;
{ Поиск начальной вершины ks цепи }
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;
    { Граф не эйлеровый }
end;

Procedure Cycle( v :Integer; var count :Integer );
{ Построение частной эйлеровой цепи r[] }
Var
    w      :Integer;
    i,j    :Integer;
begin
    count:=1; r[count]:=v;
    repeat
        w:=Adj[Fst[v]+1]; { Следующая вершина цепи }
        count:=count+1; r[count]:=w;
        { Удалить ребро (v,w) из списка смежности
          для вершины v }
        Fst[v]:=Fst[v]+1;
        Nbr[v]:=Nbr[v]-1;
    until Nbr[v]=0;
end;

```

```
{ Если ребро (w,v) не петля, то удалить и его
из списка для вершины w }
if vow 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 Euler; { Построение эйлеровой цепи z[] }

```
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='); {До объединения}
  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='); {После объединения}
    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 {Main}
  i,j :Integer;
  yes :Boolean;
begin {Main}
  Assign(f,'Euler.in');
  Reset (f);{Файл открыт для чтения}
  { Ввод списка смежности }
  Read(f,n); (Количество строк в списке)
  Fst[1]:=0; {Указатель начала первой строки списка}
  for i:=1 to n do begin
    Read(f,Vtx[i]); {Метка вершины}
    Read(f,Nbr[i]); {Количество вершин в списке}
    for j:=1 to Nbr[i] do Read(f,Adj[Fst[i]+j]);
      {Список смежных вершин}
    Fst[i+1]:=Fst[i]+Nbr[i];{Указатель начала следующей
      строки в списке}
  end;
  Close(f);
  Assign(f,'Euler.out');
  Rewrite(f); {Файл открыт для записи}
  Init(yes);
  if not yes then begin
    WriteLn(f,'Плохая структура смежности графа');
    WriteLn(f,' или граф не эйлеровый!');
    Close(f);
    exit;
  end;
  Euler;
  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 6.6 поиска эйлеровой цепи в графе, изображенного на рис. 6.17.

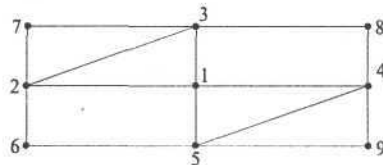


Рис. 6.17. Пример расчета эйлеровой цепи графа

Для программы алгоритма 6.6 исходные данные структуры смежности $Adj[x]$ графа на рис. 6.17 задаются в текстовом файле `Euler.in`. Структура (правило) заполнения файла совпадает с той, которая описана в рассмотренном примере поиска в глубину при расчете по программе алгоритма 6.2.

Данные файла `Euler.in` для примера на рис. 6.17:

```

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 `Euler.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 знаком O в начале строки.

6.8. Остовные деревья

- *Определение.* Остовным деревом связного неориентированного графа $\Gamma = (X, U, \Phi)$ называется дерево $\Gamma_0 = (X, U_0, \Phi)$, являющееся подграфом графа Γ и содержащее все его вершины (рис. 6.18).



Рис. 6.18. Связный граф и два остовных дерева

- **Утверждение 6.8.1.** Дерево с n вершинами содержит $n - 1$ ребро.

Доказательство. Доказательство проведем по индукции числа вершин. Заметим, что в дереве найдется вершина, степень которой равна единице (висячая вершина). Действительно, в противном случае, если степени вершин > 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 > 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 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 $G = (X, U, \Phi)$, 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 $G_0 = (X, U_0, \Phi)$, содержащий все вершины, с минимальным весом $W(G_0) = \sum_{u \in U_0} \omega(u)$. Очевидно, что G_0 — дерево, в противном случае можно было бы удалить одно ребро, не нарушая связности G_0 и уменьшая сумму весов его ребер.

- **Определение.** Минимальным остовным деревом (лесом) называется остовное дерево (лес) с минимальным общим весом его ребер.

6.8.1. Жадный алгоритм построения минимального остовного дерева

Минимум остовных деревьев графа $G = (X, U, \Phi)$ 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 $X = \{x_1, x_2, \dots, x_n\}$ графа $G = (X, U, \Phi)$ формируются начальные тривиальные компоненты связности $T_i = (X_i, U_i, \Phi)$, где $X_i = \{x_i\}$, $U_i = \emptyset$, $X_i \cap X_j = \emptyset$, $i \neq j$, $X = \bigcup_{i=1}^{|X|} X_i$, $i = 1, 2, \dots, |X|$. Компоненты T_i являются деревьями, объединение $T = \bigcup_i T_i$ которых дает начальное приближение строящегося остовного дерева $G_0 = (X, U_0, \Phi)$.

Включение в строящееся остовное дерево G_0 выбранного ребра на очередном шаге *жадного алгоритма* выполняется слиянием T_i и T_j ($X_i = X_j \cup X_i$ и $U_i = U_j \cup U_i$) 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 $T = \bigcup_i T_i$ компонент к остовному дереву $G_0 = (X, U_0, \Phi)$

продолжаем до тех пор, пока не будет включено $|X| - 1$ ребро.

Справедливость *жадного алгоритма* является следствием следующих двух лемм.

- **Лемма 6.8.1.** Пусть $G = (X, U, \Phi)$ — связный неориентированный граф и $G_0 = (X, U_0, \Phi)$ — произвольное остовное дерево для него. Тогда:
 - 1) $\forall x_1, x_2 \in X$ существует единственная между ними цепь в G_0 ;
 - 2) если к G_0 добавить ребро из $U \setminus U_0$, то возникнет ровно один цикл.

Доказательство. Утверждение 1 верно, т.к. в противном случае в G_0 существовал бы цикл, что противоречит дереву G_0 . Утверждение 2 верно, поскольку между вершинами добавляемого ребра уже есть одна цепь, а значит, возникнет один цикл.

- Лемма 6.8.2.** 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 $T = \bigcup T_i$, согласно алгоритму, растет к остовному дереву $\Gamma_0 = (X, U_0, \Phi)$, где $i = 1, 2, \dots, k$ и $k > 1$. Пусть следующим найденным для включения ребром является ребро $E = (x, y)$ наименьшего веса из оставшихся $U \setminus U_i$ и пусть $x \in X_1$ и $y \notin X_1$. Тогда найдется остовное дерево Γ_0 для $\Gamma = (X, U, \Phi)$, содержащее ребра $\bigcup U_i \cup \{E\}$, вес которого не больше любого другого остовного дерева, содержащего ребра U_i .

Доказательство. Допустим противное. Пусть существует остовное дерево $\Gamma'_0 = (X, U'_0, \Phi)$ для Γ , содержащее $\bigcup U_i$ и не содержащее ребра E , вес которого меньше веса любого остовного дерева для Γ , содержащего $U_i \cup \{E\}$. По утверждению 1 леммы 6.8.1, при

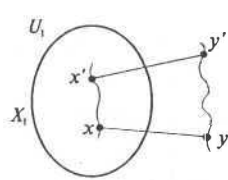


Рис. 6.19

добавлении E к Γ'_0 образуется цикл. Этот цикл должен содержать такое ребро $\epsilon' = (x', y')$, отличное от E , что $x' \in X_1$ и $y' \notin X_1$, т.к. цикл входит в U_1 по ребру (x, y) и $y \notin X_1$, то он должен и выйти из U_1 (рис. 6.19). Из условия следует, что вес $\omega(\epsilon) < \omega(\epsilon')$, так как $\bigcup U_i \cup U_0$ и $\bigcup U_i \cup U'_0$

ребро E выбиралось с минимальным весом из оставшихся ребер $U \setminus U_j$ без образования циклов, в противном же случае выбор должен был бы пасть на E' , т.к. оно тоже не образует циклов с U_i .

Рассмотрим граф $\Gamma_0 = (X, U_0, \Phi)$, образованный добавлением E к Γ'_0 и удалением E' из Γ'_0 . В Γ_0 нет циклов, так как единственный цикл разорван удалением ребра E' . Γ_0 остался связным, так как осталась цепь между x' и y' . Таким образом, Γ_0 — остовное дерево. Учитывая, что $\omega(E) < \omega(E')$, то вес дерева Γ_0 не больше веса Γ'_0 . Остовное дерево Γ_0 содержит $\bigcup U_i$ и E , а это противоречит предположению минимальности Γ'_0 , что доказывает справедливость жадного алгоритма.

Реализация «жадной» схемы формирования остовного дерева представлена в алгоритме 6.7. Используемые при его описании обозначения соответствуют тому, что вводилось при обосновании жадной схемы. Вектор $Mark[x]$ меток вершин $x \in X$ графа поддерживает их принадлежность компонентам связности U_i , из которых формируется реберный список остовного дерева. Начальные величины $Mark[x_i]$ устанавливаются равными порядковым номерам соответствующих вершин $x_i \in X$. Далее значения $Mark[x_i]$ 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 6.7. Жадный алгоритм минимального остовного дерева

```

for  $x_i \in X$  do  $Mark[x_i] = i$ ;
Sort( $U$ ); {Сортировка списка ребер по их весам}
 $U_0 = \emptyset$ ;
while  $|U_0| < |X| - 1$  do begin
  ( $x, y$ )  $\in U$ ; {Ребро с min весом из оставшихся}
  if  $Mark[x] \neq Mark[y]$  then begin
     $U_0 = U_0 \cup \{(x, y)\}$ ; {Включить ребро в остовное дерево}
     $Z = Mark[y]$ ; {Слияние  $U_x$  и  $U_y$ }
    for  $v \in X$  do if  $Mark[v] = z$  then
       $Mark[v] = Mark[x]$ ;
  end;
   $U = U \setminus \{(x, y)\}$ ; {Удалить ребро с min весом из списка}
end;
```

- **Сложность жадного алгоритма.** Жадный алгоритм требует предварительной сортировки ребер по их весам. Стандартные методы сортировки имеют сложность $O(|U|^2)$. Сложность алгоритма 6.7 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 6.7) составляет $O(|X|^2)$. Значит, сложность жадного алгоритма $O(|X|^2) + |U|^2$.

Программная реализация жадного алгоритма представлена в алгоритме 6.8, который близко соответствует множественному описанию соответствующего алгоритма 6.7.

Алгоритм 6.8. Программа жадного алгоритма

```

Program HungryOstov; {Остовное дерево. Жадный алгоритм}
uses CRT, DOS;
Const
  nVertex=50; {Максимальное количество вершин}
  nRib=1000; {Максимальная количество ребер}
Type
  TypeVertex=array[1..nVertex] of Integer;
  TypeRib=array[1..nRib] of Integer;
Var
  f      :Text;      ( Текстовый файл )
  nX     :Integer;   { Количество вершин в графе }
  nU     :Integer;   { Количество ребер в графе }
  Mark   :TypeVertex; { Метки принадлежности вершин }
  X      :TypeVertex; { Список вершин графа }
  U      :TypeRib;   { Реберный список графа }
  nUo    :Integer;   { Количество ребер в остовном дереве }
  Uo     :TypeRib;   { Ребра остовного дерева }
  We     :TypeRib;   { Веса ребер графа }
  Wt     :LongInt;   { Вес минимального остовного дерева }
Procedure Init; { Переименование меток вершин }
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; { Сортировка списка ребер по их весам }
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; { Строим минимальное остовное дерево }
Var
    i,x,y,z : Integer;
    sU      : Integer;
begin
    for i:=1 to nX do Mark[i]:=i;
    Sort; {Сортировка ребер по весу}
    nUo:=0; {Пустое множество Uo}
    sU:=1; {Начальное ребро в сортированном U}
    while nUo<nX-1 do begin
        x:=U[2*sU]; {Выбор нового ребра из списка}
        y:=U[2*sU-1];
        if Mark[x]<>Mark[y] then begin
            nUo:=nUo+1;
            Uo[nUo]:=sU; {Добавить ребро в остовное дерево}
            z:=Mark[y]; {Слияние Ux и Uy}
            for i:=1 to nX do if Mark[i]=z then
                Mark[i]:=Mark[x];
            end;
            sU:=sU+1; {Удалить ребро (x,y) из списка U}
        end;
    end;
end;
Var {Main}
    i,j : Integer;
begin {Main}
    Assign(f, 'Hungry.in');
    Reset(f); {Файл открыт для чтения}
    Read(f,nU); {Количество ребер в реберном списке графа}
    for i:=1 to nU do Read(f, U[2*i-1]); { Первые вершины
        ребер }
    for i:=1 to nU do Read(f, U[2*i]); { Вторые вершины
        ребер}
    for i:=1 to nU do Read(f,We[i]); { Веса ребер }
    Close(f);
    Assign(f, 'Hungry.out');
    Rewrite(f); {Файл открыт для записи}
    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,'uol='');
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. {Main}

```

Рассмотрим пример построения минимального остовного дерева графа, изображенного на рис. 6.20, по программе алгоритма 6.8.

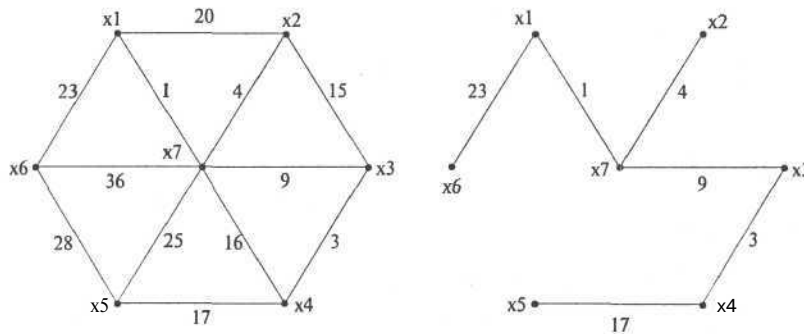


Рис. 6. 20. Пример расчета остовного дерева

Для программы этого алгоритма исходные данные графа на рис. 6.20 задаются реберным списком в текстовом файле `Hungу.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
пX = 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;
- пX — число вершин в графе;
- X — список вершин графа;
- (u1, u2) — сортированный список ребер графа;
- We — веса ребер согласно их сортировке;
- (uo1, uo2) — ребра остовного дерева;
- Woe — веса ребер остовного дерева;
- Вес — сумма весов ребер остовного дерева.

6.8.2. 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 Γ_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 Γ_0 .
3. Повторяя процесс, выполняем поиск наименьшего по весу ребра, соединяющего вершины x_1 и x_2 с некоторой другой вершиной графа x_3 .

4. Процесс включения ребер продолжаем до тех пор, пока все вершины исходного графа Γ не будут включены в дерево Γ_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 6.9, 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 $X = \{1, 2, \dots, n\}$. 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.

- *Сложность алгоритма ближайшего соседа.* Сложность алгоритма определяется двумя вложенными циклами по числу вершин. В каждом из циклов выполняется константное число операций. Следовательно, сложность составляет $O(|X|^2)$.

Алгоритм 6.9. Алгоритм ближайшего соседа для остовного дерева

```

X = {1, 2, ..., n}; {Метки вершин графа}
nX = |X|; {Количество вершин в графе}
v = rand(1, |X|); {Произвольная вершина v ∈ X}
X0 = ∅; {Вершины остовного дерева}
X = X \ {v}; {Удалить v из X}
U0 = ∅; {Ребра остовного дерева}
WT = 0; {Вес остовного дерева}
for x ∈ X do begin
  dist[x] = We[x, v]; {Минимальное расстояние от x до v ∈ X0}
  prev[x] = v; {Ребро (v, x), длина которого dist[x]}
end;
while |X0| ≠ nX do begin
  dist[v] = minx ∈ X dist[x]; {v ∈ X — вершина для включения}

```



```

 $X_0 = X_0 \cup \{v\}$ ; {Добавить вершину}
 $X = X \setminus \{v\}$ ; {Удалить вершину}
 $U_0 = U_0 \cup \{prev[v], v\}$ ; {Добавить ребро}
 $W_T = W_T + dist[v]$ ; {Поправить вес дерева}
{Поправить  $dist[x]$  — вектор расстояний}
for  $x \in X$  do if  $dist[x] > We[v, x]$  then begin
     $dist[x] = We[v, x]$ ;
     $prev[x] = v$ ;
end,
end.

```

Программная реализация алгоритма ближайшего соседа представлена в алгоритме 6.10, который близко соответствует множественному описанию соответствующего алгоритма 6.9.

Алгоритм 6.10. Программа алгоритма ближайшего соседа

```

Program NearOstov; {Остовное дерево.
                    Метод ближайшего соседа}
uses CRT, DOS;
Const
    nVertex=50; {Максимальное количество вершин}
    nRib=1000; {Максимальное количество ребер}
Type
    TypeVertex=array[1..nVertex] of Integer;
    TypeRib=array[1..nRib] of Integer;
    TypeWeight=array[1..nVertex,1..nVertex] of Integer;
Var
    f :Text; {Текстовый файл}
    nX :Integer; {Количество вершин в графе}
    nXo :Integer; {Количество вершин в остовном дереве}
    nUo :Integer; {Количество ребер в остовном дереве}
    X :TypeVertex; {Список вершин графа}
    Xo :TypeVertex; {Список вершин остовного дерева}
    Uo :TypeRib; {Ребра остовного дерева}
    Prev :TypeVertex; {Список ближайших вершин}
    Dist :TypeRib; {Расстояния до ближайших вершин}
    We :TypeWeight; {Матрица весов ребер графа}
    Wt :LongInt; {Вес минимального остовного дерева}
Procedure minDist(Var v :Integer);{Поиск v - ближайшего
                                   соседа}
Var i,d :Integer;
begin
    v:=X[1];

```



```

    Uo[2*nUo]:=v;
    Wt:=Wt+Dist[v]; { Обновить вес остовного дерева }
    newDist(v);
end;
end;

Var {Main}
    i,j :Integer;
begin {Main}
    Assign(f,'Near.in');
    Reset(f);{Файл открыт для чтения}
    Read(f,nX); {Количество вершин в графе}
    for i:=1 to nX do begin
        for j :=i to nX do begin
            Read(f,We[i,j]); { Ввод матрицы весов }
            if We[i,j]=0 then We[i,j]:=$7fff; {+бесконечность}
            We[j,i]:=We[i,j];
        end;
    end;
    Close(f);
    Assign(f,'Near.out');
    Rewrite(f); {Файл открыт для записи}
    nUo:=0;{Выделение реберного списка графа для печати}
    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,'uol=');
    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, 'Вес=' , Wt:3);
Close(f);
end. {Main}

```

Рассмотрим пример расчета по программе алгоритма 6.10 построения минимального остовного дерева графа, изображенного на рис. 6.20. 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 n 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
u 2= 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
Вес= 57.

```

Обозначения данных в файле `Near.out` соответствуют принятым обозначениям в файле `Hungry.out` при контрольном расчете остовного дерева по жадному алгоритму 6.8 (см. п.6.8.1).

6.9. 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 $G = (X, U, \Phi)$ — простой орграф, для каждого ребра $u \in U$ определен вес $\omega(u) > 0$. Найдем кратчайший путь между выделенными вершинами x_0 и z (рис. 6.21). Несуществующие ребра будем считать ребрами с бесконечными весами. Сумму весов ребер в пути будем называть весом или длиной пути. 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 временные и окончательные. Временная 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 \bar{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 $|X| - 1$ вершин присваивается временная метка да (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 $(w_{ij} + \text{окончательная метка } x_i)$, 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, представленном на рис. 6.21.

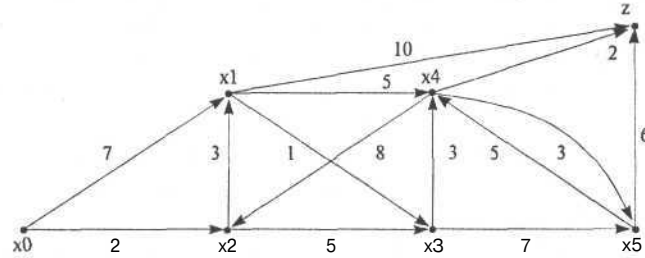


Рис. 6.21. Простой взвешенный орграф

Процесс назначения меток вершинам графа на каждом шаге удобно представить в виде следующей таблицы.

	x_0	x_1	x_2	x_3	x_4	x_5	z
0	0	∞	∞	∞	∞	∞	∞
1		7	2	∞	∞	∞	∞
2		5		7	∞	∞	∞
3				6	10	∞	15
4					9	13	15
5						12	11

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 6.11, 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.

Алгоритм 6.11. Алгоритм кратчайшего пути на орграфе

```

for  $x \in X$  do begin
   $Mark[x] = FALSE$ ;
   $Dist[x] = \infty$ ;
end;
 $y = x_0$ ;
 $Mark[x_0] = TRUE$ ;
 $Dist[x_0] = 0$ ;
while not  $Mark[z]$  do begin
  for  $x \in X$  do
    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;
    {Поиск новой вершины  $y \in X$  с минимальной временной меткой}
     $dist[y] = \min_{x \in X \text{ and } Mark[x]=FALSE} dist[x]$ ;
     $Mark[y] = TRUE$ ;
  end.

```

$Prev[x]$ указывает на вершину с окончательной меткой, ближайшую к вершине x . Последовательность вершин кратчайшего пути будет иметь следующий вид:

$$z, prev[z], prev[prev[z]], prev[prev[prev[z]]], \dots, x_0,$$

а значение $Dist[z]$ составит длину пути из x_0 в z . Очередная новая вершина, претендующая на постоянную метку, обозначается через y .

- *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 6.11 условие цикла *while not Mark[z] do begin* надо заменить на условие *while $\vee_{x \in X} \text{not } Mark[x]$ do begin*. При этом сложность алгоритма останется прежней.

Программная реализация алгоритма поиска кратчайшего пути представлена в алгоритме 6.12 на Pascal'e, который близко соответствует множественному описанию алгоритма 6.11.

Алгоритм 6.12. Программа кратчайшего пути на орграфе

```

Program Short; {Кратчайшие пути на графе}
uses CRT,DOS;
Const
  nVertex=50; {Максимальное количество вершин}
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;           { Текстовый файл }
  nX     :Integer;       { Количество вершин в графе }
  Mark   :TypeMark;     {Признаки временных и постоянных меток}
  Dist   :TypeDist;     { Значения текущих меток вершин
                        { (расстояния)}
  Prev   :TypePrev;     { Указатель на ближайшую вершину }
  We     :TypeWeight;   { Матрица весов ребер графа }
  x0     :Integer;      { Вершина начала пути }
  z      :Integer;      { Вершина конца пути }
  y      :Integer;      { Последняя вершина с постоянной меткой}

Var
  i,j,x  :Integer;
  weight :LongInt;

begin
  Assign(f,' Short.in');
  Reset(f);{Файл открыт для чтения}
  {Ввод исходных данных}
  Read(f,x0); {Начальная вершина пути}
  Read(f,z); {Конечная вершина пути}
  Read(f,nX); {Количество вершин в графе}
  nX:=nX-1; (* X={0,1,2,...,nX} - множество вершин *)
  for i:=0 to nX do begin
    for j:=0 to nX do begin
      Read(f,We[i,j]); { Ввод матрицы весов }
      if We[i,j]=0 then We[i,j]:=$7fff; {+бесконечность}
    end;
  end;
  Close(f);
  Assign(f,' Short.out');
  Rewrite(f); {Файл открыт для записи}

  for x:=0 to nX do begin
    Mark[x]:=FALSE;
    Dist[x]:=$7fffffff;
  end;

```



```

y:=x0; {Последняя вершина с постоянной меткой}
Mark[y]:=TRUE;
Dist[y]:=0;
while not Mark[z] do begin
  {Обновить временные метки}
  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;
  {Поиск вершины с минимальной временной меткой}
  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 6.12 поиска кратчайшего пути на графе, показанном на рис. 6.21. Исходные данные графа представляются матрицей весов его ребер в текстовом файле `Short.in` со следующей структурой:

- в первой строке определяется номер начальной вершины пути x_0 ;
- во второй строке определяется номер конечной вершины пути z ;
- 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	5	2
0	0	0	0	1	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.

6.10. Потoki в сетях

- Определение.* Транспортной сетью называется связный ориентированный граф без петель $\Gamma = (X, U, \Phi)$ с выделенной парой вершин x_0 и z (рис. 6.22). 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 $c(u) > 0$, где $c(u)$ — пропускная способность дуги.

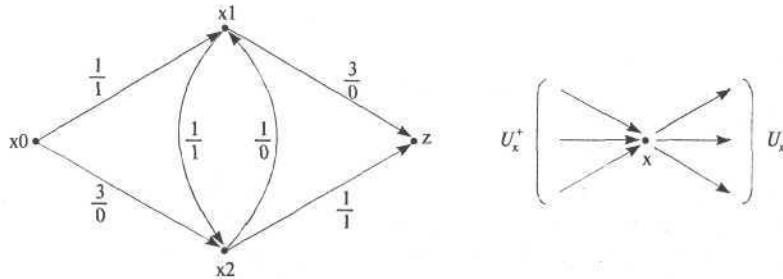


Рис. 6.22. Транспортная сеть

- Определение.* Потокom по транспортной сети называется целочисленная функция $\varphi(u) > 0$, заданная на множестве дуг $u \in U$ и обладающая следующими свойствами:

$$\forall u \in U \varphi(u) \leq c(u) \text{ и } \sum_{u \in U_x^+} \varphi(u) = \sum_{u \in U_x^-} \varphi(u), \quad (6.10.1)$$

где x — внутренняя вершина графа, т.е. $x \neq x_0$, $x \neq z$;

U_x^+ — множество дуг, заходящих в вершину x ;

U_x^- — множество дуг, выходящих из вершины x (рис. 6.22).

На рис. 6.22 напротив каждой дуги стоит дробь, числитель которой — пропускная способность дуги, знаменатель — поток по дуге. Свойство (6.10.1) утверждает, что поток, входящий в вершину, равен выходящему потоку (поток в вершинах не скапливается). Обозначим

$$\varphi(z) = \sum_{u \in U_z^+} \varphi(u) \quad \text{и} \quad \varphi(x_0) = \sum_{u \in U_{x_0}^-} \varphi(u),$$

где $\varphi(z)$ — поток, входящий в вершину z ;

$\varphi(x_0)$ — поток, выходящий из вершины x_0 .

- *Утверждение 6.10.1.* $\varphi(z) = \varphi(x_0)$.

Действительно, сумма $\sum_{x \in X} [\sum_{u \in U_x^+} \varphi(u) - \sum_{u \in U_x^-} \varphi(u)] = 0$, так как $\forall u \in U$

величина $\varphi(u)$ суммируется дважды — со знаками + и -. Здесь

$$\sum_{x \in X} [\sum_{u \in U_x^+} \varphi(u) - \sum_{u \in U_x^-} \varphi(u)] = \sum_{x \in X \setminus \{x_0, z\}} [\sum_{u \in U_x^+} \varphi(u) - \sum_{u \in U_x^-} \varphi(u)] + \varphi(z) - \varphi(x_0) = 0,$$

где первая часть выражения равна нулю вследствие 6.10.1.

- *Определение разреза.* 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.

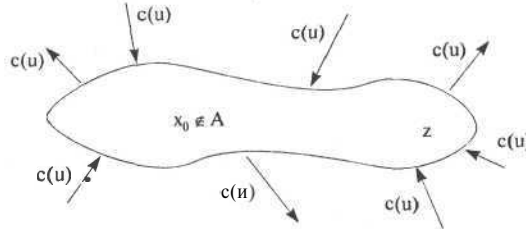


Рис. 6.23. Разрез транспортной сети

$C(A) = \sum_{u \in U_A^+} c(u)$ — называется мощностью разреза. Это максимально возможный поток, входящий в A по дугам разреза.

$\varphi(A) = \sum_{u \in U_A^+} \varphi(u)$ — поток, входящий в A по дугам разреза. Ясно, что $\varphi(A) < C(A)$, так как $\forall u \in U \varphi(u) < c(u)$.

- *Утверждение 6.10.2.* $\varphi(z) < C(A)$.

Действительно, из рис. 6.23 видно, что не весь поток, входящий в A , скатывается в z . Часть потока может выходить из A . Значит, $\varphi(z) < \varphi(A)$, но $\varphi(A) < C(A)$, тогда и $\varphi(z) < C(A)$.

- **Теорема 6.10.1 Форда и Фалкерсона.** Максимальный поток по транспортной сети равен мощности минимального разреза, т.е.

$$\max_{\varphi} \varphi(z) = \min_{\varphi} C(A).$$

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 алгоритм 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. 6.24. Пометим рекурсивным образом все возможные вершины x_i сети.

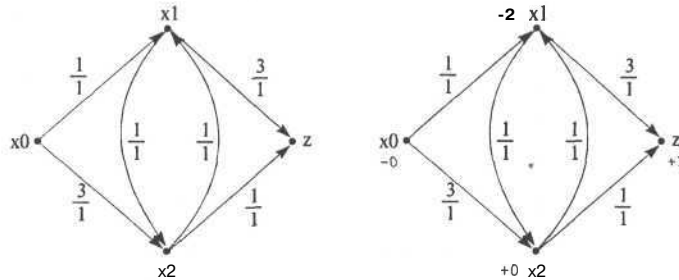


Рис. 6.24. Насыщенная транспортная сеть и пометка вершин

- 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 $x_i \rightarrow y(+i)$, 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 рис. 6.25. 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. 6.25.

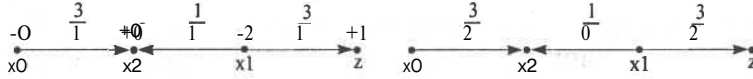


Рис. 6.25. Перераспределение потока на основе пометки вершин

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. 6.26. 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 φ^* и разрез A^* , для которых выполняется $\varphi^*(A^*) = C(A^*)$. Учитывая, что выходящие дуги из A^* пустые, то весь поток $\varphi^*(A^*)$ скатывается в z , т.е. $\varphi^*(z) = C(A^*)$.

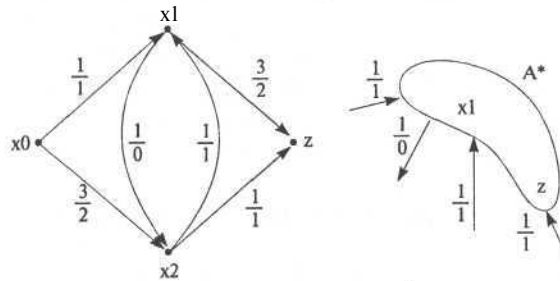


Рис. 6.26. Максимальный поток

- *Определение независимого множества вершин.* Пусть граф $G = (X, U, \Phi)$ — неориентированный и без петель. Множество $I \subset U$ вершин называется *независимым*, если между любыми его вершинами нет соединяющих ребер. В *зависимом множестве* хотя бы две вершины соединены ребром. Множество Q *полностью зависимо*, 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 $\beta(G)$ вершин, составляющих независимое множество, называется *числом (вершинной) независимости графа*.
- *Определение независимого множества ребер.* 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* $\beta_e(G)$.

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 $\beta(G) = 8$.

- **Утверждение 6.11.1.** Независимое множество максимально тогда и только тогда, когда оно доминирующее, а значит, $\beta(G) > \delta(I)$ — число (вершинной) независимости не может быть меньше числа доминирования.

Доказательство. (\Rightarrow) Если $I \subset U$ — максимальное независимое множество, то не может быть вершин $k \in \bar{I}$, не соединенных с 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 \bar{I} в I так, чтобы $k \cup I$ 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. 6.31 показан граф и его клики.*

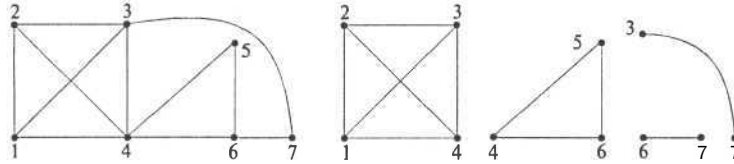


Рис. 6.31. Граф и его клики

- **Замечание.** Если предположить, что граф $G = (X, U, \Phi)$ простой, то полностью зависимые множества (клики) в G становятся максимально независимыми множествами в дополнительном графе \bar{G} , верно и обратное.

При алгоритмическом подходе к выделению клик в графе применяют *метод поиска с возвратением* 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 графа. Вершины (множества) дерева поиска определим рекурсивно. Корень дерева поиска — пустое начальное множество $S = \emptyset$. 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. 6.32 показаны некоторый граф G и дерево поиска T , которое исследуется в процессе поиска с возвратами клик графа полным перебором.

Заметим, что каждая клика порождается много раз: клика $\{1,2,3\}$ порождается $3!$ раз, клики $\{3,4\}$ $\{4,5\}$ порождаются $2!$ раз. В общем случае клика размера k порождается $k!$ раз. Все тонкие ребра на рис. 6.32 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.

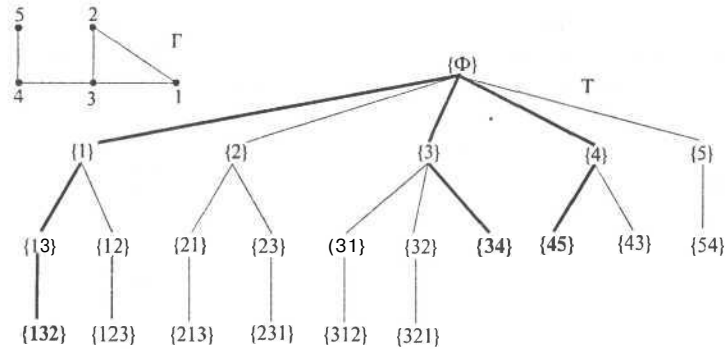
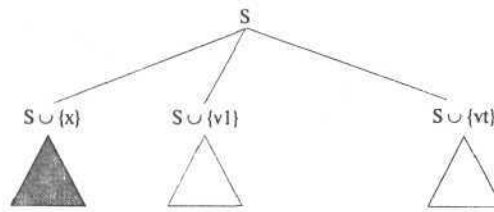


Рис. 6.32. Граф G и полный перебор дерева T поиска клик

- Утверждение 6.11.2.** Пусть $G = (X, U, \Phi)$ — 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 множество $S \cup \{x\}$, как на рис. 6.33, 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 $v \notin Adj[x]$ (рис. 6.33).

Рис. 6.33
Поддеревья
с корнями $S \cup \{v_i\}$
(эти вершины
смежны с x)



- Утверждение 6.11.3.** Пусть S — узел в дереве поиска T , и пусть $\tilde{S} \subset S$ — предок S в T . Если все поддеревья узла $\tilde{S} \cup \{x\}$ уже исследованы, так что порождены все клики, включающие $\tilde{S} \cup \{x\}$, то все неисследованные поддеревья с корнями $S \cup \{x\}$ можно игнорировать.

Алгоритм 6.13 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 S

поиска 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.

$N = \{x \in X \mid (x, s) \in U \forall s \in S \wedge \forall \tilde{S} \subset S \text{ ни одно из поддеревьев } \tilde{S} \cup \{x\} \text{ не исследовано}\}.$

Алгоритм 6.13. Порождение клик графа

```

S = ∅; {Начальная вершина дерева поиска T}
N = X;
D = ∅;
Z = X;
CLIQUE(N, D, Z); {Рекурсивный проход по дереву поиска T}
Procedure CLIQUE(N, D, Z)
  if N ∩ D = ∅ then вывести S, которое является кликой
  else if N ≠ ∅ then begin
    x ∈ N; {Исследовать первое поддерево}
    View(x); {Исследовать поддерева}
    {Исследовать оставшиеся поддерева уровня}
    Z = Z \ {x}; Z = Z \ Adj[x];
    while Z ≠ ∅ do begin
      v ∈ Z;
      View(v); {Исследовать поддерева}
      Z = Z \ {v};
    end;
  end;
end;
Procedure View(v) {Исследовать поддерева}
  N = N \ {v};
  S = S ∪ {v};
  CLIQUE(N ∩ Adj[v], D ∩ Adj[v], N ∩ Adj[v]);
  S = S \ {v};
  D = D ∪ {v};
end;

```

Процедура *CLIQUE* выбирает произвольную вершину $x \in N$, удаляет ее из N и исследует поддерево $S = S \cup \{x\}$, обращаясь к процедуре *View*. Далее, согласно утверждениям 6.11.2 и 6.11.3, при помощи процедуры исследуются только поддерева $S = S \cup \{v\}$, где $v \in N$, $v \notin \text{Adj}[x]$ 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.

$D = \{x \in X \mid (x, s) \in U \forall s \in S \text{ и поддереву } \bar{S} \cup \{x\} \text{ исследовалось для некоторых } \bar{S} \subseteq S\}$.

По алгоритму множество S с X является полным подграфом графа $G = (X, U, \Phi)$ и $N \cup D$ — множество всех вершин, смежных с каждой вершиной в S .

- Множество S будет кликой тогда и только тогда, когда $N \cup D = \emptyset$.
- Условие $N = 0$ и $D \neq 0$ обозначает, что все клики, включающие S , уже ранее порождались.
- При $N \neq 0$ могут оставаться клики, включающие S , которые еще не порождались. Исследование таких поддеревьев $S \cup \{x\}$, $x \in N$ необходимо продолжить.

Основные усилия алгоритма 6.13, 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 6.14 на Pascal'e, который близко соответствует множественному описанию алгоритма 6.13. 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 6.14. Программа порождения клик графа

```

Program PgmClique; {Выделение клик графа}
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;          { Текстовый файл }
  m      :Integer;      { Количество вершин в графе }
  Adj    :TypeAdjacent; { Список смежности графа }
  Fst    :TypeVertex;  { Указатели вершин списка смежности }
  Nbr    :TypeVertex;  { Количество вершин в списке
                        смежности }
  Vtx    :TypeVertex;  { Список вершин графа }
  S      :TypeVertex;  { Список вершин формируемых клик }
  nS     :Integer;     { Количество вершин в списке S }
  N      :TypeVertex;  { Список включаемых вершин
                        при поиске клик }
{kN,nN - указатель начала и количества вершин в списке N}
  D      :TypeVertex;  {Список вершин с ранее найденными
                        кликами}
{kD,nD - указатель начала и количества вершин в списке D}
  Z      :TypeVertex;  { Список не исследованных вершин }
{kZ,nZ - указатель начала и количества вершин в списке Z}

Procedure PrintClique; FORWARD;
Procedure Subtract( x:Integer; Var kZ,nZ:Integer ); FOR-
WARD;
Procedure InterSection( v,kM,nM:Integer; Var kMw,nMw:In-
teger; Var M:TypeVertex ); FORWARD;
Procedure View( v:Integer; Var kN,nN, kD,nD, kZ,nZ:In-
teger ); FORWARD;
Procedure Clique( kN,nN,kD,nD,kZ,nZ:Integer ); FORWARD;

Procedure Init( Var yes :Boolean );
{ Переименование меток вершин их порядковыми номерами
  по списку смежности вершин графа }
( yes - признак правильной структуры списка смежности
  графа)
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;
    end;

```

```

        if not yes then exit;
    end;
end;
Procedure PrintClique;
( Печать вершин найденной клики )
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 );
{ Формирование пересечения  $M \cap Adj[v]$  множеств M и  $Adj[v]$  }
Var
    i,j,k :Integer;
    yes :Boolean;
begin
    {kMw - указатель начала вершин в множестве  $M \cap Adj[v]$ }
    {nMw - количество вершин в множестве  $M \cap Adj[v]$ }
    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;
    end;
end;

Procedure Subtract( x:Integer; Var kZ,nZ:Integer );
{(* Формирование разности множеств  $Z=Z \setminus \{x\}$  и  $Z=Z \setminus Adj[x]$  *)}
Var
    i,j,k :Integer;
    yes :Boolean;
begin
    {(* Формирование  $Z=Z \setminus \{x\}$  *)}
    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;
    end;
    {(* Формирование  $Z=Z \setminus Adj[x]$  *)}

```

```

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;
end;

Procedure Clique( kN,nN,kD,nD,kZ,nZ:Integer );
{ Процедура рекурсивного поиска клик графа }
Var
  i,j,x :Integer;
begin
  if (nN=0) and (nD=0) then {Клика найдена}
    PrintClique {Печать клики}
  else if nN<>0 then begin {Продолжить исследование
    поддеревьев}
    x:=N[kN+nN]; {Перебор с конца вершин множества N}
    View(x,kN,nN,kD,nD,kZ,nZ); {Исследовать первое
      поддерево уровня}
    Subtract(x,kZ,nZ); (* Z=Z\{x} и Z=Z\Adj[x] *)
    while nZ<>0 do begin {Исследовать следующие
      поддерева уровня}
      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);
{ Исследование поддеревьев следующего уровня } -
Var
  i,k :Integer;
  kNw,nNw :Integer;
  kDw,nDw :Integer;
  kZw,nZw :Integer;
begin
  (* Формирование 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;
  end;
end;

```

```

(* Формирование S=S+{v} *)
nS:=nS+1;
S[nS]:=v;
(* Формирование пересечения множеств *)
InterSection(v, kN, nN, kNw, nNw, N); {N и Adj[v]}
InterSection(v, kD, nD, kDw, nDw, D); {D и Adj[v]}
(* Формирование начального Z=N нового уровня *)
kZw:=kZ+nZ;
nZw:=nNw;
for i:=1 to nNw do Z[kZw+i]:=N[kNw+i];
(* Исследование поддеревьев нового уровня *)
Clique(kNw, nNw, kDw, nDw, kZw, nZw);
(* Формирование S=S\{v} *)
nS:=nS-1;
(* Формирование 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); {Файл открыт для чтения}
{Ввод списка смежности}
Read(f, m); {Количество строк в списке}
Fst[1]:=0; {Указатель начала первой строки списка}
for i:=1 to m do begin
  Read(f, Vtx[i]); {Метка вершины}
  Read(f, Nbr[i]); {Количество вершин в списке}
  for j:=1 to Nbr[i] do Read(f, Adj[Fst[i]+j]);
    {Список смежных вершин}
  Fst[i+1]:=Fst[i]+Nbr[i]; {Указатель начала следующей
    строки в списке}
end;
Close(f);
Assign(f, 'Clique.out');
Rewrite(f); {Файл открыт для записи}
Init(yes);
if not yes then begin
  WriteLn(f, 'Плохая структура смежности графа!');
  Close(f);
  exit;
end;

```

```

{Формирование начальных множеств: S, D, N, Z}
nS:=0; {S - пустое}
kD:=0; nD:=0; {D - пустое множество}
kN:=0; nN:=m; {N - все вершины графа}
for i:=1 to m do N[i]:=i;
kZ:=0; nZ:=m; {Z - для исследования доступны все вершины}
for i:=1 to m do Z[i]:=i;
Clique (kN, nN, kD, nD, kZ, nZ); {Рекурсивное выделение
                                клик графа}

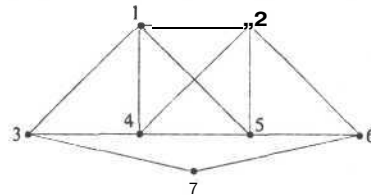
Close (f);
end.

```

Воспользуемся данной программой в качестве примера для решения следующей задачи.

Задача. 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. 6.34.

Рис. 6.34
Пример порождения клик графа



Для программы алгоритма 6.14 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 6.2.

Данные файла `Clique.in` для примера на рис. 6.34:

```

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 соответствующего клик «симпатичного» графа на рис.6.34. Отсюда видно, что в данном случае на вечер могут быть приглашены лишь четыре близких друга генерала.

6.12. 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 $G_0 = (X, U_0, \Phi)$ графа $G = (X, U, \Phi)$. 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 G относительно дерева G_0 . Так как каждое остовное дерево графа G включает $|X| - 1$ ребро, в фундаментальном множестве циклов относительно любого остовного дерева графа G 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 $F = \{C_1, C_2, \dots, C_{|U|-|X|+1}\}$ — фундаментальное множество циклов, где каждый цикл C_i является подмножеством ребер $C_i \subset U$. Тогда любой цикл графа G можно записать в виде $(\dots(C_i \oplus C_{i_2}) \oplus \dots) \oplus C_i$, где символ \oplus обозначает операцию симметрической разности, $C_p \oplus C_q = \{u \mid u \in C_p \cup C_q \wedge u \notin C_p \cap C_q\}$.

Например, на рис. 6.35 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.

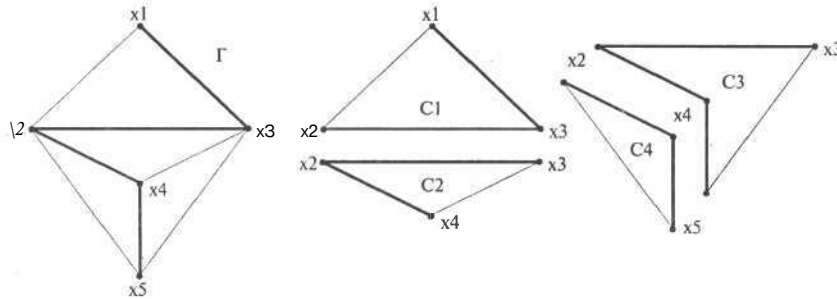


Рис. 6.35. Граф, его остовное дерево и фундаментальное множество циклов

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 6.15, который строит фундаментальное множество циклов $F = \{C_1, C_2, \dots, C_{|U|-|X|+1}\}$ графа $G = (X, U, \Phi)$.

Программная реализация поиска фундаментального множества циклов представлена в алгоритме 6.16.

Алгоритм 6.15. Фундаментальные циклы графа

```

for  $v \in X$  do  $Mark[v] = 0$ ; {Начальные метки вершин}
count = 0;
jC = 0; {Счетчик числа циклов}
nC = 0; {Вершина стека циклов}
for  $v \in X$  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; {Вершина исследована}
    for v s Adj[x] do begin
        nC = nC + 1;
        C[nC] = v; {Вершину в стек}
        if Mark[v] = 0 then Cycle(v, x)
        else if Mark[v] < Mark[x] и v ≠ y then begin
            jC = jC + 1; {Обратное ребро (x, v), найден цикл}
            WriteCycle(v, C, nC); {Печать цикла}
        end;
        nC = nC - 1; {Удалить исследованную вершину из стека}
    end;
end;
Procedure WriteCycle(x, C, nC)
    print jC; {Печать номера цикла}
    repeat
        print C[nC]; {Печать вершины из стека}
        nC = nC - 1;
    until C[nC] = x;
end.

```

Алгоритм 6.16. Программа поиска фундаментальных циклов графа

```

Program GraphCycle ; {Фундаментальные циклы графа}
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; {Текстовый файл}
    n :Integer; {Количество вершин}
    nC :Integer; {Количество вершин в стеке циклов}

```

```

Adj  :TypeAdjacent; { Список смежности графа }
Fst  :TypeVertex;  { Указатели вершин списка смежности}
Nbr  :TypeVertex;  { Количество вершин в списке
                    смежности }
Vtx  :TypeVertex;  { Список вершин графа }
Mark :TypeVertex;  { Номера компонент для вершин графа}
C    :TypeVertex;  { Стек выделения циклов графа }
B    :TypeVertex;  { Признаки основных и обратных ребер
                    прохода в глубину }

jC   :Integer;     { Счетчик числа циклов }
count:Integer;    { Счетчик меток вершин }

Procedure Init( var yes :Boolean );
{ Переименование меток вершин }
{ их порядковыми номерами в списке смежности }
{ yes - признак правильной структуры списка смежности }
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); {Печать цикла из стека}
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

```

6.13.1. Листы

- *Определение.* Пусть $G = (X, U, \Phi)$ — 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, наоборот, связный граф без циклических ребер является деревом.
- *Определение.* Ребро $u = (x, y) \in U$ называется *разделяющим ребром* (или *мостом*, или *разрезающим ребром*) в G , если в графе T , получающемся после удаления ребра u , вершины x и y не связаны. Тогда граф \tilde{T} можно представить как объединение графов

$$\tilde{T} = G_1 \cup G_2, \quad (6.13.1)$$

где $G_1 \cap G_2 = \emptyset$, и G_1 содержит x , G_2 содержит y .

- *Утверждение 6.13.1.* 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 (6.13.1) 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, \dots, 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. 6.36.



Рис. 6.36. Вершины циклически-реберно связаны

- *Утверждение 6.13.2.* Циклически-реберная связность определяет отношение эквивалентности (см. п.6.4) на множестве вершин графа $G = (X, U, \Phi)$. На рис. 6.37 показано разбиение G на компоненты циклически-реберной связности.

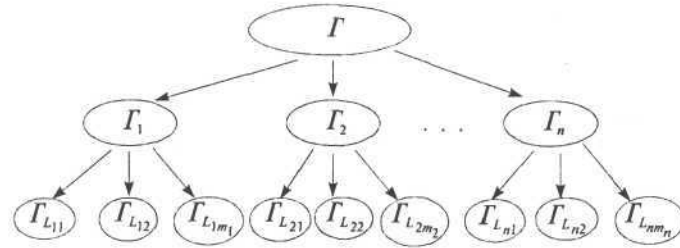


Рис. 6.37. Компоненты $\Gamma(L_{ij})$ циклически-реберной связности графа.
Компоненты $\Gamma(L_k)$ связности графа.
Компоненты $\Gamma(L_{ij}) \subseteq \Gamma(\zeta)$ связаны мостами

- *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 показано, 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 6.13.3.* Отметим следующие очевидные свойства листа $\Gamma(L)$.
 1. Граф $\Gamma(L)$ циклически замкнут, т.е. если какой-нибудь простой цикл C 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 различных листовых множеств 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.

6.13.2. 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, \dots, 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. 6.38.

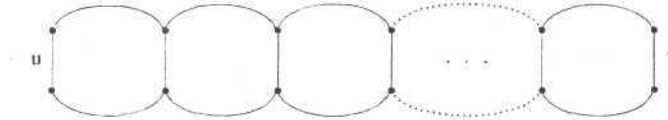


Рис. 6.38. Сильно циклически связанные ребра u, v

- *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^* этого графа называется *блоковым множеством*, определяемым ребром u . Блок является связным графом и может состоять из единственного ребра.
- *Лемма 6.13.1.* 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. 6.38).
- *Лемма 6.13.2.* 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 общнос-

ти, 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 6.13.1, 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 6.13.4.** Из леммы 6.13.2 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. 6.39). В противном случае блоки должны совпадать.



Рис. 6.39

- **Утверждение 6.13.5.** 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 $L^b \subset L$, где L — *листовое множество вершин* L^b , и каждый лист $\Gamma(L)$ имеет непересекающееся по ребрам разложение

$$\Gamma(L) = \bigcup_i \Gamma(L_i^b) \quad (6.13.2)$$

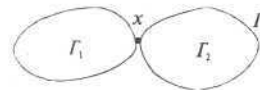
на семейство блоков.

- **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. 6.40)

$$\Gamma = \Gamma(V_1) \cup \Gamma(V_2), \quad V = V_1 \cup V_2, \quad V_1 \cap V_2 = x. \quad (6.13.3)$$

Рис. 6.40

Прямое по ребрам разложение графа



- **Утверждение 6.13.6.** Блок $\Gamma(L^b)$ не имеет разделяющих вершин, так как все его *вершины* циклически-реберно связаны.

На основании разложения (6.13.2) 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. 6.41,

в которой различные блоки соприкасаются в своих соединяющих вершинах. Соединяющие вершины блоков являются разделяющими вершинами графа. Разложение (6.13.2) листов на блоки $\Gamma(L_{ij}) = \bigcup_A \Gamma(L_{ijk}^b)$ позволяет теперь

уточнить структуру графа (рис. 6.37): каждый лист $\Gamma_{L_{ij}}$ можно представить в виде кактусообразной схемы (рис. 6.41) составляющих его блоков $\Gamma(L_{ijk}^b)$.

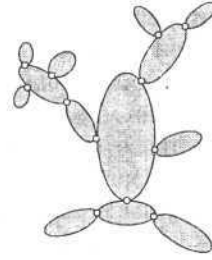


Рис. 6.41

6.13.3. Поиск блоков в глубину

Разложение графа на блоки и выделение их имеет важное практическое значение. Иногда недостаточно знать, что граф связан; может интересовать насколько «сильно связан» связный граф.

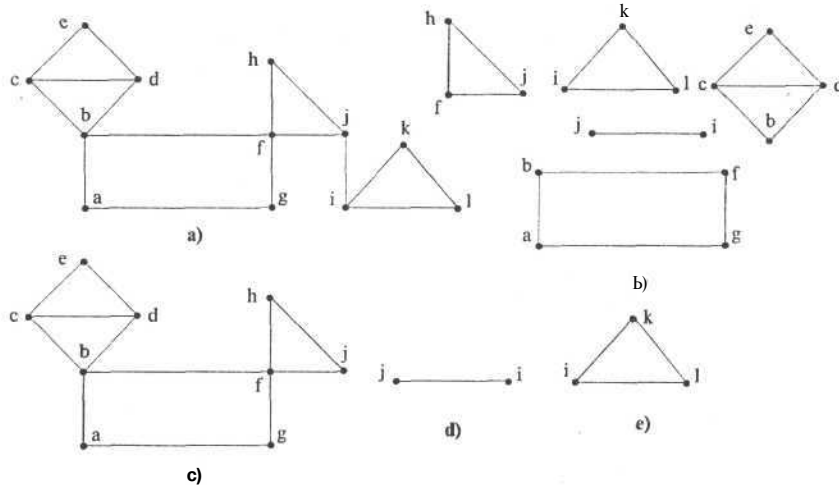


Рис. 6.42. Исходный граф (а); блоки графа (б); листья (с), (е) и один мост (д); точки сочленения: b, f, j, i

Например, удаляя вершину сочленения двух блоков (рис. 6.40) 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 транспорт-

ных 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. 6.42 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 $G = (X, U, \Phi)$. Основу алгоритма выделения блоков и точек их сочленения легче понять, рассмотрев пример на рис. 6.43, где схематически изображен связный граф, состоящий из шести блоков $G_i, i = \overline{1,6}$, и четырех точек сочленения $X_j, j = \overline{1,4}$.

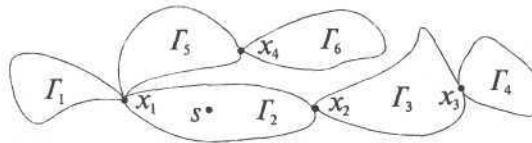


Рис. 6.43. Схематическое изображение графа с шестью блоками и четырьмя точками сочленения

Если начать поиск в глубину, скажем, из вершины $s \in G_2$, то можем перейти из G_2 в G_1 , проходя через вершину x_1 . Из свойства поиска в глубину, все ребра в G_1 должны быть пройдены до того, как вернемся в x_1 . Поэтому G_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 G_2 в G_5 , а оттуда — в G_6 через x_4 , то окажемся в G_6 , пройдя ребра из G_2, G_5 и G_6 . Если хранить ребра в стеке, то во время прохождения назад из G_6 в G_5 через вершину x_4 все ребра G_6 будут на верху стека. После их удаления на верху стека останутся ребра G_5 и снова продолжим проход блока G_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 6.17. В векторе $Mark[x]$ меток вершин $x \in X$ графа фиксируются порядковые номера обхода соответст-

вующих 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. 6.44 схематично представлен блок, где вершина w — точка входа в блок и точка сочленения. Вершина x — следующая обследованная вершина блока после w . 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 (z, 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.

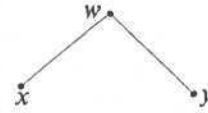


Рис. 6.44

Алгоритм 6.17. Выделение блоков графа

```

count = 0;
Stack = ∅;
for v ∈ X do Mark[v] = 0; {Начальные метки вершин}
for v ∈ X do if Mark[v] = 0 then Block(v, 0);
Procedure Block(x, w)
    count = count + 1;
    Mark[x] = count; {Вершина исследована}
    Virtual[x] = count; {Начальная минимальная метка}
    for v ∈ Adj[x] do
        if Mark[v] = 0 then begin
            Stack = Stack ∪ (x, v);

```

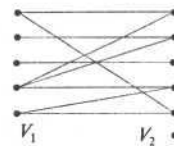
```

Block(v, x);
Virtual[x] = min(Virtual[x], Virtual[v]);
if Virtual[v] > Virtual[x] then begin
    В этой точке x — либо корень дерева, либо
    точка сочленения. Образовать новый блок,
    состоящий из всех ребер стека, включая (x, v).
end,
end;
else if Mark[v] < Mark[x] and v ≠ w then begin
    { (x, v) — обратное ребро }
    Stack = Stack ∪ (x, v);
    Virtual[x] = min(Virtual[x], Virtual[v]);
end;
end.

```

6.14. Двудольные графы

- *Определение.* Граф $G = (X, U, \Phi)$ 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 $V_1 \cap V_2 = \emptyset$. Такой граф будем обозначать как $G = (V_1 \cup V_2, U, \Phi)$.



На рис. 6.45 изображен типичный двудольный граф. Двудольные графы играют заметную роль в различных приложениях.

Рис. 6.45

6.14.1. Условия существования двудольных графов

- **Теорема 6.14.1.** Граф $G = (X, U, \Phi)$ является двудольным тогда и только тогда, когда любой его простой цикл четной длины.

Доказательство. (\Rightarrow) Ясно, что данное условие для двудольного графа всегда выполняется. (\Leftarrow) Разобьем граф G на компоненты связности G_1, G_2, \dots, G_m . Пусть $G_k = (X_k, U_k, \Phi)$ одна из них и $a \in X_k$ — произвольная вершина. Разобьем множество вершин X_k на непересекающиеся V_{k1} и V_{k2} , где V_{k1} — вершины, расстояние до которых от a нечетно; V_{k2} — вершины, расстояние до которых от a четно, $a \in V_{k2}$. Множества V_{k1} и V_{k2} являются независимыми. Действительно, если предположить, что вершины b и c смежные и

$b, c \in V_{k1}$ или $b, c \in V_{k2}$, то существовал бы цикл нечетной длины, соответственно «а нечетная длина B длина единица с нечетная длина a » или «а четная длина b длина единица с четная длина a », что противоречит условию. Рассмотренное разбиение вершин выполним для каждой компоненты $\Gamma_1, \Gamma_2, \dots, \Gamma_m$ связности. На рис. 6.46 показано объединение независимых компонент V_{k1} и V_{k2} в независимые компоненты $V_1 = \bigcup_{k1} V_{k1}$ и $V_2 = \bigcup_{k2} V_{k2}$.

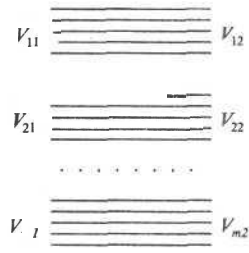


Рис. 6.46

6.14.2. Паросочетания

• *Определение.* Паросочетанием в двудольном графе $\Gamma = (V_1 \cup V_2, U, \Phi)$ называется независимое подмножество ребер π с U , ребра π не имеют общих вершин.

Для графа, изображенного на рис. 6.47, в качестве паросочетания можно взять множество ребер $\pi = \{(s_1, 1), (s_2, 2)\}$, $\pi = \{(s_3, 1), (s_2, 2)\}$ и т.п.



Рис. 6.47

• *Определение максимального паросочетания.* Паросочетание называется максимальным, если любое другое паросочетание содержит меньшее число ребер.

6.14.3. Алгоритм определения максимального паросочетания

Пусть $\Gamma = (V_1 \cup V_2, U, \Phi)$ — двудольный граф и π — произвольное паросочетание. Множество вершин $A_1 \subset V_1$ и $A_2 \subset V_2$ (рис. 6.48). Чередующейся цепью относительно паросочетания π называется цепь C , в которой ребра поочередно принадлежат U и π и которая начинается ребром, не принадлежащим π . 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 $a_2 \in A_2$.

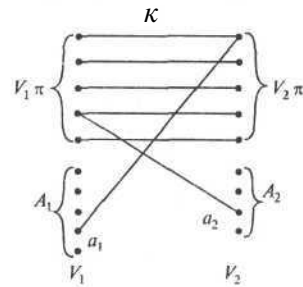


Рис. 6.48

Такую цепь S можно использовать для получения нового паросочетания γ , которое содержит на одно ребро больше, чем исходное паросочетание α . Действительно, в γ можно включить все ребра α , удалив $\pi_1, \pi_2, \dots, \pi_{n-1}$ и добавив $u_1, u_2, \dots, u_{n-1}, u_n$. Полученное таким образом γ содержит несмежные ребра, а значит, γ — паросочетание. Говорят, что π получено из α чередующимся расширением.

Пример. Дан двудольный граф $G = (V_1 \cup V_2, U, \Phi)$, 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: $s_1 \rightarrow \{4\}$, $s_2 \rightarrow \{1, 6\}$, $s_3 \rightarrow \{4, 5, 6\}$, $s_4 \rightarrow \{1, 3\}$, $s_5 \rightarrow \{2\}$, $s_6 \rightarrow \{1, 3\}$. Найти максимальное паросочетание.

Выберем начальное паросочетание α таким образом, что вершину s_1 в V_1 соединяем с вершиной $j \in V_2$ первой незанятой ранее из списка соединений для s_1 . На рис. 6.49 изображено исходное паросочетание $\alpha = \{(s_1, 4), (s_2, 1), (s_3, 5), (s_4, 3), (s_5, 2)\}$, $|\alpha| = 5$. Вершины s_6 и 6 не вошли в паросочетание. Попробуем увеличить α , используя алгоритм чередующихся цепей. Обозначим $\overset{\curvearrowright}{\rightarrow}$ переход по ребрам графа из V_1 в V_2 и $\overset{\curvearrowleft}{\rightarrow}$ переход по ребрам паросочетания α из V_2 в V_1 . Так, $s_6 \overset{\curvearrowright}{\rightarrow} \{1, 3\}$

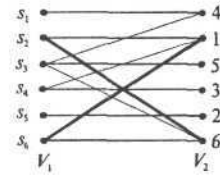


Рис 6.49

означает, что из s_6 можно перейти в вершины 1 и 3, и $\{1, 3\} \overset{\curvearrowleft}{\rightarrow} \{s_2, s_4\}$ означает, что из вершин 1 и 3 можно достичь по ребрам паросочетания вершин s_2 и s_4 . По алгоритму исходной вершинной цепи является s_6 . Тогда множество всех чередующихся цепей, начало которых в s_6 , можно представить: $s_6 \overset{\curvearrowright}{\rightarrow} \{1, 3\} \overset{\curvearrowleft}{\rightarrow} \{s_2, s_4\} \overset{\curvearrowright}{\rightarrow} \{1, 3, 6\}$. 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 не доступна из s_6 и, значит, исходное паросочетание α максимальное. В нашем случае вершина 6 оказалась доступной. Для выделения исходной чередующейся цепи из всего множества расширяющихся цепей выполним обратный ход: $6 \overset{\curvearrowleft}{\rightarrow} s_2 \overset{\curvearrowleft}{\rightarrow} 1 \overset{\curvearrowright}{\rightarrow} s_6$. Теперь новое паросочетание строим из старого, исключая из него ребро $(s_2, 1)$ и включая ребра $(6, s_2)$, $(1, s_6)$. Процесс включения также показан на рис. 6.49. Максимальное паросочетание содержит ребра $(s_1, 4)$, $(s_3, 5)$, $(s_4, 3)$, $(s_5, 2)$, $(s_2, 6)$, $(s_6, 1)$ и $|\alpha| = 6$.

• **Теорема 6.14.2** о максимальном паросочетании.

Пусть $\Gamma = (V_1 \cup V_2, U, \Phi)$ — двудольный граф. Тогда количество ребер в максимальном паросочетании равно

$$\pi(\Gamma) = |V_1| - \max_{A \subseteq V_1} (|A| - |\Delta A|), \quad (*)$$

где $\Delta A = \{y \in V_2 \mid \exists x \in V_1 \wedge (x, y) \in U\} \subseteq V_2$.

Доказательство. Пусть λ — максимальное паросочетание в исходном двудольном графе (рис. 6.50). Паросочетание λ позволяет рассматривать его как отображение $\lambda: V_2 \rightarrow V_1$ вершин множества V_2 в вершины множества V_1 по ребрам паросочетания λ .

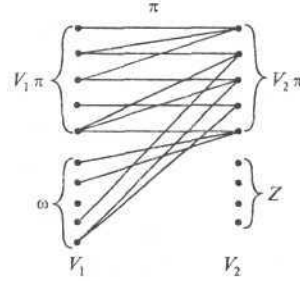


Рис. 6.50. Максимальное паросочетание λ

Аналогично, под Δ будем понимать отображение $\Delta: V_1 \rightarrow V_2$ вершин множества V_1 в вершины множества V_2 по ребрам U графа. Вершины множеств ω и Z (рис. 6.50) несмежные, так как паросочетание λ является максимальным.

Рассмотрим процесс насыщения множества ω . Под $\omega \rightarrow_{\Delta} \omega_{\pi}$ будем понимать последовательное выполнение двух отображений $\pi(\Delta(\omega))$ с V_1 множества ω . Далее применим отображения Δ и λ к полученному множеству: $\omega_{\pi} \rightarrow_{\Delta} (\omega_{\pi})_{\pi}$ с V_1 и т. д. Ясно, что каждая пара отображений приводит к расширению исходного множества $\omega_{\pi} \subseteq (\omega_{\pi})_{\pi}$. Учитывая конечные размеры исходного графа, наступит момент, когда отображаемое множество перестанет расширяться. Таким образом, процесс насыщения отображаемого множества можно представить следующим образом:

$$\omega \rightarrow_{\Delta} \omega_{\pi} \rightarrow_{\Delta} (\omega_{\pi})_{\pi} \rightarrow_{\Delta} ((\omega_{\pi})_{\pi})_{\pi} \rightarrow \dots \rightarrow_{\Delta} (\dots (\omega_{\pi})_{\pi} \dots)_{\pi} = A.$$

Для множества A выполняется условие $|A| = |\Delta A|$, в противном случае множество A можно было бы еще расширить. Следствием расширения множества $\Delta\omega$ в процессе его отображения является включение $\Delta\omega \subseteq \Delta A$

Обозначим множество $\omega \cup A = B$. Покажем, что найденное множество B удовлетворяет условию (*) теоремы, т.е. $\pi(\Gamma) = |V_1| - (|B| - |\Delta B|)$. Имеем $|\Delta B| = |\Delta(\omega \cup A)| = |\Delta\omega \cup \Delta A| = |\Delta A|$, следовательно, $|V_1| - (|B| - |\Delta B|) = |V_1| - (|\omega| + |A| - |\Delta A|) = |V_1| - |\omega| = \pi(\Gamma)$.

Покажем теперь, что $\forall A \subseteq V_1$ условие (*) теоремы записывается в следующем виде: $\pi(\Gamma) < |V_1| - (|A| - |\Delta A|)$. Тогда ранее найденное множество B , для которого данное неравенство обращается в равенство, будет доказательством условия (*) теоремы. Пусть $A_1 = A \setminus (\omega \cap A)$, тогда $A = (\omega \cap A) \cup A_1$. Отметим, что $\forall A_1 \subseteq V_1 \setminus (\omega \cap A_1)$ выполняется неравенство $|A_1| < |\Delta A_1|$, так как вершины множества $V_1 \setminus (\omega \cap V_1)$ составляют паросочетание. Тогда верно, что $|A| = |\omega \cap A| + |A_1| < |\omega| + |A_1| < |\omega| + |\Delta A_1| < |\omega| + |\Delta A|$ или $|A| - |A \setminus \omega| < |\omega|$. Теперь количество ребер в максимальном паросочетании можно оценить: $\pi(\Gamma) = |V_1| - |\omega| < |V_1| - (|A| - |\Delta A|)$. Теорема доказана.

6.14.4. 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 $x_i \in S_i, i = 1, 2, \dots, 5$. 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 $S = \{1, 2, \dots, n\}$. $M(S)$ — система подмножеств S_1, S_2, \dots, S_n множества S . Будем говорить, что $M(S)$ имеет систему различных представителей, если для всех $i = 1, 2, \dots, n$ существуют различные $x_i \in S_i$.

6.14.5. Связь системы различных представителей и двудольных графов

Определим двудольный граф $\Gamma = (V_1 \cup V_2, U, \Phi)$, соответствующий системе подмножеств. Пусть $M(S) = \{S_1, S_2, \dots, S_n\}$ — система подмножеств S_1, S_2, \dots, S_n множества S . Положим S_1, S_2, \dots, S_n вершинами V_1 графа, которые соединены ребрами со своими элементами — смежными им вершинами из V_2 (рис. 6.51).

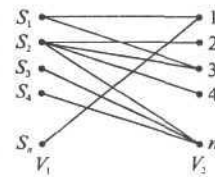


Рис. 6.51

- **Лемма 6.14.1.** Двудольный граф $\Gamma = (V_1 \cup V_2, U, \Phi)$, отвечающий системе подмножеств $M(S) = \{S_1, S_2, \dots, S_n\}$, 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 6.14.3 Ф. Холла о существовании системы различных представителей.* Система $M(S) = \{S_1, S_2, \dots, S_n\}$ имеет систему различных представителей тогда и только тогда, когда для любой подсистемы $\{S_{i_1}, S_{i_2}, \dots, S_{i_k}\}$ с $M(S)$ выполняется неравенство

$$\left| \bigcup_{j=1}^k S_{i_j} \right| \geq k, \text{ т.е. количество элементов в объединении любых } k \text{ подмножеств должно быть не менее } k.$$

Доказательство. (\Rightarrow) Необходимое условие теоремы следует из определения системы различных представителей. Каждое подмножество $S_{i_j} \in M(S)$ содержит элемент $x_{i_j} \in S_{i_j}$, отличный от элементов других подмножеств, а значит $\left| \bigcup_{j=1}^k S_{i_j} \right| \geq k$.

(\Leftarrow) Пусть $\Gamma = (V_1 \cup V_2, U, \Phi)$ — двудольный граф, соответствующий системе подмножеств $M(S) = \{S_1, S_2, \dots, S_n\}$. Покажем, что в данном графе существует максимальное паросочетание, количество ребер в котором равно n . Тогда из леммы 6.14.1 будет следовать достаточное условие теоремы. Из теоремы 6.14.2 имеем, что число ребер в максимальном паросочетании равно $\pi(\Gamma) = |V_1| - \max_{A \subseteq V_1} (|A| - |\Delta A|)$, где $\Delta A = \{y \in V_2 \mid \exists x \in V_1 \text{ л } (x, y) \in U\}$ с V_2 . В рамках принятой интерпретации $A = \{S_{i_1}, S_{i_2}, \dots, S_{i_k}\}$, $|A| = k$ и $\Delta A = \bigcup_{j=1}^k S_{i_j}$. По условию теоремы $|\Delta A| \geq k$. Таким образом, $\forall A \subseteq V_1$ $|A| - |\Delta A| \leq 0$, а значит, $\pi(\Gamma) = |V_1| - \max_{A \subseteq V_1} (|A| - |\Delta A|) \geq |V_1|$. Однако $\pi(\Gamma) < |V_1|$, следовательно, $\pi(\Gamma) = |V_1| = n$ (*достаточное условие доказано*).

6.14.6. Задача о назначениях

Существует множество задач, постановки которых укладываются в рамки задачи о назначениях. Рассмотрим две такие постановки.

Задача. Предположим, что вычислительная сеть объединяет n специализированных ЭВМ. На вход сети поступают задачи. 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 за-

дач. 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 6.14.4** — алгоритм поиска оптимальной перестановки π .

Пусть $A = [a_{ij}]$, $i, j = \overline{1, n}$ — матрица целых чисел. Тогда максимум

$$\max_{\pi} \sum_{i=1}^n a_{i\pi_i} \quad (6.14.1)$$

по всем перестановкам π равен минимуму

$$\min_{u_i + v_j \geq a_{ij}} \left(\sum_{i=1}^n u_i + \sum_{j=1}^n v_j \right) \quad (6.14.2)$$

по всем числам u_i и v_j таким, что

$$u_i + v_j \geq a_{ij}, \quad \forall i, j = \overline{1, n}. \quad (6.14.3)$$

Минимум суммы (6.14.2) достигается на перестановке π такой, что

$$u_i + v_{\pi_i} = a_{i\pi_i}, \quad \forall i = \overline{1, n}.$$

Доказательство. Пусть π — произвольная перестановка. При изменении i от 1 до n величины π_i принимают все значения множества $\{1, 2, \dots, n\}$. По условию $u_i + v_j \geq a_{ij}$, $\forall i, j = \overline{1, n}$, а значит, и для $j = \pi_i$ верно $u_i + v_{\pi_i} \geq a_{i\pi_i}$. Установим связь сумм (6.14.1) и (6.14.2).

$$\sum_{i=1}^n u_i + \sum_{j=1}^n v_j = \sum_{i=1}^n u_i + \sum_{\pi_i=1}^n v_{\pi_i} = \sum_{i=1}^n (u_i + v_{\pi_i}) \geq \sum_{i=1}^n a_{i\pi_i}.$$

Таким образом, сумма (6.14.1) для любой перестановки π не больше суммы (6.14.2). Отсюда следует, что теорема будет верна, если мы найдем такие u_i и v_j и перестановку π , что (6.14.1) и (6.14.2) совпадают.

Шаг 0. Поиск начальных u_i и v_j , удовлетворяющих ограничениям $u_i + v_j \geq a_{ij}$, $\forall i, j = \overline{1, n}$. Положим $v_j = 0$ и $u_i = \max_j a_{ij}$ — максимальный элемент в i -й строке; условие (6.14.3) $u_i + v_j = \max_j a_{ij} \geq a_{ij}$ выполняется. Для матрицы ценностей на рис. 6.52 справа и снизу указаны начальные значения, соответственно u_i и v_j .

	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. 6.52. Начальная матрица ценностей

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 $u_i + v_j > a_{ij}$ не обратится в равенство $u_i + v_j = a_{ij}$. Эту процедуру выполняем для каждой строки $i = \overline{1, n}$. В рассматриваемом примере на рис. 6.53 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. 6.53. Матрица совпадений

Шаг 2. Для каждого $u_i, i = \overline{1, n}$ определим множества совпадений

$$S_i = \{j \mid u_i + v_j = a_{ij}\}, i = \overline{1, n}.$$

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: $j_1 \in S_1, j_2 \in S_2, \dots, j_n \in S_n$, где j_1, j_2, \dots, j_n — все различные. Тогда на перестановке $\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ j_1 & j_2 & \dots & j_n \end{pmatrix}$ выполняется соотношение

$u_i + v_{j_i} = u_i + v_{\pi(i)} = a_{i\pi(i)}$, а значит, найденная перестановка π является оптимальной.

Шаг 3. Предположим, что S_1, S_2, \dots, S_n не имеют системы различных представителей, как в рассматриваемом примере. Тогда нарушается условие теоремы Ф. Холла, т. е. существуют $S_{i_1}, S_{i_2}, \dots, S_{i_k}$ такие, что $|S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k}| < k$. В нашем примере $|S_1 \cup S_3 \cup S_4| = 2 < 4$. Перейдем к новым значениям u_i^* и v_j^* , полагая

$$u_i^* = u_{i_1} - 1, u_{i_2}^* = u_{i_2} - 1, \dots, u_{i_k}^* = u_{i_k} - 1 \text{ и} \\ v_j^* = v_j + 1, \text{ если индекс } j \in S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k}.$$

Остальные u_i и v_j остаются без изменений. Обратимся к нашему примеру. На рис. 6.54 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. 6.54. Новые значения u_i^* и v_j^*

	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. 6.55. Матрица совпадений для новых значений u_i^* и v_j^*

При такой замене переменных не нарушается основное условие $u_i + v_j > a_{ij}, \forall i, j = 1, n$. Действительно, возможны два случая:

- $j \in S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k}$, тогда $u_i^* + v_j^* = (u_i - 1) + (v_j + 1) = u_i + v_j \geq a_{ij}$.
- $j \notin S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k}$, а значит, $v_j^* = v_j$ и $u_i + v_j > a_{ij}$ или

$$u_i + v_j \geq a_{ij} + 1. \text{ Отсюда } u_i^* + v_j^* = (u_i - 1) + v_j \geq a_{ij} + 1 - 1 = a_{ij}.$$

Обозначим $|S_{i_1} \cup S_{i_2} \cup \dots \cup S_{i_k}| = t < k$, тогда новая сумма (6.14.2)

$$\sum_{i=1}^n u_i^* + \sum_{j=1}^n v_j^* = \left(\sum_{i=1}^n u_i \right) - k + \left(\sum_{j=1}^n v_j \right) + t = \left(\sum_{i=1}^n u_i + \sum_{j=1}^n v_j \right) - (k - t).$$

Сумма уменьшилась на $k - t$. Если теперь перейти к шагу 2, то в случае существования различных представителей для новых значений u_i^* и v_j^* , задача будет решена, в противном случае шагу 2 и 3 алгоритма продолжаем. Так как каждое повторение шагов алгоритма приводит к уменьшению суммы (6.14.2)

$$\sum_{i=1}^n u_i^* + \sum_{j=1}^n v_j^* \text{ на } k - t \text{ единиц,}$$

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: $u_i + v_j \geq a_{ij} \forall i, j = \overline{1, n}$.

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 на рис. 6.54, 6.55. Составим для нее систему множеств совпадений $S_1 = \{2\}, S_2 = \{2, 3, 4\}, S_3 = \{1, 4\}, S_4 = \{1, 3, 4\}$. 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 $G = (V_1 \cup V_2, U, \Phi)$, где $V_1 = \{S_1, S_2, S_3, S_4\}, V_2 = \{1, 2, 3, 4\}$, и связь вершин $S_1 \rightarrow \{2\}, S_2 \rightarrow \{2, 3, 4\}, S_3 \rightarrow \{1, 4\}, S_4 \rightarrow \{1, 3, 4\}$. 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:

$$\pi_1 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix} \quad \pi_2 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix} \quad \pi_3 = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 4 & 1 & 3 \end{pmatrix}.$$

Ценность назначения составляет $a_{12} + a_{23} + a_{31} + a_{44} = 18$.

6.15. Хроматические графы

Пусть $G = (X, U, \Phi)$ — простой граф. Граф G называется k -раскрашиваемым, если существует такое разложение множества его вершин X на k непересекающихся подмножеств (компонент) C_1, C_2, \dots, C_k таких, что

$$X = \bigcup_{i=1}^k C_i, \quad C_i \cap C_j = \emptyset \text{ при } i \neq j \quad (6.15.1)$$

и вершины в каждой компоненте C_i независимы, т.е. ребра соединяют вершины только из разных компонент (рис. 6.56). Представление (6.15.1) называется k -раскраской графа G . 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 (6.15.1) называется *хроматическим числом* графа Γ .

Рассмотрим несколько примеров хроматического разложения графов.

- **Утверждение 15.1.** Полный граф $\Gamma = (X, U, \Phi)$ на $|X| = n$ вершинах имеет хроматическое число $\chi(\Gamma) = n$. Здесь каждая компонента $C_i, i = \overline{1, n}$ разложения (6.15.1) состоит из одной вершины, $|C_i| = 1, i = \overline{1, n}$.
- **Утверждение 15.2.** Граф $\Gamma = (X, U, \Phi)$ содержит максимальный подграф (клик) из k вершин тогда и только тогда, когда его хроматическое число $\chi(\Gamma)$ равно k .
Доказательство. (\Rightarrow) Пусть $\{x_1, x_2, \dots, x_k\} \in X$ вершины максимального подграфа. Для разложения (6.15.1) максимального подграфа требуется k компонент C_1, C_2, \dots, C_k таких, что $x_i \in C_i$. Покажем, что этого числа компонент достаточно для разложения (6.15.1) всего графа Γ . Пусть $y \in X$ — произвольная вершина и $y \notin \{x_1, x_2, \dots, x_k\}$. 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 Γ содержит m вершин и $m \neq k$. Необходимое условие теоремы в этом случае утверждает: $\chi(\Gamma) = m$, что противоречит условию.
- **Утверждение 15.3.** Граф $\Gamma = (X, U, \Phi)$ является двудольным тогда и только тогда, когда $\chi(\Gamma) = 2$.
- **Утверждение 15.4.** Хроматическое число дерева равно 2, так как дерево является двудольным графом.
- **Теорема 6.15.1.** Для числа $\beta(\Gamma)$ вершинной независимости и хроматического числа $\chi(\Gamma)$ графа $\Gamma = (X, U, \Phi), |X| = n$ выполняется соотношение

$$\beta(\Gamma)\chi(\Gamma) \geq n. \quad (6.15.2)$$

Доказательство. В разложении (6.15.1) все компоненты C_i являются независимыми. Из определения числа $\beta(\Gamma)$ следует, что

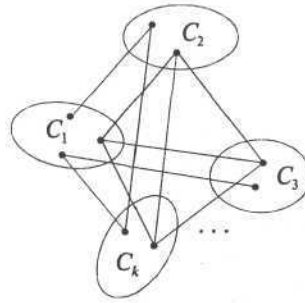


Рис 6 56

$|C_i| \leq \beta(G)$. Суммируя по всем компонентам, получим $n = \sum_{i=1}^k |C_i| \leq k \cdot \beta(G)$, где $k = \chi(G)$.

Задача. Для графа G на рис. 6.57 найти разложение (6.15.1) и установить, что хроматическое число $\chi(G) = 3$.

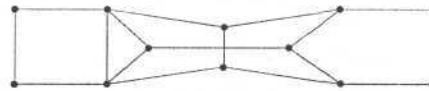


Рис. 6.57

Задача. Пусть l — длина самой длинной простой цепи в графе G . Показать, что $\chi(G) < l + 1$.

Решение. Предположим, что $\chi(G) \geq l + 2$. Тогда утверждение 15.2 позволяет сделать заключение, что граф G содержит максимальный подграф (клику) из $l + 2$ вершин или более. Такой подграф содержит 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 $\chi(G) \leq l + 1$.

6.16. Диаметр, радиус и центры графа

I. Пусть $G = (X, U, \Phi)$ — конечный связный псевдограф. Обозначим через $d(x, y)$ длину минимального маршрута между вершинами $x, y \in X$. Величина $d(G) = \max_{x, y \in X} d(x, y)$ называется *диаметром* графа.

II. Пусть $x \in X$ — произвольная вершина графа. Величина $r(x) = \max_{y \in X} d(x, y)$ называется *максимальным удалением* в графе G от вершины x .

III. *Радиусом* графа G называется величина $r(G) = \min_{x \in X} r(x)$.

IV. Любая вершина $z \in X$, для которой $r(z) = r(G)$, называется *центром* графа.

Задача. Для графа G на рис. 6.58 найти диаметр, радиус и все центры.

Решение. Диаметр $d(G) = \max_{x, y \in X} d(x, y) = 3$. $r(x_1) = 3$, $r(x_2) = 2$, $r(x_3) = 3$, $r(x_4) = 2$, $r(x_5) = 3$, $r(x_6) = 2$. Следовательно, радиус графа $r(G) = \min_{x \in X} r(x) = 2$.

Центры графа: x_2, x_4, x_6 .

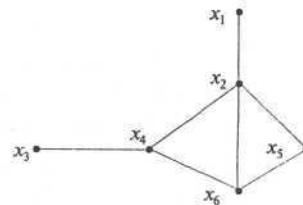


Рис. 6.58

Введение в теорию групп. Приложения

7.1. 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 относительно 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. $G_1 = \{n \mid n \in \mathbb{Z}\}$ — группа с операцией сложения чисел, где \mathbb{Z} — множество целых чисел. Действительно, 0 — единица группы; $n^{-1} = (-n)$ — обратный элемент к $n \in G_1$.
2. $G_2 = \{2n \mid n \in \mathbb{Z}\}$ — группа с операцией сложения чисел.
3. $G_3 = \{2^n \mid n \in \mathbb{Z}\}$ — 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.7.6).
- *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 с G_1 , G_5 с G_4 , G_7 с G_8 .
 - *Утверждение 7.1.1.* 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 $e \cdot a^{-1} \in M$ или $a^{-1} \in M$.
3. *Замкнутость.* Пусть $a, b \in M$, тогда $b^{-1} \in M$. Значит, и $a \cdot (b^{-1})^{-1} \in M$ или $ab \in M$.

7.2. Гомоморфизм групп

- *Определение.* Гомоморфизмом группы G_1 в G_2 называется отображение $f: G_1 \rightarrow G_2$, сохраняющее операции: $\forall a, b \in G_1$ $f(a \circ b) = f(a) \bullet f(b)$, где \circ — операция в группе G_1 ; \bullet — операция в группе G_2 . Если f — взаимно однозначное отображение, то/ 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 $M = e_2$. 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/называется подмножество $\text{Im} f = \{a_2 \in G_2 \mid \exists a_1 \in G_1 \ a_2 = f(a_1)\} \subset G_2$.
- *Определение.* Ядром гомоморфизма/называется подмножество $\text{Ker} f = \{a_1 \in G_1 \mid f(a_1) = e_2 \in G_2\} \subset G_1$.
- *Утверждение 7.2.1.* $\text{Im} f \subseteq G_2$ — подгруппа.
Доказательство. Проверим все свойства (аксиомы) группы.
 1. *Замкнутость.* $\forall a_2, b_2 \in \text{Im} f \ \exists a_1, b_1 \in G_1 \ f(a_1) = a_2 \wedge f(b_1) = b_2$. Тогда $a_2 b_2 = f(a_1) f(b_1) = f(a_1 b_1) \in \text{Im} f$.
 2. *Существование единичного элемента.* Так как $e_2 = f(e_1) \in \text{Im} f$.
 3. *Существование обратного элемента.* $\forall a_2 \in \text{Im} f \ \exists a_1 \in G_1 \ f(a_1) = a_2$. Тогда и $a_2^{-1} = (f(a_1))^{-1} = f(a_1^{-1}) \in \text{Im} f$.
- *Утверждение 7.2.2.* $\text{Ker} f \subseteq G_1$ — подгруппа.
Доказательство. Проверим все свойства (аксиомы) группы.
 1. *Замкнутость.* $\forall a_1, b_1 \in \text{Ker} f \ f(a_1 b_1) = f(a_1) f(b_1) = e_2 e_2 = e_2$. Следовательно, $f(a_1 b_1) = e_2$.
 2. *Существование единичного элемента.* Так как $f(e_1) = e_2$, значит, $e_1 \in \text{Ker} f$.
 3. *Существование обратного элемента.*
 $\forall a_1 \in \text{Ker} f \ f(a_1^{-1}) = (f(a_1))^{-1} = e_2^{-1} = e_2$, откуда $a_1^{-1} \in \text{Ker} f$.

7.3. Смежные классы

- *Определение.* Пусть $H = \{e, h_1, h_2, \dots, h_{m-1}\}$ — подгруппа группы G . Множество $gH = \{ge, gh_1, gh_2, \dots, gh_{m-1}\}$, полученное умножением элементов H слева на элемент $g \in G$, называется *левым смежным классом* группы G по подгруппе H .
- *Лемма 7.3.1.* Пусть Y — подгруппа группы G , тогда $\forall h \in H \ hY = Y$.
Доказательство. Пусть $H = \{e, h_1, h_2, \dots, h_{m-1}\}$. $\forall h_k \in H \ hh_k \in H$ — замкнутость подгруппы, значит $hY \subset Y$. С другой стороны, $\forall h_k \in H \ h_k = eh_k = (hh^{-1})h_k = h(h^{-1}h_k) \in hY$, откуда $H \subset hY$. Таким образом, $\forall h \in Y \ hY = Y$.
- *Лемма 7.3.2.* Пусть Y — подгруппа группы G . Если $g_1 H \cap g_2 H \neq \emptyset$, то $g_1 H = g_2 H$, где $g_1, g_2 \in G$.

Доказательство. Пусть $t \in g_1H \cap g_2H$, тогда $\exists h_1, h_2 \in Y$ такие, что $t = g_1h_1 = g_2h_2$. Рассмотрим левый смежный класс $tH = (g_1h_1)H = (g_2h_2)H = g_1(h_1H) = g_2(h_2H)$, откуда $g_1H = g_2H$.

- **Лемма 7.3.3.** Пусть Y — подгруппа группы G , тогда $\forall g \in G$ $|gH| = |H|$.

Доказательство. Для доказательства достаточно показать, что все элементы в gH различны. Пусть $\exists h_1 \neq h_2 \in Y$ такие, что $gh_1 = gh_2$. Тогда $g^{-1}(gh_1) = g^{-1}(gh_2)$ или $(g^{-1}g)h_1 = (g^{-1}g)h_2$, откуда $h_1 = h_2$, что противоречит предположению.

- **Лемма 7.3.4.** Группа G распадается на непересекающиеся левые смежные классы по подгруппе Y , т.е. $G = g_1H \cup g_2H \cup \dots \cup g_sH$, где все g_1, g_2, \dots, g_s — различные; $g_iH \cap g_jH = \emptyset$, если $fi \neq fj$.

Доказательство. Пусть $G = \{g_{k_1}, g_{k_2}, \dots, g_{k_s}\} \cdot Y$. Ясно, что $G = g_{k_1}H \cup g_{k_2}H \cup \dots \cup g_{k_s}H$. Воспользовавшись леммой 7.3.2 и удалив совпадающие левые смежные классы из последнего разложения, получим искомое разложение группы $G = g_1H \cup g_2H \cup \dots \cup g_sH$.

- **Теорема 7.3.1 Лагранжа.** Порядок конечной группы G кратен порядку любой из ее подгрупп Y и $|G| = |Y| \cdot |G: Y|$, где $|G: Y|$ — целое число, называется индексом подгруппы Y в группе G .

Доказательство. Лемма 7.3.4 позволяет записать разложение группы $G = g_1H \cup g_2H \cup \dots \cup g_sH$, где $g_iH \cap g_jH = \emptyset$ при $fi \neq fj$. Из леммы 7.3.3 имеем $\forall g_i |g_iH| = |H|$, тогда $|G| = |g_1H \cup g_2H \cup \dots \cup g_sH| = |g_1H| + |g_2H| + \dots + |g_sH| = s \cdot |H|$. Следовательно, $|G| = |Y| \cdot |G: Y|$, где $|G: Y| = s$.

- **Определение.** Пусть G — группа. Порядком элемента $g \in G$ ($g \neq e$) называется наименьшее целое k такое, что $g^k = e$.
- **Утверждение 7.3.1.** Пусть G — группа и $g \in G$ — произвольный элемент порядка k . Тогда множество $H = \{g, g^2, \dots, g^k = e\}$ называется циклической подгруппой, а g есть образующий ее элемент.
- **Утверждение 7.3.2.** Порядок группы $|G|$ кратен порядку любого элемента $g \in G$. (Следствие теоремы 7.3.1 Лагранжа и утверждения 7.3.1).
- **Утверждение 7.3.3.** Всякая циклическая группа коммутативна (абелева).
- **Теорема 7.3.2.** Всякая подгруппа циклической группы сама является циклической.

Доказательство. Пусть G — циклическая группа с образующим элементом $g \in G$ и $H \subset G$ — 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. 8.1). Следовательно, подгруппа H в этом случае должна содержать элемент $g^d = (g^k)^u (g^l)^v = g^{ku+lv}$. Так как $d < k$, то приходим к противоречию относительно выбора элемента g^k . Таким образом, g^k — образующий элемент подгруппы H с (Я)

- *Утверждение 7.3.4.* Число образующих циклической группы $G = \{g, g^2, \dots, g^n = e\}$ равно значению функции Эйлера $\varphi(n)$ — количество чисел из множества $\{1, 2, \dots, n-1\}$ взаимно простых с n .

Доказательство. Пусть $r \in \{1, 2, \dots, n-1\}$ и $(r, n) = 1$ — НОД (см. п. 8.1), 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 $(g^r)^q = (g^{sp})^q = g^{spq} = g^{p \cdot n} = (g^n)^p = e^p = e$, а значит порядок элемента $|g^r| = q < n$. Действительно, порядок $|g^r|$ не меньше q , в противном случае имеем $(g^r)^k = (g^r)^k = e$, 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 , т.к. $(p, q) = 1$. Это противоречит предположению $k < q$.

- *Определение.* Подгруппа H группы G называется *нормальным делителем*, если правые смежные классы совпадают с левыми: $\forall g \in G \quad gH = Hg$.
- *Утверждение 7.3.5.* Множество смежных классов группы G по нормальному делителю H является группой с операцией умножения смежных классов. Такая группа называется *фактор-группой* и обозначается G/H . Элементами этой группы являются смежные классы разложения группы $G = g_1H \cup g_2H \cup \dots \cup g_sH$ на непересекающиеся левые смежные классы, т.е. $G/H = \{g_1H, g_2H, \dots, g_sH\}$.

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 7.3.3.* Для любого нормального делителя H группы G отображение

$$f: G \rightarrow G/H, \text{ где } \forall g \in G \ f(g) = gH,$$

является гомоморфизмом, ядро которого H , G/H — факторгруппа.

Доказательство. Проверим свойство гомоморфизма сохранения операций: $f(g_1 \cdot g_2) = (g_1 \cdot g_2)H = (g_1H) \circ (g_2H) = f(g_1)f(g_2)$. Единицей факторгруппы G/H является H , тогда

$$\text{Ker } f = \{g \in G \mid f(g) = H\}.$$

Имеем $\forall h \in H \ f(h) = hH = H$, откуда $H \subseteq \text{Ker } f$. 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 $H \subseteq \text{Ker } f$, а значит, $\text{Ker } f = H$.

- *Теорема 7.3.4.* Ядро произвольного гомоморфизма есть нормальный делитель.

Доказательство. Пусть $f: G \rightarrow K$, где G, K — группы, f — гомоморфизм. $\text{Ker } f = \{h \in G \mid f(h) = e \in K\}$. Обозначим $H = \text{Ker } f$ — подгруппа. Покажем, что $\forall g \in G \ gH = Hg$, 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 $f(g_1g^{-1}) = f(g_1)f(g^{-1}) = kk^{-1} = e$ и $f(g^{-1}g_1) = f(g^{-1})f(g_1) = k^{-1}k = e$. Отсюда $g_1g^{-1} \in H$ 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, $\forall h \in H \ f(gh) = f(g)f(h) = f(g)e = f(g)$. 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.

7.4. 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 $g = g_1 g_2 \dots g_k$, 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 $g = g_1 g_2 \dots g_k \in G$, где $g_i \in G_1 \cup G_2$, в виде $g = g_{s_1} g_{s_2} \dots g_{s_n}$, где $g_{s_1} \dots g_{s_k} \in G_1$ и $g_{s_{k+1}} \dots g_{s_n} \in G_2$. *Лемма 7.4.1* утверждает, что это представление однозначно.
- *Лемма 7.4.1.* 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 $g_1^{-1} g'_1 = g_2^{-1} g'_2 = e$, т.к. $G_1 \cap G_2 = e$. Следовательно, $g_1 = g'_1$ и $g_2 = g'_2$.
- *Теорема 7.4.1.* Группа G является прямым произведением своих подгрупп G_1 и G_2 тогда и только тогда, когда выполняются условия:
 1. Пересечение подгрупп $G_1 \cap G_2 = e$.
 2. Любой элемент $g \in G$ однозначно представим в виде произведения элементов $g = g_1 g_2 \dots g_k$, 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 1–3 выполняются.
 (\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 $g_1 g_2 \in G_2 g_1$, откуда $g_1 g_2 = g_2 g_1$. Также $g_1 g_2 \in G_1 g_2$ 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 $G_1 \cap G_2 = e$, то из *леммы 7.4.1* следует, что $g_2 = g'_2$ и $g'_1 = g_1$. Следовательно, $g_1 g_2 = g_2 g_1$.

- **Утверждение 7.4.1.** Пусть G — конечная абелева группа порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, где p_1, p_2, \dots, p_k — простые различные числа. Множество $A(p) = \{x \in G \mid |x| = p^\alpha\}$, где α принимает произвольные целые значения, является подгруппой и называется примарной подгруппой группы G , соответствующей простому числу p .
- **Теорема 7.4.2.** Всякая конечная абелева группа G разлагается в прямое произведение своих примарных подгрупп $A(p_1), A(p_2), \dots, A(p_k)$, где $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$.

Доказательство — индукция по числу простых в разложении порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$. Очевидно, что для $|G| = p^\alpha$ справедливо, т.к. 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 $\exists x \in A(p) \times A(q)$ 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 $\lambda_1 \leq \alpha$, $\lambda_2 \leq \beta$. Так как $(p^{\lambda_1}, q^{\lambda_2}) = 1$ — взаимно простые, то существует представление $p^{\lambda_1} \cdot m + q^{\lambda_2} \cdot n = 1$ (см. п.8.1 алгоритм Евклида), где m, n — целые. Тогда $x^{p^{\lambda_1} \cdot m + q^{\lambda_2} \cdot n} = x$ или $x = y \cdot z$, где $y = x^{q^{\lambda_2} \cdot n}$ и $z = x^{p^{\lambda_1} \cdot m}$, для которых $y^{p^{\lambda_1}} = (x^{p^{\lambda_1} q^{\lambda_2}})^n = e$ и $z^{q^{\lambda_2}} = (x^{p^{\lambda_1} q^{\lambda_2}})^m = e$. Проверить, если $y^N = e$, то порядок $|y|$ делит N . Отсюда $y \in A(p)$ и $z \in A(q)$.

Пусть разложение верно для $l < k$, т.е. для $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_l^{\alpha_l}$. Рассмотрим группу порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, тогда возможно прямое разложение $G = A(p_1, p_2, \dots, p_{k-1}) \times A(p_k)$, где $A(p_1, p_2, \dots, p_{k-1}) = \{x \in G \mid |x| = p_1^{\gamma_1} p_2^{\gamma_2} \dots p_{k-1}^{\gamma_{k-1}}, \gamma_i \leq \alpha_i\}$ — это доказывается как и для случая $|G| = p^\alpha q^\beta$. Имеем: группа $A(p_k)$ — примарная, группа $A(p_1, p_2, \dots, p_{k-1})$ по предположению индукции разлагается в прямое произведение своих примарных подгрупп; теорема доказана.

- **Утверждение 7.4.2.** Порядок конечной примарной группы (подгруппы) $A(p) = \{x \mid x \setminus = p^\alpha\}$ равен $|A(p) \setminus = p^n$, где n — некоторое положительное целое; α принимает произвольные целые значения.

Доказательство. Рассмотрим следующее разложение примарной группы. Пусть $A_1 = \{a, a^2, \dots, a^{p-1} = e\} \subseteq A(p)$ — циклическая подгруппа максимального порядка $|A_1| = p^{\alpha_1}$. По теореме Лагранжа (см. теорему 7.3.1) $|A(p)| = |A_1| \cdot |A(p)/A_1|$, где $A(p)/A_1 = \{x_1A, x_2A, \dots, x_kA\}$ — факторгруппа по подгруппе A_1 . Ясно, что $\forall x_i A \in A(p)/A_1 (x_i A)^{p^{\alpha_1}} = e$ и, значит, $B(p) = A(p)/A_1$ — примарная группа, которая вновь допускает разложение на смежные классы по циклической подгруппе A_2 с $B(p)$ максимального порядка, т.е. $|A(p)| = |A_1| \cdot |A_2| \cdot |B(p)/A_2|$ где $|A_2| = p^{\alpha_2}$. Исходная примарная группа $A(p)$ конечного порядка и, следовательно, за конечное число m шагов получим разложение $|A(p)| = |A_1| \cdot |A_2| \cdot \dots \cdot |A_m|$, или $|A(p)| = p^{\alpha_1} p^{\alpha_2} \dots p^{\alpha_m} = p^n$, где $|A_i| = p^{\alpha_i}$.

- **Лемма 7.4.2.** Пусть $A(p)$ — примарная группа (подгруппа), A с $A(p)$ — циклическая подгруппа максимального порядка $|A \setminus = p^n$ с образующим элементом a , $A(p)/A$ — факторгруппа, $y \in A(p)/A$ — класс смежности порядка $|y \setminus = p^\alpha$, т.е. $\bar{y}^{p^\alpha} = \bar{e} = A$, тогда существует элемент $u \in y$ того же порядка $|u \setminus = p^\alpha$.

Доказательство. Так как $\bar{y}^{p^\alpha} = A$, то для любого $u \in y$ выполняется $u^{p^\alpha} \in A$, где A — единица факторгруппы $A(p)/A$. Следовательно, $u^{p^\alpha} = a^c = a^{c_1 \cdot p}$, где $(c_1, p) = 1$ и $p < n$. Положим $z = a^{c_1}$ — образующий элемент группы A , тогда $u^{p^\alpha} = z^{p^\alpha}$ и $(u^{p^\alpha})^{p^{n-\beta}} = (z^{p^\alpha})^{p^{n-\beta}} = z^{p^{\alpha+n-\beta}} = e$ или $u^{p^{\alpha+n-\beta}} = e$. Так как z — образующий элемент A , то $p^{\alpha+n-\beta} \leq p^n$ — порядку $|A \setminus = p^n$, порядок $|u \setminus \leq p^n$. Отсюда $p^{\alpha+n-\beta} \leq p^n$ или $\alpha < p$. Теперь равенство $u^{p^\alpha} = z^{p^\alpha}$ можно записать в виде $u^{p^\alpha} = z^{p^\alpha} = (z^{p^{\beta-\alpha}})^{p^\alpha}$, а так как $z_1 = z^{p^{\beta-\alpha}} \in A$, то $u^{p^\alpha} = z_1^{p^\alpha}$ или $(uz^{-1})^{p^\alpha} = e$. Таким образом, элемент $y_1 = uz^{-1} \in y$, порядок которого $|y_1 \setminus = p^\alpha$.

- **Теорема 7.4.3.** 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 $A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \dots \times C(p^{\lambda_m})$, где $C(p^{\lambda_i})$ — циклическая подгруппа порядка p^{λ_i} , $\lambda_1 < \lambda_2 < \dots < \lambda_m$ и $\lambda_1 + \lambda_2 + \dots + \lambda_m = n$.

Доказательство — индукция по числу л. Для $n = 1$ теорема верна, т.к. (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 с $A(p)$ — циклическая подгруппа максимального порядка $|A| = p^\alpha$. Рассмотрим факторгруппу $A(p)/A = \{x_1A, x_2A, \dots, x_kA\}$. Данная группа является примарной порядка $|A(p)/A| < p^n$, т.к. $|A(p) \setminus A| = |A| \cdot |A(p)/A|$. Предположение индукции позволяет записать для нее разложение $A(p)/A = C(p^{\alpha_1}) \times C(p^{\alpha_2}) \times \dots \times C(p^{\alpha_k})$. Обозначим через $z_i \in C(p^{\alpha_i})$ образующие циклических подгрупп. Лемма 7.4.2 утверждает, что существует $y_i \in \bar{z}_i$ и $|y_i| = |z_i|$ и можно положить $z_i = y_iA$. Из прямого разложения факторгруппы следует, что любой класс смежности $x \in A(p)/A$ можно представить в виде $x = \bar{z}_1^{t_1} \bar{z}_2^{t_2} \dots \bar{z}_k^{t_k}$, или $x = y_1^{t_1} y_2^{t_2} \dots y_k^{t_k} A$, где $0 \leq t_i \leq \alpha_i$. Так как $A(p) = x_1A \cup x_2A \cup \dots \cup x_kA$, то произвольный элемент $x \in A(p)$ представим как $x = y_1^{t_1} y_2^{t_2} \dots y_k^{t_k} a^i$. Таким образом, искомое прямое разложение на циклические подгруппы найдено.

Покажем, что представление $A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \dots \times C(p^{\lambda_m})$ единственно. Доказательство единственности — индукция по числу n . Для $n = 1$ имеем $A(p) = C(p)$ — свойство выполняется. Пусть оно верно для всех $k < n$. Покажем, что верно и для групп порядка p^n . Предположим существование другого разложения группы $A(p) = C(p^{\beta_1}) \times C(p^{\beta_2}) \times \dots \times C(p^{\beta_l})$, где $\beta_1 < \beta_2 < \dots < \beta_l$. Ясно, что порядок $|A(p)| = p^{\lambda_1} \cdot p^{\lambda_2} \cdot \dots \cdot p^{\lambda_m} = p^{\beta_1} \cdot p^{\beta_2} \cdot \dots \cdot p^{\beta_l} = p^n$.

Запишем разложение группы в уточненном виде

$$A(p) = C(p^{\lambda_1}) \times C(p^{\lambda_2}) \times \dots \times C(p^{\lambda_m}) \times \underbrace{C(p) \times C(p) \times \dots \times C(p)}_{m-t_1} \quad \text{и}$$

$$A(p) = C(p^{\beta_1}) \times C(p^{\beta_2}) \times \dots \times C(p^{\beta_l}) \times \underbrace{C(p) \times C(p) \times \dots \times C(p)}_{l-t_2}$$

Рассмотрим группу (подгруппу) $[A(p)]^p = \{x^p \mid x \in A(p)\}$. Для этой группы справедливы разложения:

$$[A(p)]^p = C(p^{\lambda_1-1}) \times C(p^{\lambda_2-1}) \times \dots \times C(p^{\lambda_{t_1}-1}) \quad (*)$$

$$[A(p)]^p = C(p^{\beta_1-1}) \times C(p^{\beta_2-1}) \times \dots \times C(p^{\beta_{l_2}-1}), \quad (**)$$

где $C(p^\alpha) = \{x, x^2, \dots, x^{p^\alpha} = e\}$ и $[C(p^\alpha)]^p = \{x^p, x^{2p}, \dots, x^{p^\alpha} = e\}$.

Порядок $|[A(p)]^p| < |A(p)|$ и, следовательно, по предположению индукции разложения (*) и (**) совпадают, т.е. $l_1 = \beta_1$, $l_2 = \beta_2, \dots, \lambda_{t_1} = \beta_{l_2}, t_1 = t_2$, 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 $G = C(2) \times C(3) \times C(7)$.

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 $G = \{x, y, xy, e\} = \{x, x^2 = e\} \times \{y, y^2 = e\}$.

7.5. Строеие некоммутативных групп

- *Определение.* Пусть G — конечная группа. Подгруппа $H \subset G$ называется p -подгруппой, если порядок ее $|H| = p^\alpha$.
- *Определение.* p -подгруппа называется *силовой*, если порядок ее p^α имеет максимальную степень в разложении порядка групп G .
- *Теоремы 7,5.1(Силова).* Пусть G — конечная группа порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, где p_i — простые числа.
 1. Для каждого p_i существует силовая подгруппа группы G .
 2. Всякая p -подгруппа группы G содержится в некоторой силовой подгруппе.
 3. Все силовые подгруппы сопряжены, т.е. если H, P — силовые подгруппы, то существует такое $t \in G$, что $H = tPt^{-1}$.
 4. Количество силовских p -подгрупп равно $n_p \cdot k + 1$, где k — некоторое целое.

Пример. Пусть G — группа порядка $|G| = 28 = 2^2 \cdot 7^1$, тогда существуют силовые подгруппы H_1 , $|H_1| = 4$ и H_2 , $|H_2| = 7$.

7.6. 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 $\{1, 2, \dots, m\}$. Каждое такое отображение $\lambda : S \rightarrow S$ называется *подстановкой* или *перестановкой* и записывается $\lambda = \begin{pmatrix} 1 & 2 & \dots & m \\ \pi_1 & \pi_2 & \dots & \pi_m \end{pmatrix}$, где

$\pi_i = \lambda(i)$ — образ элемента $i = \overline{1, m}$. Произведением подстановок является композиция отображений (*операция группы*) $(\lambda\sigma)(i) = \sigma(\lambda(i))$.

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 $m > 3$. Порядок данной группы $|S_m| = m!$ — количество всех перестановок из m элементов. Единичная (тождественная) подстановка обозначается $e = \begin{pmatrix} 1 & 2 & \dots & m \\ 1 & 2 & \dots & m \end{pmatrix}$, которая удовлетворяет $\forall \lambda \in S_m$

$\lambda e = e\lambda = \lambda$. Обратной к $\lambda = \begin{pmatrix} 1 & 2 & \dots & m \\ \pi_1 & \pi_2 & \dots & \pi_m \end{pmatrix}$ является подстановка

$\lambda^{-1} = \begin{pmatrix} \pi_1 & \pi_2 & \dots & \pi_m \\ 1 & 2 & \dots & m \end{pmatrix}$, для которой верно, что $\lambda\lambda^{-1} = \lambda^{-1}\lambda = e$.

- **Утверждение 7.6.1.** Симметрическая группа S_2 степени 2 — абелева.
- **Определение.** Подстановка λ , перемещающая элементы i_1, i_2, \dots, i_k так, что $\lambda(i_1) = i_2, \lambda(i_2) = i_3, \dots, \lambda(i_k) = i_1$ и оставляющая на месте остальные элементы, называется циклом длины k и обозначается (i_1, i_2, \dots, i_k) .
- **Равносильное определение.** Подстановка называется циклической, если каждый из ее действительно перемещаемых элементов i_1, i_2, \dots, i_k можно перевести в любой другой (из действительно перемещаемых элементов), если подстановку применить достаточное число раз. Например, $\lambda(i_1) = i_2, \lambda^2(i_1) = \lambda(\lambda(i_1)) = \lambda(i_2) = i_3, \dots, \lambda^k(i_1) = i_1$. Теперь цикл можно записать: $(i_1, i_2, \dots, i_k) = (i_1, \lambda(i_1), \lambda^2(i_1), \dots, \lambda^{k-1}(i_1))$.

Пример. $\left(\begin{array}{cccccc} 1 & 4 & 5 & 6 & 7 & 3 \end{array} \right) = (3, 2, 1, 4, 5, 6, 7)$ — 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 7.6.1.** Каждую нетождественную подстановку S_m можно разложить единственным образом в произведение независимых циклов.

Доказательство. Пусть $\alpha \in S_m$ 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 $S = S_1 \cup S_2 \cup \dots \cup S_r$. Каждый элемент $i \in S$ принадлежит одному и только одному классу S_i , причем множество S_i состоит из образов элемента i при действии степеней подстановки α : $S_i = \{i, \alpha(i), \alpha^2(i), \dots, \alpha^{k_i-1}(i)\}$, где k_i — количество элементов в S_i . Множества S_i еще называют л-орбитами. Выберем в каждом классе S_i по одному представителю i_i и поставим ему в соответствие цикл $\pi_i = \{i_i, \alpha(i_i), \alpha^2(i_i), \dots, \alpha^{k_i-1}(i_i)\}$. Так как любой элемент, не принадлежащий S_i , остается на месте при действии степеней π_i , то перестановка α есть произведение независимых циклов $\alpha = \pi_1 \pi_2 \dots \pi_r$.

- **Замечание 1.** Если цикл $\pi_i = (i_i)$ имеет длину 1, то он действует как тождественная подстановка. Такие циклы в записи $\alpha = \pi_1 \pi_2 \dots \pi_r$ можно опускать.
- **Замечание 2.** Независимые циклы в записи $\alpha = \pi_1 \pi_2 \dots \pi_r$ можно произвольным образом переставлять между собой. Так, $\alpha = \pi_{k_1} \pi_{k_2} \dots \pi_{k_r}$, где $\{\pi_{k_1} \pi_{k_2} \dots \pi_{k_r}\} \equiv \{\pi_1, \pi_2, \dots, \pi_r\}$.

Пример. $\left(\begin{array}{cccccc} 12 & 3 & 4 & 5 & 6 \\ 3 & & & & \end{array} \right) = (1345)(2)(6) = (1345),$

$$\left(\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 3 & 2 & 1 & 5 & 4 & 6 \end{array} \right) = (13)(2)(45)(6) = (13)(45) = (45)(13).$$

- *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

$$\operatorname{sgn}(\pi) = \begin{cases} +1, & \text{если } \pi \text{ - четная подстановка,} \\ -1, & \text{если } \pi \text{ - нечетная подстановка,} \end{cases}$$

где $\pi \in S_m$, тогда $\operatorname{sgn}(\pi) = (-1)^d$.

Например, для подстановки $\pi = [1\ 2\ 3\ 5\ 4\ 6] = (132)(45)$ 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 7.6.2.** При умножении подстановки $\pi \in S_m$ на транспозицию $\tau = (\alpha\beta)$ она меняет свою четность.

Доказательство. Пусть $\pi = (i_1 i_2 \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)$ — разложение подстановки в произведении 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. $\alpha \in \pi, \beta \in \pi$, α и β принадлежат одному и тому же циклу.
5. $\alpha \in \pi, \beta \in \pi$, α и β принадлежат разным циклам.

Пусть k — число действительно перемещаемых элементов π ; l — число независимых циклов в разложении π . Декремент подстановки $d(\pi) = k - l$.

1. Пусть $\alpha \notin \pi, \beta \notin \pi$, тогда $\pi\tau = (i_1 i_2 \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)(\alpha\beta)$. 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 $\pi\tau = (\alpha i_2 \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)(\alpha\beta) = (\alpha i_2 \dots i_r \beta)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = (k + 1) - l = k - l + 1$.
3. Случай $\alpha \notin \pi, \beta \in \pi$ рассматривается подобно случаю 2.
4. Пусть $\alpha \in \pi, \beta \in \pi$, α и β принадлежат одному и тому же циклу. Тогда $\pi\tau = (\alpha i_2 \dots i_m \beta i_{m+2} \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)(\alpha\beta) = (\alpha i_2 \dots i_m)(\beta i_{m+2} \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = k - (l + 1) = k - l - 1$.
5. Пусть $\alpha \in \pi, \beta \in \pi$, α и β принадлежат разным циклам. Тогда $\pi\tau = (\alpha i_2 \dots i_r)(\beta j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)(\alpha\beta) = (\alpha i_2 \dots i_r \beta j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)$. Декремент $d(\pi\tau) = k - (l - 1) = k - l + 1$.

Таким образом, во всех случаях $\operatorname{sgn}(\pi\tau) = (-1)^{k-l\pm 1} = (-1)^{\pm 1}(-1)^{k-l} = -\operatorname{sgn}(\pi)$ четность подстановки $\pi\tau$ меняется.

- **Теорема 7.6.3.** 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. $\pi = (i_1 i_2 \dots i_r)(j_1 j_2 \dots j_s) \dots (l_1 l_2 \dots l_p)$ — разложение подстановки в произведение независимых циклов. Каждый цикл разлагается в произведение транспозиций $(i_1 i_2 \dots i_r) = (i_1 i_2)(i_1 i_3) \dots (i_1 i_r)$. Таким способом можно разложить в произведение транспозиций все циклы подстановки $\pi = \tau_1 \tau_2 \dots \tau_n$, где τ_i — транспозиции. Теорема 7.6.2 позволяет записать $\operatorname{sgn}(\pi) = \operatorname{sgn}(\tau_1 \tau_2 \dots \tau_n) = \operatorname{sgn}((\tau_1 \tau_2 \dots \tau_{n-1}) \tau_n) = \operatorname{sgn}(\tau_1 \tau_2 \dots \tau_{n-1}) \cdot (-1) = \operatorname{sgn}(\tau_1 \tau_2 \dots \tau_{n-2}) \cdot (-1) \cdot (-1) = \dots = \operatorname{sgn}(\tau_1) \cdot (-1)^{n-1} = (-1)^n$.

Таким образом, четность подстановки $\operatorname{sgn}(\pi) = (-1)^n$ совпадает с четностью числа транспозиций n в разложении $\pi = \tau_1 \tau_2 \dots \tau_n$.

- **Следствие 1.** $\operatorname{sgn}(\sigma\tau) = \operatorname{sgn}(\sigma) \cdot \operatorname{sgn}(\tau)$, σ, τ — произвольные подстановки.

Доказательство. Пусть $\sigma = s_1 s_2 \dots s_r$ и $\tau = t_1 t_2 \dots t_l$ — разложения в произведение транспозиций s_i, t_j . Тогда $\sigma\tau = s_1 s_2 \dots s_r t_1 t_2 \dots t_l$ — разложение в произведение транспозиций. Из теоремы $\operatorname{sgn}(\sigma\tau) = (-1)^{r+l} = (-1)^r (-1)^l = \operatorname{sgn}(\sigma) \cdot \operatorname{sgn}(\tau)$.

- **Следствие 2.** $\operatorname{sgn}(\sigma^{-1}) = \operatorname{sgn}(\sigma)$, σ^{-1} — обратная подстановка.

Доказательство. Пусть $\sigma = s_1 s_2 \dots s_r$, тогда $\sigma^{-1} = s_r \dots s_2 s_1$, так как $\sigma\sigma^{-1} = (s_1 s_2 \dots s_r)(s_r \dots s_2 s_1) = (s_1 (s_2 (\dots (s_r s_r) \dots) s_2) s_1) \neq e$ и $(s_i s_i) = e$ — тождественная подстановка, где s_i — транспозиции. Первое следствие позволяет записать $\operatorname{sgn}(\sigma\sigma^{-1}) = \operatorname{sgn}(\sigma) \cdot \operatorname{sgn}(\sigma^{-1})$. С другой стороны, $\sigma\sigma^{-1} = e$ — тождественная подстановка и $\operatorname{sgn}(e) = 1$. Тогда $\operatorname{sgn}(\sigma) \cdot \operatorname{sgn}(\sigma^{-1}) = 1$ и, следовательно, $\operatorname{sgn}(\sigma^{-1}) = \operatorname{sgn}(\sigma)$.

- **Теорема 7.6.4.** Число четных подстановок A_m в 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 τ — произвольная фиксированная транспозиция. Рассмотрим отображение $\varphi: S_m \rightarrow S_m$, 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 φ верно, что $\forall a_1 \neq a_2 \in A_m \varphi(a_1) \neq \varphi(a_2)$, в противном случае $a_1\tau = a_2\tau$. Отсюда, умножая справа последнее равенство на τ , 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 φ является взаимно однозначным.

- *Утверждение 7.6.2* A_m — подгруппа симметрической группы S_m .
- *Теорема 7.6.5* Кэли. Всякая конечная группа G порядка m изоморфна некоторой подгруппе симметрической группы S_m .

Доказательство. Для любого элемента $a \in G$ рассмотрим отображение $L_a : G \rightarrow G$, состоящее в умножении всех элементов $G = \{g_1, g_2, \dots, g_m\}$ 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: $(L_a L_b)(g) = L_b(L_a(g)) = b(ag) = (ab)g = L_{ab}(g)$, т.е. $L_a L_b = L_{ab}$. Отсюда следует, что множество $H = \{L_{g_1}, L_{g_2}, \dots, L_{g_m}\}$ образует подгруппу в множестве всех взаимно однозначных отображений G в себя, т.е. в симметрической группе S_m . Тогда отображение $\varphi : G \rightarrow H \subset S_m$ такое, что $\forall a \in G \varphi(a) = L_a$ есть изоморфизм, поскольку φ — взаимно однозначное и выполняется свойство $\varphi(ab) = L_{ab} = L_a L_b = \varphi(a)\varphi(b)$ сохранения операций.

7.7. Действие групп на множестве

- *Определение.* Говорят, что задано действие группы G на множестве $S = \{1, 2, \dots, m\}$, если определен гомоморфизм Γ группы G в симметрическую группу S_m подстановок: $T : G \rightarrow S_m$. Свойство сохранения операций для гомоморфизма: $\forall g_1, g_2 \in G T(g_1 \circ g_2) = T(g_1) \cdot T(g_2)$, где

$$T(g_i) = \begin{pmatrix} 1 & 2 & \dots & m \\ \alpha_1 & \alpha_2 & \dots & \alpha_m \end{pmatrix} \in S_m.$$

Далее полагаем, что $g_i = \begin{pmatrix} 1 & 2 & \dots & m \\ \alpha_1 & \alpha_2 & \dots & \alpha_m \end{pmatrix}$

- *Замечание.* Чаще всего бывает так, что действие G на S возникает естественным образом, как группа симметрии структуры, определенной на S .

Пример. Пусть $S = \{1, 2, 3, 4, 5\}$ — вершины графа на рис. 7.1. Найти G — группу самосовмещений данного графа.

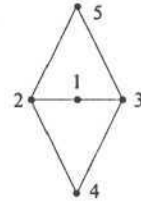


Рис. 7.1

Решение. 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;

$a = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix} = (45)$ — 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. В табл. 7.1 содержатся все возможные произведения элементов рассматриваемой группы. Из табл. 7.1 видно, что G — коммутативная группа и $G = \{e, a\} \times \{e, b\}$.

Таблица 7.1

	e	a	b	ab
e	e	a	b	ab
a	a	e	ab	b
b	b	ab	e	a
ab	ab	b	a	e

- *Определение.* 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. $g = \begin{pmatrix} s_1 & \dots & s_m \\ \alpha_1 & \alpha_2 & \dots & \alpha_m \end{pmatrix}$ или

более короткая запись этого $gs_1 = s_2$.

- *Утверждение 7.7.1.* Определенная *g-эквивалентность* на множестве S является отношением эквивалентности.

Доказательство. 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. $\forall s_1, s_2 \in S \quad s_1 \sim s_2 \rightarrow s_2 \sim s_1$.

Имеем $s_1 \sim s_2$, тогда $\exists g \in G \quad gs_1 = s_2$ и $s_2 = g^{-1}s_1$, а значит, $s_2 \sim s_1$.

3. $\forall s_1, s_2, s_3 \in S \quad s_1 \sim s_2 \wedge s_2 \sim s_3 \rightarrow s_1 \sim s_3$.

Имеем, что $\exists t, g \in G$ $ts_1 = s_2$ и $gs_2 = s_3$, откуда $(tg)s_1 = g(ts_1) = gs_2 = s_3$. Следовательно, $s_1 \sim s_3$.

- **Утверждение 7.7.2.** Группа $G = \{g_1, g_2, \dots, g_k\}$, действуя на множестве S , порождает его разбиение на непересекающиеся подмножества — классы эквивалентности (рис. 7.2):

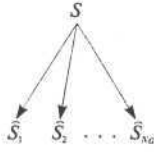


Рис. 7.2

$S = \bar{S}_1 \cup \bar{S}_2 \cup \dots \cup \bar{S}_{N_G}$, где $\forall s_1, s_2 \in \bar{S}_\alpha \exists g \in G$
 $gs_1 = s_2$ и $\forall s_1 \in \bar{S}_\alpha \forall s_2 \in \bar{S}_\beta \forall g \in G$ $gs_1 \neq s_2$,
 N_G — количество классов эквивалентности.

Пусть $s_1 \in \bar{S}_\alpha$, тогда класс эквивалентности \bar{S}_α составят все различные элементы множества $\bar{S}_\alpha = \{g_1s_1, g_2s_1, \dots, g_k s_1\}$.

- **Определение.** Множество $Z(s_1) = \{g \in G \mid gs_1 = s_1, s_1 \in S\}$ называется стабилизатором s_1 в S . Элементы стабилизатора оставляют s_1 на месте.

Пример. Продолжим рассмотрение примера на рис. 7.1. $S = \{1, 2, 3, 4, 5\}$ — вершины графа на рис. 7.3. На S действует группа самосовмещений $G = \{e, a, b, ab\}$, где

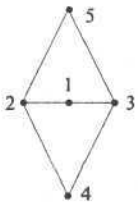


Рис. 7.3

$$e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}, \quad a = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 5 & 4 \end{pmatrix},$$

$$b = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 4 & 5 \end{pmatrix}, \quad ab = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 3 & 2 & 5 & 4 \end{pmatrix}.$$

Найдем все классы эквивалентности.

$$\bar{S}_1 = \{1\}, \quad a(1), b(1), ab(1) = \{1, 1, 1, 1\} = \{1\},$$

$$\bar{S}_2 = \{e(2), a(2), b(2), ab(2)\} = \{2, 2, 3, 3\} = \{2, 3\},$$

$$\bar{S}_3 = \{e(4), a(4), b(4), ab(4)\} = \{4, 5, 4, 5\} = \{4, 5\}.$$

$S = \bar{S}_1 \cup \bar{S}_2 \cup \bar{S}_3$ — всего три класса эквивалентности.

Определим стабилизаторы для вершин графа $\{1, 2, 3, 4, 5\}$.
 $Z(1) = \{e, a, b, ab\}$, $Z(2) = \{e, a\}$, $Z(3) = \{e, a\}$, $Z(4) = \{e, b\}$, $Z(5) = \{e, b\}$.

- **Утверждение 7.7.3.** $Z(s_1)$ с G — подгруппа группы G .

Доказательство. Проверим свойства группы.

1. **Замкнутость.** $\forall g_1, g_2 \in Z(s_1)$ $g_1s_1 = s_1, g_2s_1 = s_1$, тогда и $g_1g_2s_1 = g_1(s_1) = s_1$, следовательно, $g_1g_2 \in Z(s_1)$.
2. **Единичный элемент** $e \in Z(s_1)$, так как $es_1 = s_1$.
3. **Обратный элемент.** Пусть $g \in Z(s_1)$, тогда $gs_1 = s_1$, откуда $s_1 = g^{-1}gs_1$, следовательно, $g^{-1} \in Z(s_1)$.

- *Утверждение 7.7.4.* $\forall s_1, s_2 \in \widehat{S}_\alpha \quad |Z(s_1)| = |Z(s_2)|$ и $\exists t \in G \quad Z(s_1) = tZ(s_2)t^{-1}$ — в этом случае говорят, что подгруппы $Z(s_1), Z(s_2)$ сопряжены.

Доказательство. Имеем $s_1, s_2 \in \widehat{S}_\alpha$, следовательно, $\exists t \in G \quad ts_1 = s_2$. $\forall g \in Z(s_2) \quad (tg)s_1 = g(ts_1) = gs_2 = s_2 = ts_1$ 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 $\forall g \in Z(s_2) \quad tgt \in Z(s_1)$. Заметим, что $\forall g_1 \neq g_2 \in Z(s_2) \quad tg_1t^{-1} \neq tg_2t^{-1}$, значит, $|Z(s_2)| = |tZ(s_2)t^{-1}| < |Z(s_1)|$ или $|Z(s_2)| < |Z(s_1)|$. Подобным образом доказывается в обратную сторону: $|Z(s_2)| \geq |Z(s_1)|$, следовательно, $|Z(s_1)| = |Z(s_2)|$. Показали, что $\exists t \in G \quad \forall g \in Z(s_2) \quad tgt^{-1} \in Z(s_1)$ и $|Z(s_1)| = |Z(s_2)| = |tZ(s_2)t^{-1}|$, отсюда $Z(s_1) = tZ(s_2)t^{-1}$.

- *Утверждение 7.7.5.* $|\widehat{S}_\alpha| = \frac{|G|}{|Z(s_1)|}$, где $s_1 \in \widehat{S}_\alpha$.

Доказательство. Пусть $G = \{g_1, g_2, \dots, g_k\}$. Из утверждения 7.7.2 следует, что $\widehat{S}_\alpha = \{g_1s_1, g_2s_1, \dots, g_ks_1\}$, однако среди выписанных элементов множества \widehat{S}_α могут встречаться одинаковые. Назовем $g_i, g_{i_2} \in G$ эквивалентными $g_i \sim g_{i_2}$, если они действуют на элемент s_1 одинаковым образом, т.е. $g_i s_1 = g_{i_2} s_1$. Данное отношение является отношением эквивалентности. Введенная эквивалентность разбивает группу G на классы, количество которых равно числу различных элементов среди выписанных $\widehat{S}_\alpha = \{g_1s_1, g_2s_1, \dots, g_ks_1\}$, т.е. равно $|\widehat{S}_\alpha|$.

Пусть $g_{i_1}s_1 = g_{i_2}s_1 \quad (g_{i_1} \sim g_{i_2})$ или $fe_{i_1} \wedge_{s_1}^1 * i = ? r_2^1 fe_{i_2} \wedge_{s_1}^1 = = g_{i_2}^{-1}(g_{i_1}s_1) = fe_{i_2} \wedge_{s_1}^1 s_1 = s_1$, 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, $g_i \sim g_{i_2}$ тогда и только тогда, когда g_i, g_{i_2} лежат в одном правом классе смежности по стабилизатору $Z(s_1)$ ($g_i \in Z(s_1)g_{i_2}$), а значит, и число элементов в \widehat{S}_α равно количеству правых смежных классов в G по подгруппе $Z(s_1)$. Согласно *теореме 7.3.1 Лагранжа*, $|\widehat{S}_\alpha| = \frac{|G|}{|Z(s_1)|} = |G:Z(s_1)|$.

- *Лемма 7.7.1 Бернсайда.* Число классов \widehat{S}_α эквивалентности, на которые распадается множество $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \dots \cup \widehat{S}_{N_G}$ под действием группы G , равно $N_G = \frac{1}{|G|} \sum_{g \in G} |\psi(g)|$, где $\psi(g) = \{s \in S \mid gs = s\}$.

Доказательство. Подсчитаем двумя различными способами множество всех таких пар элементов, что $\forall g \in G \forall s \in S \{(g, s) \mid gs = s\}$. Это можно выполнить следующим образом:

$$\sum_{s \in S} |Z(s)| = \sum_{g \in G} |\psi(g)|.$$

Первая сумма $\sum_{s \in S} |Z(s)| = \sum_{i=1}^{N_G} \sum_{s \in \widehat{S}_i} |Z(s)|$. Так как $\forall s_1, s_2 \in S_\alpha$ $|Z(s_1)| = |Z(s_2)|$, то $\sum_{s \in S_\alpha} |Z(s)| = |S_\alpha| \cdot |Z(s_\alpha)| = \frac{|G|}{|Z(\widehat{S}_\alpha)|} |Z(s_\alpha)| = |G|$ где $s_\alpha \in \widehat{S}_\alpha$. Таким образом, $\sum_{s \in S} |Z(s)| = \sum_{i=1}^{N_G} \sum_{s \in \widehat{S}_i} |Z(s)| = \sum_{i=1}^{N_G} |G| = N_G \cdot |G|$.

Получили, что $N_G \cdot |G| = \sum_{g \in G} |\psi(g)|$, откуда $N_G = \frac{1}{|G|} \sum_{g \in G} |\psi(g)|$.

Пример. Продолжим рассмотрение примера на рис. 7.3. $S = \{1, 2, 3, 4, 5\}$ — вершины графа. На S действует группа самосовмещений $G = \{e, a, b, ab\}$, где $e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, $a = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 4 & 5 \end{pmatrix}$, $b = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 5 & 4 \end{pmatrix}$, $ab = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 5 & 4 \end{pmatrix}$.

Полным перебором установили, что под действием G множество S распадается на три класса эквивалентности: $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \widehat{S}_3$, где $S_1 = \{1\}$, $S_2 = \{2, 3\}$, $S_3 = \{4, 5\}$. Установим данный факт, применяя лемму 7.7.1 Бернсайда: $N_G = \frac{1}{|G|} \sum_{g \in G} |\psi(g)|$ — 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 пересчета Пойа. Это позволит нам с более формальных позиций подойти к пониманию самой техники подсчета и к применению ее для решения задач.

7.8. Цикловой индекс группы

Пусть группа G действует на множестве $S = \{1, 2, \dots, m\}$ и $T: G \rightarrow S_m$ — гомоморфизм в симметрическую группу S_m . Рассмотрим разложение $g \in G$ на независимые циклы

$$g = \left(\begin{array}{cccc} 1 & 2 & \dots & m \\ \alpha_1 & \alpha_2 & \dots & \alpha_m \end{array} \right) = \underbrace{(\beta_1)}_{k_1} \underbrace{(\beta_2)}_{k_2} \dots \underbrace{(\beta_{k_1})}_{k_1} \underbrace{(\beta_{i_1} \beta_{i_2})}_{k_2} \dots \dots \underbrace{(\beta_{j_1} \beta_{j_2} \dots \beta_{j_m})}_{k_m},$$

где k_1 — количество циклов длины 1;

k_2 — количество циклов длины 2;

.....

k_m — количество циклов длины m .

Набор (k_1, k_2, \dots, k_m) называется характеристикой элемента $g \in G$, где $1 \cdot k_1 + 2 \cdot k_2 + \dots + m \cdot k_m = m$.

- **Определение.** Цикловым индексом $Z(G, x_1, x_2, \dots, x_m)$ группы G , действующей на множестве S , называется полином от переменных x_1, x_2, \dots, x_m , определяемый формулой

$$Z(G, x_1, x_2, \dots, x_m) = \frac{1}{|G|} \sum_{g \in G} x_1^{k_1} x_2^{k_2} \dots x_m^{k_m},$$

где (k_1, k_2, \dots, k_m) — характеристика элемента $g \in G$.

Пример. Продолжим рассмотрение примера на рис. 7.3. $S = \{1, 2, 3, 4, 5\}$ — вершины графа. На S действует группа самосовмещений $G = \{e, a, b, ab\}$, где $e = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{pmatrix}$, $a = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 5 & 4 \end{pmatrix}$,

$b = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 4 & 5 \end{pmatrix}$, $ab = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 3 & 2 & 5 & 4 \end{pmatrix}$. Найдём Цикловой индекс группы

G , для этого выполним разложение на независимые циклы подстановок элементов G и установим их характеристики:

$$\begin{aligned}
 e &= (1)(2)(3)(4)(5), (k_1 k_2 \dots k_m) = (5, 0, 0, 0, 0); \\
 a &= (1)(2)(3)(45), (k_1 k_2 \dots k_m) = (3, 1, 0, 0, 0); \\
 b &= (1)(23)(4)(5), (k_1 k_2 \dots k_m) = (3, 1, 0, 0, 0); \\
 ab &= (1)(23)(45), (k_1 k_2 \dots k_m) = (1, 2, 0, 0, 0).
 \end{aligned}$$

$$\begin{aligned}
 Z(G, x_1, x_2, \dots, x_m) &= \frac{1}{4} (r_1^5 r_2^0 r_3^0 r_4^0 r_5^0 + r_1^3 r_2^1 r_3^0 r_4^0 r_5^0 + r_1^3 r_2^1 r_3^0 r_4^0 r_5^0 + \\
 &+ x_1^1 x_2^2 x_3^0 x_4^0 x_5^0) = \frac{1}{4} (x_1^5 + x_1^3 x_2^1 + x_1^3 x_2^1 + x_1^1 x_2^2) = \frac{1}{4} (x_1^5 + 2x_1^3 x_2^1 + x_1^1 x_2^2).
 \end{aligned}$$

7.9. Теория перечисления Пойа

Введем следующие обозначения.

- $D = \{d_1, d_2, \dots, d_m\}$ — конечное множество.
- $G = \{g_1, g_2, \dots\}$ — конечная группа, действующая на D .
- $R = \{r_1, r_2, r_3, \dots\}$ — конечное множество качественных признаков (цвета).
- $S = \{f \mid f: D \rightarrow R\}$ — множество функций; каждая функция/определяет качественные признаки элементов D .
- $\Omega = \{\omega_1, \omega_2, \omega_3, \dots\}$ — множество весов.
- $\omega : R \rightarrow \Omega$ — весовая функция, назначает веса $\omega(r) \in \Omega$ признакам $r \in R$.
- $R(\omega) = \sum_{r \in R} \omega(r)$ — сумма весов элементов $r \in R$ или
- $R(\omega) = \sum_{\omega \in \Omega} C_\omega \omega$, где C_ω — число элементов $r \in R$ с весом $\omega \in \Omega$, множество $\{C_\omega\}$ — перечень R относительно весовой функции $\omega(r)$.
- $R(\omega^2) = \sum_{r \in R} \omega^2(r) = \sum_{\omega \in \Omega} C_\omega \omega^2$.
- $R(\omega^i) = \sum_{r \in R} \omega^i(r) = \sum_{\omega \in \Omega} C_\omega \omega^i$.

Рассмотрим введенные понятия на следующем примере, к которому ниже не раз будем возвращаться.

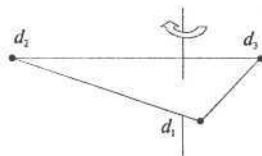


Рис. 7.4

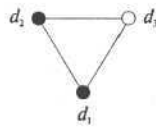
Пусть $D = \{d_1, d_2, d_3\}$ — вершины правильного треугольника (рис. 7.4). Треугольник нанизан на вертикальную ось, вокруг которой он свободно вращается. $R = \{\bullet, \circ\}$ — множество из двух красок. Найти количество различных раскрасок вершин треугольника.

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° , $a^2 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix}$ — поворот на 240° .

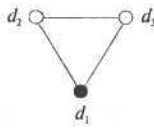
$\Omega = \{x, y, x^{-1}, y^{-1} \text{ и их произведения}\}$ — множество весов.

$S = \{f \mid f: D \rightarrow R\}$ — 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. 7.4). Если допустить вращение, то различные раскраски неподвижного треугольника становятся одинаковыми для вращающегося треугольника. Приведем пример трех раскрасок: f_1, f_2, f_3 .

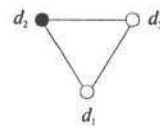
$$\begin{aligned} f_1: D &\rightarrow R, \\ f_1(d_1) &= \bullet, \\ f_1(d_2) &= \bullet, \\ f_1(d_3) &= \circ, \end{aligned}$$



$$\begin{aligned} f_2: D &\rightarrow R, \\ f_2(d_1) &= \bullet, \\ f_2(d_2) &= \circ, \\ f_2(d_3) &= \circ, \end{aligned}$$



$$\begin{aligned} f_3: D &\rightarrow R, \\ f_3(d_1) &= \circ, \\ f_3(d_2) &= \bullet, \\ f_3(d_3) &= \circ. \end{aligned}$$



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 $W(f) = \omega(\bullet)\omega(\bullet)\omega(\circ) = xxy = x^2y$,

$$W(f_2) = \omega(\bullet)\omega(\circ)\omega(\circ) = xy^2,$$

$$W(f_3) = \omega(\circ)\omega(\bullet)\omega(\circ) = yxy = xy^2.$$

- *Определение.* Группа G , действуя на D , индуцирует (наводит, создает, определяет) свое действие на множество функций S . Положим $\forall f \in S \forall g \in G \quad gf \equiv f(g(d))$ — это рассматривается $\forall d \in D$.

В нашем примере рассмотрим действие элемента $a \in G$ на $f_2 \in S$.

$$\begin{aligned} f_2(a^2(d_1)) &= f_2(d_3) = \circ \text{ и } f_3(d_3) = \circ, \\ f_2(a^2(d_2)) &= f_2(d_1) = \bullet \text{ и } f_3(d_2) = \bullet, \\ f_2(a^2(d_3)) &= f_2(d_2) = \circ \text{ и } f_3(d_3) = \circ. \end{aligned}$$

Таким образом, элемент группы a 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 $d \in D$.

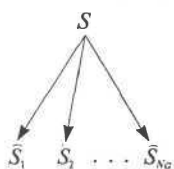


Рис. 7.5

Введенная эквивалентность функций есть отношение эквивалентности, которое порождает разбиение множества элементов/е S на непересекающиеся классы эквивалентности (рис. 7.5): $S = \bar{S}_1 \cup \bar{S}_2 \cup \dots \cup \bar{S}_{N_G}$, где $\forall f_1, f_2 \in \bar{S}_\alpha \exists g \in G gf_1 = f_2$ и $\forall f_1 \in \bar{S}_\alpha \forall f_2 \in \bar{S}_\beta \forall g \in G gf_1 \neq f_2, S_\alpha \neq \bar{S}_\beta, N_G$ —

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

7.7.1 Бернсайда: $N_G = \frac{1}{|G|} \sum_{g \in G} |\psi(g)|$, в данном случае

$$\psi(g) = \{f \in S \mid gf = f \text{ или } f(gd = f(d) \forall d \in D)\}.$$

Вернемся к нашему примеру на рис. 7.4 и найдем для него число N_G . $\psi(e) = \{f \in S \mid ef = f\}$ удовлетворяют все $f \in S$, откуда $|\psi(e)| = 2^3 = 8$. $\psi(a) = \{f \in S \mid af = f\}$ — 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, $N_G = \frac{1}{3}(8 + 2 + 2) = 4$.

- **Утверждение 7.9.1.** Если $f_1 \sim f_2$, то $W(f_1) = W(f_2)$, т.е. эквивалентные функции имеют одинаковые веса.

Доказательство. $D = \{d_1, d_2, d_3, \dots\}$ и $D - \{gd_1, gd_2, gd_3, \dots\}$ — верно $\forall g \in G$, так как G действует на D и $g = \begin{pmatrix} d_1 & d_2 & d_3 & \dots \\ ga_1 & ga_2 & ga_3 & \dots \end{pmatrix}$ — это

подстановка. Теперь $W(f_1) = \prod_{d \in D} \omega(f_1(d)) = \prod_{d \in D} \omega(f_1(gd))$. Имеем

$$f_1 \sim f_2, \text{ тогда } \exists g \in G \quad \forall d \in D \quad \omega(f_1(gd)) = \omega(f_2(d)). \text{ Отсюда}$$

$$W(f_1) = \prod_{d \in D} \omega(f_1(gd)) = \prod_{d \in D} \omega(f_2(d)) = W(f_2).$$

- *Определение.* Последнее утверждение позволяет определить вес каждого класса эквивалентности \widehat{S}_i в разложении $S = S_1 \cup \dots \cup S_{N_G}$, 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 7.9.1 Пойа.* $\sum_{\omega \in \Omega} C_\omega \cdot \omega = Z(G, R(\omega^1), R(\omega^2), \dots, R(\omega^m))$, где

C_ω — число классов эквивалентности $S = \widehat{S}_1 \cup \widehat{S}_2 \cup \dots \cup \widehat{S}_{N_G}$ с весом $\omega \in \Omega$, $Z(G, x_1, x_2, \dots, x_m)$ — цикловой индекс группы G , действующей на множестве $D = \{d_1, d_2, \dots, d_m\}$. Отметим, что $\sum_{\omega \in \Omega} C_\omega = N_G$.

Доказательство. Пусть « Δ » — разбиение множества D на непересекающиеся подмножества:

$$D = \underbrace{D_{11} + D_{12} + \dots + D_{1k_1}}_{k_1} + \underbrace{D_{21} + D_{22} + \dots + D_{2k_2}}_{k_2} + \dots + \underbrace{D_{m1} + D_{m2} + \dots + D_{mk_m}}_{k_m},$$

где $|D_{ij}| = i, |D| = m, 1 \cdot k_1 + 2 \cdot k_2 + \dots + m \cdot k_m = m$.

- *Определение.* Говорят, что $f \in S$ подчинена разбиению « Δ » множества D и записывают $f \in \Delta$, если f постоянна на каждом подмножестве D_{ij} из разбиения: $\forall d \in D_{ij} \quad f(d) = r_{ij}$, где $r_{ij} \in R$.

Функция f , подчиненная разбиению « Δ », взаимно однозначно определяется набором $(\underbrace{r_{11}r_{12} \dots r_{1k_1}}_{k_1}, \underbrace{r_{21}r_{22} \dots r_{2k_2}}_{k_2}, \dots, \underbrace{r_{m1}r_{m2} \dots r_{mk_m}}_{k_m})$, где $r_{ij} \in R$.

Все такие наборы составляют множество:

$$S = \underbrace{S_{11} \times S_{12} \times \dots \times S_{1k_1}}_{k_1} \times \underbrace{S_{21} \times S_{22} \times \dots \times S_{2k_2}}_{k_2} \times \dots \times \underbrace{S_{m1} \times S_{m2} \times \dots \times S_{mk_m}}_{k_m},$$

где $S_{ij} \in R, i = \overline{1, m}, j = \overline{1, k_i}$. Полагая веса элементов $s_{ij} \in S_{ij}$ равными $\omega(s_{ij}) = [\omega(r_{ij})]^i$ и вес $\omega = (s_{11}s_{12} \dots s_{1k_1} s_{21}s_{22} \dots s_{2k_2} \dots s_{m1}s_{m2} \dots s_{mk_m}) \in S$

равным произведению весов $\omega(s) = \prod_{i=1}^m \prod_{j=1}^{k_j} [\omega(s_{ij})] = \prod_{i=1}^m \prod_{j=1}^{k_j} [\omega(r_{ij})]^i$,

имеем $\omega(s) = W(f)$ где $f \in A$ подчинена разбиению.

Для множества S выполнены все условия правила обобщенного произведения, тогда сумма весов элементов данного множества, а значит, и сумма весов всех функций $f \in A$ составит

$$W(S) = \prod_{i=1}^m \prod_{j=1}^{k_j} W(S_{ij}) = \prod_{i=1}^m [W(S_{ij})]^{k_i}, \text{ вследствие } W(S_{ij_1}) = W(S_{ij_2}).$$

$$W(S_{ij}) = \sum_{s \in S_{ij}} \omega(s) = \sum_{r \in R} [\omega(r)]^i = R(\omega^i). \text{ Следовательно,}$$

$$W(S) = \prod_{i=1}^m [W(S_{ij})]^{k_i} = \prod_{i=1}^m [R(\omega^i)]^{k_i} = [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \dots [R(\omega^m)]^{k_m}$$

$$\text{или } \sum_{f \in \Delta} W(f) = [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \dots [R(\omega^m)]^{k_m}.$$

Пусть $g \in G$ и $g = \underbrace{(\beta_1)(\beta_2) \dots (\beta_{k_1})}_{k_1} \underbrace{(\beta_{i_1})(\beta_{i_2}) \dots}_{k_2} \dots \underbrace{(\beta_{j_1} \beta_{j_2} \dots \beta_{j_m})}_{k_m}$ —

разложение на независимые циклы, где (k_1, k_2, \dots, k_m) — характеристика g и $1 \cdot k_1 + 2 \cdot k_2 + \dots + m \cdot k_m = m$. Запишем разложение на независимые циклы в виде

$$g = \underbrace{(D_{11})(D_{12}) \dots (D_{1k_1})}_{k_1} \underbrace{(D_{21})(D_{22}) \dots (D_{2k_2})}_{k_2} \dots \underbrace{(D_{m1})(D_{m2}) \dots (D_{mk_m})}_{k_m},$$

где D_{ij} — цикл длины i , $|D_{ij}| = i$.

Обозначим разбиение множества D на D_u через Δg . Элементы $d \in D_u$ одного цикла при умножении на g переходят последовательно по циклу в элементы того же цикла. Указанное свойство позволяет заключить, что $\forall d_1, d_2 \in D_u \exists k g^k d_1 = d_2$.

Пример. Пусть $g = (12345)(678)$, 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 7.9.2.* Функция $f \in S$ неподвижна относительно $g \in G$ тогда и только тогда, когда $f \in \Delta g$ подчинена разбиению Δg .

Доказательство. (\Rightarrow) Пусть $d_1, d_2 \in D_u$ — принадлежат одному циклу. Следовательно, $\exists k g^k d_1 = d_2$. Тогда $f(d_2) = f(g^k d_1) = f(g(g^{k-1} d_1)) = f(g^{k-1} d_1) = \dots = f(g d_1) = f(d_1)$ 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. $f \in \Delta g$.

(\Leftarrow) Пусть $d \in D_{ij}$, тогда и $gd \in D_{ij}$ — свойство циклов. Имеем $f \in \Delta g$ или $\forall d_1, d_2 \in D_{ij} f(d_1) = f(d_2)$, следовательно, и $f(gd) = f(d)$, тогда f есть неподвижная относительно $g \in G$.

Запишем сумму весов подчиненных разбиению Δg , в следующем виде $\sum_{f \in \Delta g} W(f) = \sum_{f \in \Delta g} a_{\Delta g, \omega} \cdot \omega$, где $a_{\Delta g, \omega}$ — количество функций, подчиненных разбиению Δg , с весом $\omega \in \Omega$.

Составим множество всех функций, неподвижных относительно $g \in G$, с весом $\omega \in \Omega: \Psi_\omega(g) = \{f \mid \forall d \in D f(gd) = f(d) \wedge W(f) = \omega\}$.

Утверждение 7.9.2 позволяет записать $a_{\Delta g, \omega} = |\Psi_\omega(g)|$. Величина C_ω — число классов эквивалентности с весом $\omega \in \Omega$, на которые распадается множество функций $S = S_1 \cup S_2 \cup \dots \cup S_{N_G}$ под действием группы G .

По лемме Бернсайда, $C_\omega = \frac{1}{|G|} \sum_{g \in G} |\Psi_\omega(g)| = \frac{1}{|G|} \sum_{g \in G} a_{\Delta g, \omega}$. Умножив последнее равенство на ω и просуммировав по всем весам из Ω , получим, что $\sum_{\omega \in \Omega} C_\omega \cdot \omega = \frac{1}{|G|} \sum_{g \in G} \sum_{\omega \in \Omega} a_{\Delta g, \omega} \cdot \omega$.

Сумма $\sum_{\omega \in \Omega} a_{\Delta g, \omega} \cdot \omega = \sum_{f \in \Delta g} W(f)$ — сумма весов функций, подчиненных разбиению Δg .

Так как $\sum_{f \in \Delta g} W(f) = [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \dots [R(\omega^m)]^{k_m}$, то

$$\sum_{\omega \in \Omega} C_\omega \cdot \omega = \frac{1}{|G|} \sum_{g \in G} [R(\omega^1)]^{k_1} [R(\omega^2)]^{k_2} \dots [R(\omega^m)]^{k_m}.$$

7.10. 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 пересчета Пойа.

7.10.1. 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 $gs, g^2s, g^3s, \dots, g^d s = s$ различны. 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

$$Z(G, x_1, x_2, \dots, x_n) = \frac{1}{|G|} \sum_{d|n} \gamma(d) \cdot x_d^{n/d}, \quad (7.10.1)$$

где $\gamma(d)$ — количество элементов $g \in G$ с порядком $|g| = d$. Суммирование выполняется по всем делителям d числа n .

7.10.2. Цикловой индекс циклической группы

Пусть $S = \{0, 1, \dots, n-1\}$ — множество вершин правильного n -угольника (рис. 7.6) на плоскости. Группа G — циклическая группа самосовмещений.

Образующий элемент группы $a \in G$ — вращение (отображение) S вокруг центра на угол $2\pi/n$. Следовательно, $G = \{a^n = e, a, a^2, \dots, a^{n-1}\}$, где e — тождественная подстановка. Если обозначить вершины многоугольника $\{0, 1, \dots, n-1\}$ элементами группы $\{a^n = e, a, a^2, \dots, a^{n-1}\}$, то вращение будет эквивалентно умножению вершины на соответствующий элемент группы. Например, совмещение при вращении на угол $2\pi/n$ равносильно умножению новых меток $\{a^n = e, a, a^2, \dots, a^{n-1}\}$ вершин n -угольника на элемент a , действительно, $a \cdot e \rightarrow a, a \cdot a \rightarrow a^2, \dots, a \cdot a^{n-1} \rightarrow a^n$. Таким образом, можно считать, что циклическая группа G действует на себе. Согласно (7.10.1) цикловой индекс данной группы примет вид

$$Z(G, x_1, x_2, \dots, x_n) = \frac{1}{|G|} \sum_{d|n} \gamma(d) x_d^{n/d} = \frac{1}{|G|} \sum_{d|n} \varphi(d) x_d^{n/d}, \quad (7.10.2)$$

где $\gamma(d) = \varphi(d)$ — функция Эйлера. Действительно, если порядок элемента равен d (элемент суммируется в $\gamma(d)$), то он является образующим элементом подгруппы H с G порядка $|H| = d$. Показать, что если порядки элементов циклической группы совпадают

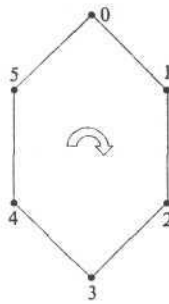


Рис. 7.6

$|a^k| = |a^m| = d$, то они являются образующими одной и той же подгруппы $H \subset G$. Число образующих в такой подгруппе $\varphi(d)$.

Найдем количество возможных раскрасок N_G двумя цветами вершин данного (рис. 7.6) n -угольника. Пусть цвета — $R = \{\bullet, \circ\}$. Воспользуемся теоремой 7.9.1 Поля. Для определения количества раскрасок положим веса цветов равными $\omega(\bullet) = 1$ и $\omega(\circ) = 1$. Тогда $R(\omega^k) = \omega^k(\bullet) + \omega^k(\circ) = 2$.

$$N_G = \sum_{\omega \in \Omega} C_\omega = Z(G, R(\omega^1), R(\omega^2), \dots, R(\omega^n)) = \frac{1}{n} \sum_{d|n} \varphi(d) [R(\omega^d)]^{n/d} = \\ = \frac{1}{n} \sum_{d|n} \varphi(d) 2^{n/d}.$$

Для треугольника ($n = 3$) число раскрасок равно

$$N_G = \frac{1}{3} \sum_{d|3} \varphi(d) 2^{n/d} = \frac{1}{3} (\varphi(1)2^3 + \varphi(3)2^1) = \frac{1}{3} (1 \cdot 2^3 + 2 \cdot 2^1) = 4.$$

Для шестиугольника число раскрасок равно

$$N_G = \frac{1}{6} \sum_{d|6} \varphi(d) 2^{n/d} = \frac{1}{6} (\varphi(1)2^6 + \varphi(2)2^3 + \varphi(3)2^2 + \varphi(6)2^1) = \\ = \frac{1}{6} (1 \cdot 2^6 + 1 \cdot 2^3 + 2 \cdot 2^2 + 2 \cdot 2^1) = 14.$$

7.10.3. Цикловой индекс симметрической группы

Пусть $S = \{1, 2, \dots, n\}$ — произвольное множество, на котором действует группа всех подстановок S_n — симметрическая группа, $|S_n| = n!$. Рассмотрим произвольную подстановку n в S_n с характеристикой (k_1, k_2, \dots, k_n) , где $1k_1 + 2k_2 + \dots + nk_n = n$. Подсчитаем количество подстановок с данной характеристикой. Для этого запишем в цикловом представлении, помещая знаки « \rightarrow » 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, \dots, 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, \dots, k_n) . Поэтому различных подстановок с указанной характеристикой будет $\frac{n!}{k_1! \cdot 1^{k_1} k_2! \cdot 2^{k_2} \dots k_n! \cdot n^{k_n}}$. Тогда цикло-

вой индекс симметрической группы можно записать в следующем виде: $Z(G, x_1, x_2, \dots, x_n) =$

$$= \frac{1}{n!} \left(\sum_{1k_1+2k_2+\dots+nk_n=n} \frac{n!}{k_1! \cdot 1^{k_1} k_2! \cdot 2^{k_2} \dots k_n! \cdot n^{k_n}} \cdot x_1^{k_1} x_2^{k_2} \dots x_n^{k_n} \right) =$$

$$= \sum_{1k_1+2k_2+\dots+nk_n=n} \frac{1}{k_1! \cdot k_2! \dots k_n!} \left(\frac{x_1}{1} \right)^{k_1} \left(\frac{x_2}{2} \right)^{k_2} \dots \left(\frac{x_n}{n} \right)^{k_n}.$$

Рассмотрим пример определения числа N_G раскрасок двумя цветами $R = \{\bullet, \circ\}$ трех усов $S = \{1, 2, 3\}$ антенны (рис. 7.7). 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: $(3,0,0)$, $(1,1,0)$ и $(0,0,1)$. Цикловой индекс примет вид

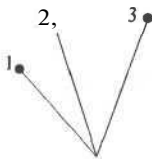


Рис. 7.7

$$Z(G, x_1, x_2, x_3) =$$

$$= \sum_{1k_1+2k_2+3k_3=3} \frac{1}{k_1! \cdot k_2! \cdot k_3!} \left(\frac{x_1}{1} \right)^{k_1} \left(\frac{x_2}{2} \right)^{k_2} \left(\frac{x_3}{3} \right)^{k_3} =$$

$$= \frac{1}{3!} \left(\frac{x_1}{1} \right)^3 + \frac{1}{1!1!} \left(\frac{x_1}{1} \right)^1 \left(\frac{x_2}{2} \right)^1 + \frac{1}{1!} \left(\frac{x_3}{3} \right)^1.$$

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,

$$N_G = \sum_{\omega \in \Omega} C_\omega = Z(G, R(\omega^1), R(\omega^2), \dots, R(\omega^n)) =$$

$$= \frac{1}{3!} \left(\frac{2}{1} \right)^3 + \frac{1}{1!1!} \left(\frac{2}{1} \right)^1 \left(\frac{2}{2} \right)^1 + \frac{1}{1!} \left(\frac{2}{3} \right)^1 = 4.$$

Глава 8

Элементы теории чисел

Теория чисел занимается изучением свойств целых чисел. Целыми называются числа $Z = \{\dots, -3, -2, -1, 0, 1, 2, 3, \dots\}$.

Для любых $a, b \in Z$ сумма $a + b$, разность $a - b$ и произведение $a \cdot b$ являются целыми числами. Но частное $\frac{a}{b}$ от деления a на b (если b не равно нулю) может быть как целым, так и не целым. В случае, когда частное $\frac{a}{b}$ является целым, то обозначают $a = bq$, где q — целое число, а b тогда называют делителем числа a и записывают так: $b \mid a$. В общем случае единственным является представление $a = bq + r$, $0 \leq r < b$ где r — называют остатком от деления.

8.1. 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, 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:

$$\begin{aligned} a &= bq_1 + r_2, & 0 < r_2 < b, \\ b &= r_2q_2 + r_3, & 0 < r_3 < r_2, \end{aligned}$$

$$\begin{aligned}
 r_2 &= \text{ОДЗ} + r_4, & 0 < r_4 < r_3, & & (8.1.1) \\
 \dots & & \dots & & \\
 r_{n-2} &= r_{n-1}q_{n-1} + r_n, & 0 < r_n < r_{n-1}, & & \\
 r_{n-1} &= r_n q_n, & & &
 \end{aligned}$$

заканчивающийся, когда получается некоторое $r_{n+1} = 0$. Последнее неизбежно случится, так как ряд B, r_2, r_3, \dots убывающих целых чисел не может содержать более чем b положительных.

4. Из формул (8.1.1) алгоритма Евклида следует, что $(a, b) = (b, L) = (r_2, r_3) = \dots = (r_{n-1}, r_n) = r_n$. Наибольший общий делитель равен последнему не равному нулю остатку алгоритма Евклида.
5. Из формул (8.1.1) 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, $\text{НОД} = (525, 231) = 21$.

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 .

8.2. 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. $\frac{ab}{(a,b)}$.

8.3. 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 $b \geq q$ (q — наименьший делитель), откуда $a > q$ или $q \leq \sqrt{a}$.
4. *Простых чисел бесконечно много*. Это следует из того, что каковы бы ни были различные простые числа p_1, p_2, \dots, p_k , можно получить новое простое, среди них не находящееся. Таковым будет простой делитель суммы $p_1 p_2 \dots p_k + 1$, который, деля всю сумму, не может совпадать ни с одним из простых p_1, p_2, \dots, p_k .
5. *Решето Эратосфена* для составления таблицы простых чисел. Данный способ состоит в следующем.

Выписываем числа

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, \dots, N.$$

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 не вычеркнутое 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, меньших 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 простым, не превосходящих \sqrt{N} .

В алгоритме 8.1 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 8.1. Решето Эратосфена

```

X = (1, 1, ..., 1);
for k = 3 to  $\sqrt{2N+1}$  by 2 do
  {X(k-1)/2 — индикатор нечетного числа k}
  {Вычеркнуть числа кратные k}
  if X(k-1)/2 = 1 then for i = k2 to 2N+1 by 2k do X(i-1)/2 = 0;
  { Печать простых чисел }
for k = 1 to N do if Xk = 1 then вывести 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 Си.

Алгоритм 8.2. Программа на Pascal'e генерации простых чисел

```

Program PgmPrimary; {Программа генерации простых чисел}
uses CRT;
Const
  N=45; {Простые числа среди 3, 5, ..., 2*N+1}
Type
  Vector=array[1..N] of Boolean;
Var
  f :Text;      {Текстовый файл простых чисел}
  X :Vector;    {Вектор индикаторов простых чисел}

Procedure Primary; {Генерация простых чисел}
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; {Исключить числа кратные k}
      while i<=nm do begin
        X[(i-1) div 2]:=FALSE;
        i:=i+2*k;
      end;
    end;
    k:=k+2; {Выбираем нечетные числа k}
  end;
end;

```



```

{Печать простых чисел по индикаторам X[k]}
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'); {Файл для записи простых чисел}
  Rewrite(f);
  Primary;
  Close(f);
end.

```

Алгоритм 8.3. Программа на Си генерации простых чисел

```

#include <stdio.h>
#include <math.h>
#define N 45
char X[N+1]; // Индикаторы простых чисел
// в диапазоне 3, 5, . . . , 2*N+1}

int main (void) {
  FILE *f;
  int i, k, M, run;
  f = fopen ("primary. out", "wt"); // Файл для записи
  // простых чисел

  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;
  //Печать простых нечетных чисел
  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 8.2 и 8.3 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 8.2 и 8.3 сохраняются в выходном файле `Primagy.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 $[3, 27Y + 1]$, границы которого распечатываются во второй строке результирующего файла. Во второй же строке указывается и общее количество найденных простых чисел.

8.4. 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 $a = b \pmod{m}$.

Например, $77 \equiv 5 \pmod{8}$, $102 \equiv 0 \pmod{3}$.

Свойства сравнений

1. $a = a \pmod{m}$ — свойство рефлексивности.
2. $a \equiv b \pmod{m} \Rightarrow b \equiv a \pmod{m}$ — свойство симметричности.
3. $a \equiv b \pmod{m}$, $b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$ — свойство транзитивности.
4. $a \equiv b \pmod{m} \Rightarrow (a, m) = (b, m)$.
5. $a \equiv b \pmod{m}$, $c \equiv d \pmod{m} \Rightarrow a + csb + d \pmod{m}$.
6. $a \equiv b \pmod{m}$, $c \equiv d \pmod{m} \Rightarrow ac \equiv bd \pmod{m}$.
7. $a = a_1d$, $b = b_1d$, $(d, m) = 1$, $a \equiv b \pmod{m} \Rightarrow a_1 \equiv b_1 \pmod{m}$.
8. $a \equiv b \pmod{m}$, $m_1 \setminus m$ (m_1 делит m) $\Rightarrow a \equiv b \pmod{m_1}$.
9. $a \equiv b \pmod{m}$, $d \setminus a$, $d \setminus b$, $d \setminus m \Rightarrow a_1 \equiv b_1 \pmod{m_1}$.

8.5. Полная система вычетов

Свойства 1–3 сравнений показывают, что операция сравнения целых чисел по модулю является отношением эквивалентности. Множество всех целых чисел Z разбивается на классы эквивалентности (рис. 8.1), 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 $r = s \pmod{m}$ имеют одинаковые остатки от деления на m . Числа одного и того же класса имеют с модулем m один и тот же наибольший общий делитель, т.к. из $r, s \in \{k_i\}$ следует, что $(r, m) = (s, m)$ (см. п.8.4). Особенно важны классы, для которых этот делитель равен единице, т.е. классы, содержащие числа, взаимно простые с модулем.

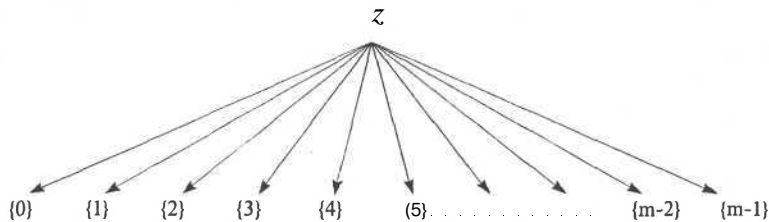


Рис. 8.1. Полная система вычетов

- *Определение.* Множество классов вычетов $\{0\}, \{1\}, \dots, \{m-1\}$ по модулю m называется полной системой вычетов (п.с.в.). 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).

8.6. Приведенная система вычетов

- *Определение.* Пусть m — целое положительное число. Множество классов из полной системы вычетов $\{0\}, \{1\}, \dots, \{m-1\}$ взаимно простых с m называется *приведенной системой вычетов*, которую будем обозначать как M_m или $M_m(m)$. Приведенную систему вычетов, следовательно, можно составить из чисел п.с.в., взаимно простых с модулем. Обыкновенно M_m выделяют из системы наименьших неотрицательных вычетов: $0, 1, 2, \dots, m-1$.

- **Утверждение 8.6.1.** M_π является группой с операцией умножения.

Доказательство. Проверим все свойства (аксиомы) группы.

1. *Замкнутость.* Пусть произвольные $a, b \in M_\pi$. Покажем, что $a \cdot b \in M_\pi$. По условию $a = 1 \pmod{m}$ и $b = 1 \pmod{m}$, тогда $ab = 1 \pmod{m}$.
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 $aM_\pi \subset M_\pi$, так как для M_π выполняется свойство замкнутости. Для доказательства $aM_\pi = M_\pi$ теперь достаточно показать, что все элементы множества aM_π различны. Предположим, что существуют $b_1 \neq b_2 \in M_\pi$ для которых $ab_1 - ab_2 = 0 \pmod{m}$. Отсюда $a(b_1 - b_2) = 0 \pmod{m}$. А так как $(a, m) = 1$ — взаимно простые, то $b_1 - b_2 \equiv 0 \pmod{m}$ или $b_1 = b_2 \pmod{m}$, что противоречит принадлежности их разным классам.

Таким образом, все элементы множества aM_π различны, значит, $\exists b \in M_\pi$, что $a \cdot b = 1 \pmod{m}$. Элемент b является обратным к a , и по доказательству он единственный такой элемент (*существование обратного элемента a доказано*).

Таким образом, получили, что M_π является группой по умножению. Порядок этой группы равен количеству чисел меньших m и взаимно простых с ним.

8.7. Функция Эйлера

- *Определение.* Функция Эйлера $\varphi(m)$ определяется для всех целых положительных m и равна количеству чисел ряда

$$1, 2, \dots, m - 1,$$

взаимно простых с m , где число 1 полагается взаимно простым с любым из чисел и $\varphi(1) = 1$.

Примеры. $\varphi(1) = 1$, $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(4) = 2$, $\varphi(5) = 4$, $\varphi(6) = 2$.

- *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. Пусть $C(m_1 m_2)$ — циклическая группа порядка $|C(m_1 m_2)| = m_1 m_2$, число образующих ее равно $\varphi(m_1 m_2)$ (см. утверждение 7.3.4). Так как $(m_1, m_2) = 1$, то допустимо разложение (см. п. 7.4) группы $C(m_1 m_2)$ в прямое произведение своих циклических подгрупп $C(m_1 m_2) = C(m_1) \times C(m_2)$ и, следовательно, число образующих группы $C(m_1 m_2)$ равно $\varphi(m_1)\varphi(m_2)$.

Доказательство 2. Достаточно показать, что

$$|M_\pi(m_1 m_2)| = |M_\pi(m_1)| \cdot |M_\pi(m_2)|.$$

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. 8.1). Приведем значения целых a и b к значениям $a \in \{1, 2, \dots, m_1\}$ и $b \in \{1, 2, \dots, m_2\}$, для которых $am_2 + bm_1 = 1 \pmod{m_1 m_2}$. Пусть $c \in \{1, 2, \dots, m_1 m_2\}$, тогда верно $cam_2 + cbm_1 = c \pmod{m_1 m_2}$, где значения ca и cb приведены к значениям $a \in \{1, 2, \dots, m_1\}$ и $b \in \{1, 2, \dots, m_2\}$. Таким образом, произвольное число $c \in \{1, 2, \dots, m_1 m_2\}$ можно записать в виде $am_2 + bm_1 = c \pmod{m_1 m_2}$, где $a \in \{1, 2, \dots, m_1\}$ и $b \in \{1, 2, \dots, m_2\}$. Данное представление является однозначным, так как число возможных пар (a, b) равно $m_1 m_2$ и такое же количество представляемых чисел $c \in \{1, 2, \dots, m_1 m_2\}$. Далее рассмотрим все представления $am_2 + bm_1$ для всех $a \in \{1, 2, \dots, m_1\}$ и $b \in \{1, 2, \dots, m_2\}$.

2. Пусть $a \in M_\pi(m_1)$ и $b \in M_\pi(m_2)$, т.е. $(a, m_1) = 1$ и $(b, m_2) = 1$ и $(m_1, m_2) = 1$. Покажем, что $(am_2 + bm_1, m_1 m_2) = 1$. Выражение $(am_2 + bm_1, m_1 m_2) = 1$ 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 $m_1 m_2$, равно $\varphi(m_1) \cdot \varphi(m_2)$. Покажем, что других чисел взаимно простых с $m_1 m_2$ 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 $(am_2 + bm_1, m_1 m_2) \neq 1$, т.е. такие числа не являются взаимно простыми с $m_1 m_2$.

4. Из пунктов 1), 2), 3) следует, что $\varphi(m_1 m_2) = \varphi(m_1)\varphi(m_2)$.

$$\text{Свойство 2. } \varphi(p^\alpha) = p^\alpha - p^{\alpha-1} = p^\alpha \left(1 - \frac{1}{p}\right),$$

где p — простое число и $\alpha > 0$.

Доказательство. Числа вида $k \cdot p$, где $k = \{1, 2, \dots, p^{\alpha-1}\}$ — это все числа не взаимно простые с p^α среди $1, 2, \dots, p^\alpha$, а значит остальные являются взаимно простыми с p^α . Отсюда $\varphi(p^\alpha) = p^\alpha - p^{\alpha-1}$.

Свойство 3. Пусть разложение числа m на простые множители имеет вид $m = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, где p_i — простые числа, тогда

$$\varphi(m) = \prod_{i=1}^k \varphi(p_i^{\alpha_i}) = \prod_{i=1}^k p_i^{\alpha_i} \left(1 - \frac{1}{p_i}\right) = m \cdot \prod_{i=1}^k \left(1 - \frac{1}{p_i}\right).$$

Свойство 4. $\sum_{d|n} \varphi(d) = n$, где d — различные делители числа n .

Доказательство 1. Пусть $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ — разложение числа n на простые множители. Правило обобщенного произведения (см. п. 3.4) позволяет записать:

$$\begin{aligned} \sum_{d|n} \varphi(d) &= \sum_{(r_1, r_2, \dots, r_k)} \varphi(p_1^{r_1} p_2^{r_2} \dots p_k^{r_k}) = \sum_{(r_1, r_2, \dots, r_k)} \varphi(p_1^{r_1}) \varphi(p_2^{r_2}) \dots \varphi(p_k^{r_k}) = \\ &= \prod_{i=1}^k \sum_{r_i=0}^{\alpha_i} \varphi(p_i^{r_i}) = \prod_{i=1}^k \sum_{d|p_i^{\alpha_i}} \varphi(d). \end{aligned}$$

Сумма

$$\begin{aligned} \sum_{d|p^\alpha} \varphi(d) &= \varphi(p^0) + \varphi(p^1) + \dots + \varphi(p^\alpha) = \\ &= 1 + (p-1) + (p^2-p) + \dots + (p^\alpha - p^{\alpha-1}) = p^\alpha. \end{aligned}$$

Отсюда искомая сумма

$$\sum_{d|n} \varphi(d) = \prod_{i=1}^k \sum_{d|p_i^{\alpha_i}} \varphi(d) = \prod_{i=1}^k p_i^{\alpha_i} = n.$$

Доказательство 2. Пусть $C(n)$ — циклическая группа порядка $|C(n)| = n$. Для всякого делителя d числа n существует единственная подгруппа $C(d)$ с $C(n)$ порядка $|C(d)| = d$. Образующие группы $C(d)$ составляют множество $S_d = \{x \in C(n) \mid |x| = d\}$. Из утверждения 7.3.4 следует, что число образующих группы $C(d)$ равно $\varphi(d) = |S_d|$. Так как $S_d \cap S_{d'} = \emptyset$ для $d \neq d'$, то $\bigcup_{d|n} S_d = C(n)$ или

$$\sum_{d|n} \varphi(d) = n.$$

- *Теорема 8.7.1 Эйлера.* $x^{\varphi(m)} \equiv 1 \pmod{m}$, где $m > 0$ и $(x, m) = 1$.

Доказательство. Приведенная система вычетов M_π по модулю m является группой, порядок которой $|M_\pi| = \varphi(m)$. Пусть x — произвольное такое, что $(x, m) = 1$ и $x \equiv r \pmod{m}$, где $r \in M_\pi$. Из свойств сравнений следует, что наибольший общий делитель $(r, m) = 1$. Теорема 7.3.1 Лагранжа утверждает, что порядок элемента группы кратен порядку этой группы. Пусть k — порядок элемента r , т.е. $r^k \equiv 1 \pmod{m}$. Отсюда $\varphi(m) = k \cdot d$, где d — положительное целое. Тогда $r^{\varphi(m)} = r^{kd} \equiv 1 \pmod{m}$. Из $x \equiv r \pmod{m}$ следует, что $x^{\varphi(m)} \equiv r^{\varphi(m)} \pmod{m}$, а значит, и $x^{\varphi(m)} \equiv 1 \pmod{m}$.

- *Теорема §. 7.2 Ферма.* $x^p \equiv x \pmod{p}$, где p — простое число; x — произвольное целое положительное число.

Доказательство. Пусть $x \equiv 0 \pmod{p}$, тогда и $x^p \equiv 0 \pmod{p}$. Пусть теперь $x \equiv r \pmod{p}$, где $r \in \{1, 2, \dots, p-1\}$ и, значит, $(r, p) = 1$. Из теоремы Эйлера следует, что $x^{\varphi(p)} \equiv 1 \pmod{p}$, где $\varphi(p) = p-1$. Отсюда $x^{p-1} \equiv 1 \pmod{p}$ и $x^p \equiv x \pmod{p}$.

- *Теорема 8.7.3 Вильсона.* $(p-1)! + 1 \equiv 0 \pmod{p}$, где p — простое число.

Доказательство. Пусть $M_\pi = \{1, 2, \dots, p-1\}$ — приведенная система вычетов по модулю p . M_π является группой. Для любого $x \in \{1, 2, \dots, p-1\}$ существует единственный обратный элемент $y \in \{1, 2, \dots, p-1\}$ такой, что $x \cdot y \equiv 1 \pmod{p}$.

Заметим, что $x \cdot x = 1 \pmod{p}$ выполняется только для двух элементов: $x = 1$ и $x = p-1$. Действительно $x^2 - 1 = (x-1)(x+1) \equiv 0 \pmod{p}$ равносильно $x-1 \equiv 0 \pmod{p}$ или $x+1 \equiv 0 \pmod{p}$. 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 $x \in \{2, 3, \dots, p-2\}$ существует единственный обратный $y \in \{2, 3, \dots, p-2\}$ такой, что $xy \equiv 1 \pmod{p}$ и $x \neq y$. Тогда верно $2 \cdot 3 \cdot \dots \cdot (p-1) \equiv 1 \pmod{p}$. Умножив последнее сравнение на $1 \cdot (p-1)$, получим $1 \cdot 2 \cdot \dots \cdot (p-1) \equiv (p-1)! \equiv -1 \pmod{p}$ или $(p-1)! + 1 \equiv 0 \pmod{p}$.

Задача. Пусть p — простое и h_1, h_2, \dots, h_n — целые числа. Доказать, что $(h_1 + h_2 + \dots + h_n)^p \equiv h_1^p + h_2^p + \dots + h_n^p \pmod{p}$.

Решение. Согласно теореме Ферма (8.7.2), $h_i^p \equiv h_i \pmod{p}$, $i = \overline{1, n}$. Свойства операции сравнения (см. п.8.4) позволяют записать данные n сравнений $h_i^p \equiv h_i \pmod{p}$ в виде их суммы: $h_1^p + h_2^p + \dots + h_n^p \equiv h_1 + h_2 + \dots + h_n \pmod{p}$. С другой стороны, верно и $(h_1 + h_2 + \dots + h_n)^p \equiv h_1 + h_2 + \dots + h_n \pmod{p}$, что непосредственно вытекает из теоремы Ферма.

8.8. Функция Мёбиуса. Формула обращения Мёбиуса

- *Определение.* Функция Мёбиуса $\mu(n)$ определяется для всех целых положительных n и равна

$$\mu(n) = \begin{cases} 1, & \text{если } n = 1, \\ 0, & \text{если } n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k} \text{ и } \exists \alpha_i > 1, \\ (-1)^k, & \text{если } n = p_1 p_2 \dots p_k, \end{cases}$$

где $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ — разложение на простые множители, p_i — простые числа, α_i — кратность p_i в разложении.

Пример. $\mu(1) = 1$, $\mu(2) = -1$, $\mu(3) = -1$, $\mu(4) = 0$, $\mu(5) = -1$, $\mu(6) = 1$, $\mu(7) = -1$, $\mu(8) = 0$, $\mu(9) = 0$, $\mu(10) = 1$, $\mu(11) = -1$, $\mu(12) = 0$, $\mu(13) = -1$, $\mu(14) = 1$, $\mu(15) = 1$, $\mu(16) = 0$, $\mu(17) = -1$, $\mu(18) = 0$, $\mu(19) = -1$, $\mu(20) = 0$, $\mu(21) = 1$, $\mu(22) = 1$, $\mu(23) = -1$.

- *Лемма 8.8.1.* $\sum_{d|n} \mu(d) = \begin{cases} 0, & \text{если } n > 1, \\ 1, & \text{если } n = 1, \end{cases}$

где суммирование идет по всем делителям d числа n .

Доказательство. Если $n = 1$, то $\sum_{d|1} \mu(d) = \mu(1) = 1$. Пусть теперь

$n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k} \neq 1$ — разложение на простые множители. Тогда

$$\sum_{d|1} \mu(d) = \sum_{d|n, \mu(d) \neq 0} \mu(d) = \sum_{r=0}^k C_k^r (-1)^r = (1-1)^k = 0.$$

Все делители d , для которых $\mu(d) \neq 0$, имеют вид $p_{i_1} p_{i_2} \dots p_{i_r}$, и $\mu(p_{i_1} p_{i_2} \dots p_{i_r}) = (-1)^r$.

Количество таких делителей $p_{i_1} p_{i_2} \dots p_{i_r}$, выбираемых из p_1, p_2, \dots, p_k , равно числу сочетаний C_k^r .

• **Теорема 8.8.1.** Формула обращения Мёбиуса:

если $f(n) = \sum_{d|n} g(d)$ то $g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$,

где $f(n), g(n)$ — функции, определенные для всех целых положительных n .

Доказательство. Выполним подстановку $f(\frac{n}{d})$ в сумме $\sum_{d|n} \mu(d) f(\frac{n}{d}) = \sum_{d|n} \mu(d) \left(\sum_{\delta|\frac{n}{d}} g(\delta) \right)$. Заметим, что здесь число n неявно

рассматривается в виде произведения $n = d \cdot 5 \cdot r$, где делители d и 5 принимают все допустимые значения независимо друг от друга и порядок суммирования не влияет на значение суммы, т.е.

$$\sum_{d|n} \mu(d) \left(\sum_{\delta|\frac{n}{d}} g(\delta) \right) = \sum_{\delta|n} g(\delta) \left(\sum_{d|\frac{n}{\delta}} \mu(d) \right), \text{ где } \sum_{d|\frac{n}{\delta}} \mu(d) = \begin{cases} 0, & \text{если } \frac{n}{\delta} \neq 1, \\ 1, & \text{если } \frac{n}{\delta} = 1 \end{cases}$$

это вследствие леммы 8.8.1. Тогда

$$\sum_{\delta|n} g(\delta) \left(\sum_{d|\frac{n}{\delta}} \mu(d) \right) = g(n) \left(\sum_{d|\frac{n}{n}} \mu(d) \right) = g(n) \left(\sum_{d|1} \mu(d) \right) = g(n) \mu(1) = g(n).$$

Задача. Установить связь функций Эйлера $\phi(n)$ и Мёбиуса $\mu(n)$.

Решение. $\sum_{d|n} \mu(d) = 0$ — свойство 4 функции Эйлера (см. п.8.7).

К данной сумме применим формулу обращения Мёбиуса:

$$\phi(n) = \sum_{d|n} \mu(d) \frac{n}{d} = n \sum_{d|n} \frac{\mu(d)}{d} \text{ и, наконец, } \frac{\phi(n)}{n} = \sum_{d|n} \frac{\mu(d)}{d}.$$

Например, для $n = 6$ имеем $\phi(6) = 2$, все делители $d \in \{1, 2, 3, 6\}$, $\mu(1) = 1, \mu(2) = -1, \mu(3) = -1, \mu(6) = 1$ и, наконец, выражение $\frac{\phi(n)}{n} = \sum_{d|n} \frac{\mu(d)}{d}$ в этом случае примет вид $\frac{2}{6} = \frac{1}{1} + \frac{-1}{2} + \frac{-1}{3} + \frac{1}{6}$.

Пусть $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ — разложение на простые множители.

Так как $\phi(n) = n \cdot \prod_{i=1}^k \left(1 - \frac{1}{p_i} \right)$ — свойство 3 функции Эйлера (см.

п.8.7), то $n \cdot \prod_{i=1}^k \left(1 - \frac{1}{p_i} \right) = n \cdot \sum_{d|n} \dots$ или $\prod_{i=1}^k \left(1 - \frac{1}{p_i} \right) = \sum_{d|n} \frac{\mu(d)}{d}$.

Задачи и упражнения



Комбинаторные схемы

1. 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) $C_n^k = C_{n-1}^k + C_{n-2}^{k-1} + \dots + C_{n-k-1}^0$;

в) $C_{2n}^{n-1} = C_{2n-1}^n + C_{2n-2}^n + \dots + C_{2n-n}^n$;

г) $C_{n+m+1}^m = C_{n+m}^m + C_{n+m-1}^{m-1} + \dots + C_n^0$.

2. Доказать тождества:

а) $\sum_{k=0}^n C_n^k = 2^n$;

б) $\sum_{k=0}^n k \cdot C_n^k = n2^{n-1}$;

в) $\sum_{k=0}^n k^2 \cdot C_n^k = n(n+1)2^{n-2}$;

г) $\sum_{k=0}^n C_n^k (m-1)^{n-k} = m^n$;

д) $\sum_{k=0}^n (-1)^k C_n^k = 0$.

3. Доказать тождество $\sum_{i_n=1}^m \sum_{i_{n-1}=1}^{i_n} \sum_{i_{n-2}=1}^{i_{n-1}} \dots \sum_{i_2=1}^{i_3} \sum_{i_1=1}^{i_2} \sum_{i_0=1}^{i_1} 1 = C_{n+m}^{n+1}$.

4. Доказать, что $\sum_{k=1}^n \frac{C_{n-1}^{k-1}}{C_{2n-1}^k} = \frac{1}{n+1}$, $n > 1$.

5. Доказать, что следующие числа: $\frac{(2n)!}{2^n}$, $\frac{(3n)!}{2^n 3^n}$, $\frac{(n^2)!}{n^n}$, $\frac{(2n)!}{n!(n+1)!}$

являются целыми.

6. Доказать формулу бинома Ньютона $(a+b)^n = \sum_{k=0}^n C_n^k a^{n-k} b^k$.

7. Найти число подмножеств множества $M = \{a_1, a_2, \dots, a_n\}$.

8. Доказать, что $\sum_k C_n^{2k} = \sum_k C_n^{2k+1} = 2^{n-1}$.

9. Доказать, что $\sum_{r=0}^A C_m^r C_n^k = C_{n+m}^k$ и $\sum_{k=0}^n (C_n^k)^2 = C_{2n}^n$ (теорема сложения).

10. Доказать равенство $\sum_{k=0}^n \sum_{r=0}^{n-k} C_n^k C_{n-k}^r = 3^n$.

11. Доказать равенство $\sum_{k=0}^n \frac{1}{k+1} C_n^k = \frac{2^{n+1}-1}{n+1}$.

12. Доказать равенство $\sum_{k=0}^n (-1)^k \frac{1}{k+1} C_n^k = \frac{1}{n+1}$.

13. Найти сумму $\sum_{k=0}^n \left(k^2 - 1 + \frac{1}{k+1} \right) C_n^k$.

14. Доказать тождество $\sum_{n_1! n_2! \dots n_k!} = k^n$, где суммирование распространяется на все упорядоченные разбиения n на k слагаемых:

$n = n_1 + n_2 + \dots + n_k$, $n > 0$ — целые числа.

15. 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 ?

16. Сколькими способами число 11^n можно представить в виде трех сомножителей (представления, отличающиеся порядком сомножителей, считаются различными; 1^0 — сомножитель)?

17. Сколькими способами можно указать на шахматной доске $2l \times 2l$ два квадрата — белый и черный?

18. Сколькими способами можно указать на шахматной доске $2n \times 2l$ белый и черный квадраты, не лежащие на одной горизонтали и вертикали?

19. Какое количество матриц можно составить из l строк и m столбцов с элементами из множества $\{0, 1\}$?

20. Сколькими способами можно составить трехцветный флаг, если имеется материал 5 различных цветов? Та же задача, если одна из полос должна быть красной.

21. Надо послать 6 срочных писем. Сколькими способами это можно сделать, если любое письмо можно передать с любым из 3 курьеров?

22. У одного студента 7 книг, у другого 9 различных книг. Сколькими способами они могут обменять одну книгу одного на одну книгу другого?

23. Сколько различных словарей надо издать, чтобы можно было переводить с любого из данных n языков на любой другой язык этого же множества?
24. В правление избрано m человек. Из них надо выбрать председателя, заместителя председателя, секретаря и казначея. Сколькими способами можно это сделать?
25. 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?
26. У мамы m яблок и n груш. Каждый день в течение $n + m$ дней подряд она выдает сыну по одному фрукту. Сколькими способами это может быть сделано?
27. Найти число векторов $a = (a_1, a_2, \dots, a_n)$, координаты которых удовлетворяют условию $a_i \in \{0, 1\}, i = 1, 2, \dots, n, \sum_{i=1}^n a_i = r$.
28. У англичан принято давать детям несколько имен. Сколькими способами можно назвать ребенка, если ему дают не более трех имен, а общее число имен равно t ?
29. Сколькими способами можно расставить белые фигуры: 2 коня, 2 слона, 2 ладьи, ферзя и короля на первой линии шахматной доски?
30. 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?
31. Сколькими способами можно посадить n мужчин и l 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.
32. На школьном вечере присутствуют 12 девушек и 15 юношей. Сколькими способами можно выбрать из них 4 пары?
33. Пусть n ($n > 2$) 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?
34. Хор состоит из 10 участников. Сколькими способами можно в течение трех дней выбирать по 6 участников, так, чтобы каждый день были различные составы хора?

35. Сколькими способами можно распределить $3l$ различных предметов между тремя людьми так, чтобы каждый получил n предметов?
36. Имеется n абонентов. Сколькими способами можно одновременно соединить три пары?
37. Сколькими способами можно составить три пары из n шахматистов?
38. Рассматриваются всевозможные разбиения $2l$ элементов на пары, причем разбиения, отличающиеся друг от друга порядком элементов внутри пар и порядком расположения пар, считаются совпадающими. Определить число таких разбиений.
39. Доказать, что нечетное число предметов можно выбрать из n предметов 2^{n-1} способами.
40. Сколькими способами можно посадить рядом 3 англичан, 3 французов и 3 немцев так, чтобы никакие три соотечественника не сидели рядом?
41. 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?
42. Сколькими способами можно выбрать 6 карт из колоды, содержащей 52 карты, так, чтобы среди них были карты каждой масти?
43. На железнодорожной станции имеется m светофоров. Сколько может быть дано различных сигналов, если каждый светофор имеет три состояния: красный, желтый и зеленый?
44. 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.
45. Найти число способов раскладки n различных шаров по m различным корзинам.
46. Найти число способов раскладки l одинаковых шаров по m различным корзинам.
47. Сколькими способами можно разместить l одинаковых шаров по m различным корзинам при следующих условиях:
- пустых корзин нет;
 - во второй корзине k шаров;
 - в первых k корзинах соответственно a_1, a_2, \dots, a_k шаров?

48. Сколькими способами можно разместить n_1 красных, n_2 желтых и n_3 зеленых шаров по m различным урнам?
49. Сколькими способами 3 человека могут разделить между собой 6 одинаковых яблок, 1 апельсин, 1 сливу, 1 лимон, 1 грушу, 1 айву и 1 финик?
50. Поезду, в котором находится l пассажиров, предстоит сделать m остановок. Сколькими способами могут распределиться пассажиры между этими остановками?
51. Сколькими способами можно раскрасить квадрат, разделенный на четыре части, пятью цветами:
- допуская окрашивание разных частей в один цвет;
 - если различные части окрашиваются разными цветами?
52. Сколькими способами можно выбрать 5 номеров из 36?
53. В скольких случаях при игре в «Спортлото» (угадывание 5 номеров из 36) будут правильно выбраны: а) ровно 3 номера; б) не менее 3 номеров?
54. Сколько существует различных комбинаций из 30 монет достоинством 1, 2 и 5 рублей (*построить дереворешений, см. п. 2.2.2*)?
55. 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?
56. Определить коэффициент c в одночлене $cx_1^3x_2^4x_3^3$ после разложения выражения $(x_1 + x_2 + x_3)^{10}$ и приведения подобных членов.
57. Сколько делителей имеет число $q = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_n^{\alpha_n}$, где p_i — простые числа, не равные единице, α_i — некоторые натуральные числа? Чему равна сумма этих делителей?
58. Доказать, что в разложении числа $n!$ на простые множители простое число p входит с показателем $r = \text{Иг} \left[\frac{n}{p} \right] + \left[\frac{n}{p^2} \right] + \left[\frac{n}{p^3} \right] + \dots$
59. Из выбранных k различных чисел от 1 до l составляют произведение, k фиксировано. Какое количество полученных таким образом произведений делится на простое число $p < n$?
60. Сколько можно составить перестановок из n элементов, в которых данные m элементов не стоят рядом в любом порядке?
61. На шахматную доску $n \times n$ произвольным образом поставили две ладьи — черную и белую. Что вероятнее: ладьи бьют друг друга или нет?

62. Доказать, что из пяти грибов, растущих в лесу и не расположенных на одной прямой, всегда можно найти четыре таких, которые служат вершинами выпуклого четырехугольника.
63. В розыгрыше первенства мира по футболу участвуют 20 команд. Какое наименьшее число игр должно быть сыграно, чтобы среди любых трех команд нашлись две, уже игравшие между собой?
64. 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.
65. В некотором 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 члена.
66. 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?
67. Группа из 41 студента успешно сдала сессию из трех экзаменов. Возможные оценки: 5,4,3. Доказать, что, по крайней мере, пять студентов сдали сессию с одинаковыми оценками.
68. 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, см.п.2.2.2*)?
69. Каких чисел больше среди первого миллиона: тех, в записи которых встречается 1, или тех, в записи которых ее нет?
70. Пусть числа $1, 2, \dots, n$ расположены подряд по кругу. Двигаясь по кругу, вычеркиваем каждое второе число. Показать, что последнее не вычеркнутое число равно $2n - 2^{\lfloor \log_2 n \rfloor + 1} + 1$.
71. 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?
72. Какова таблица инверсий для перестановки 271845936?
73. Какой перестановке соответствует таблица инверсий 50121200?
74. Пусть перестановке $a_1 a_2 \dots a_n$ соответствует таблица инверсий $d_1 d_2 \dots d_n$. Какой перестановке тогда будет соответствовать следующая таблица инверсий $(n - 1 - d_1)(n - 2 - d_2) \dots (0 - d_n)$?

75. 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} .

76. Сколькими способами можно распределить $3n$ различных книг между тремя лицами так, чтобы числа книг образовывали арифметическую прогрессию?

77. 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?

78. Сколькими способами можно разбить 30 рабочих на 3 бригады по 10 человек в каждой бригаде? На 10 групп по 3 человека в каждой группе?

79. Сколькими способами можно разделить колоду из 36 карт пополам так, чтобы в каждой пачке было по два туза?

80. Сколькими способами можно разложить 10 книг в 5 бандеролей по 2 книги в каждую (порядок бандеролей не принимается во внимание)?

81. Сколькими способами можно разложить 9 книг в 4 бандероли по 2 книги и в 1 бандероль 1 книгу (порядок бандеролей не принимается во внимание)?

82. Сколькими способами можно разделить 9 книг в 3 бандероли по 3 книги в каждую (порядок бандеролей не принимается во внимание)?

83. 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?

84. Сколькими способами можно расположить в 9 лузах 7 белых шаров и 2 черных шара? Часть луз может быть пустой, и лузы считаются различными.

85. В лифт сели 8 человек. Сколькими способами они могут выйти на четырех этажах так, чтобы на каждом этаже вышел, по крайней мере, один человек?

86. Доказать, что число упорядоченных разбиений числа n на k натуральных слагаемых, т. е. число решений уравнения $n = x_1 + x_2 + \dots + x_k$, $x_i > 0, i = 1, 2, \dots, k$, равно C_{n-1}^{k-1} , а общее число упорядоченных разбиений для различных k равно 2^{n-1} .
87. 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 $n = n_1 + n_2 + \dots + n_k$?
88. Сколько существует чисел от 0 до 10^n , которые не содержат две идущие друг за другом одинаковые цифры?
89. Сколько существует натуральных n -значных чисел, у которых цифры расположены в неубывающем порядке?
90. Сколько существует натуральных чисел, не превышающих 10^n , у которых цифры расположены в неубывающем порядке?
91. Сколькими способами можно расставить n нулей и k единиц так, чтобы никакие две единицы не стояли рядом?
92. Город имеет вид прямоугольника, разделенного улицами на квадраты. Число 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?
93. Как разбить квадратное поле на участки так, чтобы высеять на нем m сортов пшеницы для сравнения урожайности этих сортов, исключая влияние изменения плодородия в пределах участка? Считаем, что плодородие убывает при удалении от одной стороны поля (неизвестно, какой именно) к противоположной.
94. Бросают m игральных костей, помеченных числами 1, 2, 3, 4, 5, 6. Сколько может получиться различных результатов (результаты, отличающиеся порядком очков, считаются одинаковыми)?
95. Имеем m различных шаров и k различных корзин. Сколькими способами можно разместить предметы по корзинам, допускаются пустые корзины?
96. 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.*
97. Найти число способов разложения m шаров по k корзинам так, чтобы r корзин остались пустыми. *Указание: воспользоваться правилом включения и исключения.*

98. Имеем t различных шаров a_1, a_2, \dots, a_m и столько же различных корзин k_1, k_2, \dots, k_m . 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.*

99. Задача о беспорядках. Имеем t различных шаров a_1, a_2, \dots, a_m и столько же различных корзин k_1, k_2, \dots, k_m . 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.*

100. Найти число перестановок t шаров, в которых ровно r элементов остаются на месте. *Указание: воспользоваться правилом включения и исключения.*

101. Доказать, что r различных вещей можно разделить между $n + r$ людьми так, чтобы данные n людей получили, по крайней мере, по одному предмету, способами $C_n^0(n+r)^r - C_n^1(n+r-1)^r + C_n^2(n+r-2)^r - \dots + (-1)^n C_n^n(n+r-n)^r$. *Указание: воспользоваться правилом включения и исключения.*

102. Рыцарские переговоры. К обеду за круглым столом приглашены n пар враждующих рыцарей, $n > 2$. Требуется рассадить их так, чтобы никакие два врага не сидели рядом. Показать, что это можно сделать $\sum_{k=0}^n (-1)^k C_n^k 2^k (2n-k)!$ способами. *Указание: воспользоваться правилом включения и исключения.*

таться правилом включения и исключения.

103. Найти число целых положительных чисел, не превосходящих 1000 и не делящихся ни на одно из чисел 3, 5 и 7.

104. Найти число целых положительных чисел, не превосходящих 1000 и не делящихся ни на одно из чисел 6, 10 и 15.

105. Показать, что если $n = 30m$, то число целых, не превосходящих n и не делящихся ни на одно из чисел 6, 10, 15, равно $22m$.

106. При обследовании читательских вкусов студентов оказалось, что 60% студентов читают журнал А, 50% — журнал В, 50% — журнал С, 30% — журналы А и В, 20% — журналы В и С, 40% — журналы А и С, 10% — журналы А, В и С. Сколько процентов студентов а) не читают ни одного из журналов; б) читают в точности два журнала; в) читают не менее двух журналов?

107. На одной из кафедр университета работают тринадцать человек, причем каждый из них знает хотя бы один иностранный язык. Десять человек знают английский, семеро — немецкий, ше-

стеро — 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?

108. Имеются $3n + 1$ предметов (n одинаковых, остальные различные). Доказать, что из них можно извлечь n предметов 2^{2n} способами.

109. Применяя формулу включения и исключения, определить количество целочисленных решений системы уравнений и неравенств: $x_1 + x_2 + \dots + x_n = r$, $a_i < x_i \leq b_i$, $i = 1, n$, a_i, x_i, b_i — целые числа.

110. Определить количество целочисленных решений системы $x_1 + x_2 + x_3 = 40$, $x_1 > 3$, $x_2 > 0$, $x_3 > 2$.

111. 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?

112. Имеем n предметов, расположенных в ряд. Сколькими способами можно выбрать из них три предмета так, чтобы не брать никаких двух соседних элементов?

113. Даны $2n$ различных предметов $a_1, a_1, a_2, a_2, \dots, a_n, a_n$. Сколько существует перестановок из этих $2n$ предметов, в которых не стоят рядом одинаковые элементы?

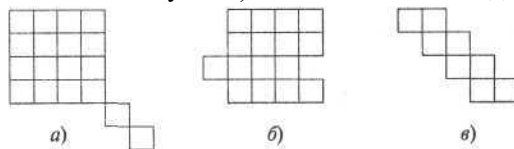
114. В шахматной олимпиаде участвуют представители n стран по 4 представителя от каждой страны. Сколькими способами они могут встать в ряд так, чтобы рядом с каждым был представитель той же страны?

115. Имеется n одинаковых вещей и еще n различных вещей. Сколькими способами можно выбрать из них n вещей? Сколькими способами можно упорядочить все $2n$ вещей?

116. Найти число способов распределения $2n$ одинаковых шаров по двум неразличимым корзинам.

117. В каждой клетке шахматной доски размером $n \times n$ поставили число, указывающее количество прямоугольников, в которые входит эта клетка. Чему равна сумма всех поставленных чисел?

118. Найти число расстановок k ладей так, чтобы они не били друг друга, на доске $n \times l$ (см. случаи а, б, в) с выколотыми или добавленными клетками. В случае в) использовать n ладей.



Производящие функции и рекуррентные соотношения

119. Найти производящую функцию последовательности $\{2(n-5) + 7^{n+2}\}$.

120. Применить технику производящих функций для нахождения суммы чисел $1 + 2^3 + \dots + l$.

121. 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) $u_{n+2} - 4u_{n+1} + 4u_n = 0, u_0 = 1, u_1 = 2.$

122. Решить неоднородные рекуррентные соотношения:

- 1) $u_{n+1} = u_n + n, u_0 = 1;$
- 2) $u_{n+2} = -2u_{n+1} + 8u_n + 27 \cdot 5^n, u_0 = 0, u_1 = -9;$
- 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) $u_{n+2} = u_{n+1} - \frac{1}{4}u_n + 2^{-n}, u_0 = 1, u_1 = 3/2;$
- 6) $u_{n+2} - 3u_{n+1} + 2u_n = (-1)^n, u_0 = 1, u_1 = 2.$

123. 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 $n = m \cdot k$.

124. Найти общее решение рекуррентных соотношений:

- 1) $u_{n+2} = u_{n+1} - \frac{1}{4}u_n,$
- 2) $u_{n+2} - 4u_{n+1} + 3u_n = 0;$
- 3) $u_{n+2} - u_{n+1} - u_n = 0.$

125. Найти решение системы рекуррентных соотношений:

$$\begin{cases} a_{n+1} = 3a_n + b_n \\ b_{n+1} = -a_n + b_n \end{cases}, \text{ где } c_0 = 1^A, h_0 = 6.$$

126. Найти число решений уравнения $x + 2y = n$, где $x, y, n \in \mathbb{Z}^+$ — положительные целые числа; x, y — неизвестные.

127. Найти число решений уравнения $x + 2y + 4z = n$, $x, y, z \in \mathbb{Z}^+$, x, y, z — неизвестные.

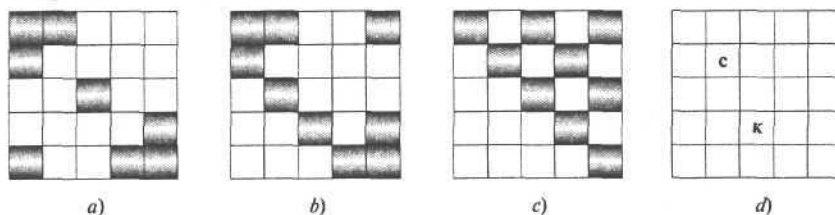
128. Найти определитель матрицы

$$A_n = \begin{pmatrix} \alpha + p & \alpha\beta & & & & \\ & 1 & a + p & ap & & \\ & & 1 & a + p & ap & \\ & & & \dots & & \\ & & & & \dots & \\ & & & & & 1 & a + p & ap \\ & & & & & & 1 & \alpha + \beta \end{pmatrix}$$

где a, p — произвольные числа; вне обозначенных диагоналей матрицы располагаются нули.

129. Вычислить сумму $\sum (1/2^k)$, где суммирование производится по всем натуральным k , не кратным 2, 3 и 5.

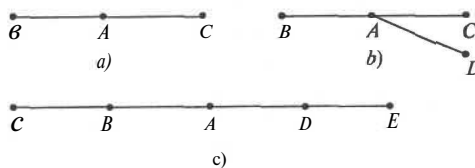
130. Найти ладейный многочлен запрещенных позиций для досок $a)$ и $b)$ и для каждого из этих случаев составить многочлен попаданий. Для доски $c)$ составить многочлен запрещенных позиций. Для досок $a), b)$ и $c)$ найти число расстановок трех ладей на запрещенных позициях.



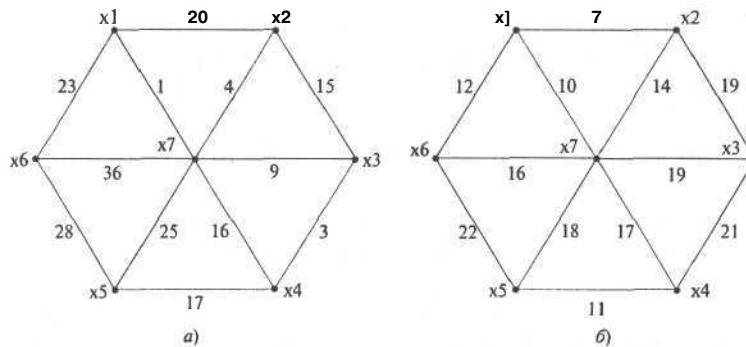
131. Найти число способов расставить 5 ладей на доске 5×5 так, чтобы ни одну из них не бил слон. Позиция слона указана на доске $d)$ символом «с».

132. Найти число способов расставить 5 ладей на доске 5×5 так, чтобы ни одну из них не бил конь. Позиция коня указана на доске $d)$ символом «к».

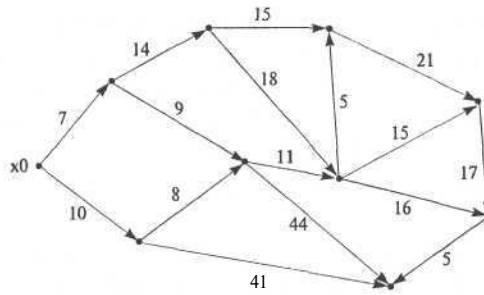
133. Найти число замкнутых маршрутов длины $2l$ по ребрам графа для случаев $a), b)$ и $c)$. Длина ребра равна 1. Начало и конец пути есть вершина A .



163. Найти хроматическое число n -вершинного дерева, $n > 1$.
164. Доказать, что в компании из 6 человек всегда найдутся либо трое знакомых друг с другом, либо трое друг с другом не знакомых.
165. Доказать, что при любой раскраске ребер полного 6-графа в два цвета найдется монохроматический 3-граф.
166. Рассмотрим граф M_n Муна–Мозера с $3n$ вершинами $\{1, 2, \dots, 3n\}$, в котором вершины разбиты на триады $\{1, 2, 3\}, \{4, 5, 6\}, \dots, \{3l - 2, 3l - 1, 3n\}$; 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.
167. Показать, что в графе с l вершинами и с k компонентами связности число ребер не более $((n - k)(n - k + 1))/2$.
168. 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.
169. 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.
170. Для каждого графа а) и б) найти остовное дерево, используя программную реализацию жадного алгоритма (алгоритм б.7) и алгоритма ближайшего соседа (алгоритм б.9). На каждом шаге алгоритмов отобразить состояние используемых структур данных.

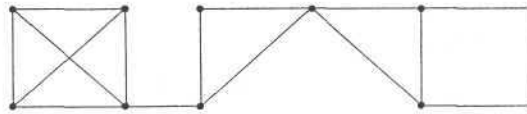


171. Найти кратчайшие пути от вершины x_0 до всех остальных вершин ориентированного графа (см. рис. на следующей странице).



172. Пусть $A^k = [a_{ij}^{(k)}]$, $i, j = 1, n$ 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 .

173. Выполнить хроматическое разложение графа. Найти все клики, фундаментальное множество циклов, листья, блоки и мосты. Определить центры, радиус и диаметр графа.



174. Сколь много ребер может иметь n -вершинный граф, у которого степень всякой вершины не превосходит d ?

175. Показать, что в дереве с нечетным диаметром любые две простые цепи наибольшей длины имеют хотя бы одно общее ребро.

176. Пусть корневое дерево с n ($n > 2$) висячими вершинами не имеет вершин степени 2, отличных от корня. Показать, что общее число вершин дерева не превосходит $2n - 1$.

177. Доказать, что дерево обладает единственным центром в случае, когда его диаметр есть число четное, и обладает двумя центрами, когда диаметр есть нечетное число.

178. Доказать, что в любом дереве с $n > 2$ вершинами имеется не менее двух висячих вершин.

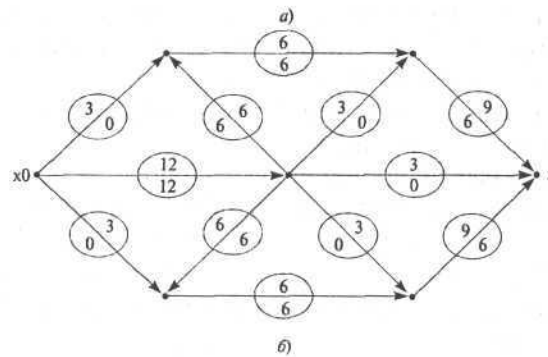
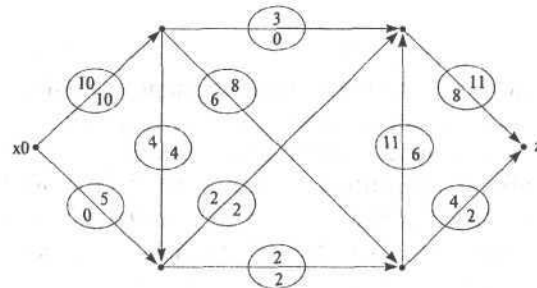
179. Вершины графа Γ пронумерованы в порядке возрастания их степеней. Доказать, что если k — наибольшее число, такое, что $k < d(x_k) + 1$, то $\chi(\Gamma) \leq k$, где $d(x_k)$ — степень вершины x_k .

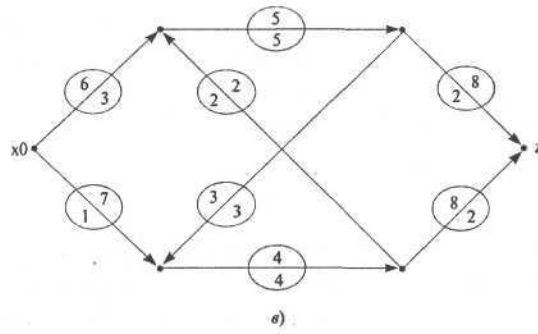
180. Доказать, что если для любых двух вершин x и y связного n -вершинного графа выполняется $d(x) + d(y) > n$, то граф имеет гамильтонов цикл.

181. Используя алгоритм чередующихся цепей (см. п.6.14.3), 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 $\pi = \{(s_1, 1), (s_2, 2), (s_4, 3), (s_5, 4), (s_6, 5)\}$.

182. Используя алгоритм чередующихся цепей (см. п.6.14.3), 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 $\pi = \{(s_1, 1), (s_2, 2), (s_3, 5), (s_5, 3), (s_6, 7), (s_7, 6)\}$.

183. Перераспределить заданный начальный поток по транспортной сети а), б) и в) так, чтобы он стал максимальным, а сеть — насыщенной (см. п.6.10). Найти все разрезы транспортной сети и их мощности, сравнить мощность минимального разреза с найденным максимальным потоком.





184. Решить задачи о назначениях максимального выбора

a)

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

185. Решить задачи о назначениях минимального выбора

a)

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

186. 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	Гружёный
X	20	40
Y	30	60

Расстояния от стоянки грузовиков до места назначения приведены ниже:

Стоянки	Расстояние				
	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.

187. 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.

188. Авиалиния связывает три города А, В, С. Полеты происходят семь дней в неделю согласно расписанию:

Вылет		Прибытие	
Город	Время	Город	Время
А	8.00	В	12.00
А	9.00	С	12.00
А	10.00	В	14.00
А	14.00	В	18.00
А	18.00	В	22.00
А	20.00	С	23.00
В	7.00	А	11.00
В	9.00	А	13.00
В	13.00	А	17.00
В	18.00	А	22.00
С	9.00	А	12.00
С	15.00	А	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

189. Показать, что $G_1 = \{n \mid n \in \mathbb{Z}\}$ — группа с операцией сложения чисел, где \mathbb{Z} — множество целых чисел; $G_2 = \{2n \mid n \in \mathbb{Z}\}$ — группа с операцией сложения чисел; $G_3 = \{2^n \mid n \in \mathbb{Z}\}$ — 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 операций сложения и умножения (без нуля). Установить, какие из групп являются подгруппами других групп.

190. Пусть M — подмножество группы G и $\forall a, b \in M$ выполняется $ab^{-1} \in M$. Показать, что M — подгруппа.

191. Пусть G_1, G_2 — группы и отображение $f: G_1 \rightarrow G_2$ — гомоморфизм. Показать, что ядро и образ гомоморфизма являются подгруппами.

192. 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 $\varphi: G_1 \rightarrow G_2$ такое, что $\forall x \in G_1 \varphi(x) = \ln(x) \in G_2$. Показать, что φ — изоморфизм групп.

193. Пусть G — группа и $r \in G$ — фиксированный элемент. Определим отображение $\varphi: G \rightarrow G$ формулой $\forall g \in G \varphi(g) = r^{-1}gr$. Докажите, что φ — изоморфизм группы на себя.

194. Докажите, что если порядок группы G равен $2n$ и H — подгруппа порядка n группы G , то H — ее нормальный делитель.

195. Доказать, что если в квадратной матрице порядка n содержится нулевая подматрица размера $r \times s$ и $r + s > n$, то определитель матрицы равен нулю.

196. Найти порядок группы, порожденной подстановкой $\begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 1 \end{pmatrix}$.

197. Разложить подстановки $\begin{pmatrix} 1 & 2 & 4 & 7 & 5 & 6 & 8 \\ 3 & 2 & 4 & 7 & 5 & 6 & 8 \end{pmatrix}$ и $\begin{pmatrix} 1 & 6 & 2 & 4 & 4 & 3 & 7 \\ 7 & 6 & 2 & 4 & 1 & 4 & 3 & 5 \end{pmatrix}$ в произведение независимых циклов и в произведение транспозиций, определить их четность.

198. Пусть $H \in S_4$, где S_4 — симметрическая группа. Будет ли H подгруппой в следующих случаях:

$$a) H = \left\{ \begin{pmatrix} 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 4 & 1 & 2 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix} \right\};$$

$$b) H = \left\{ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 1 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix}, \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 2 & 1 & 3 \end{pmatrix} \right\}.$$

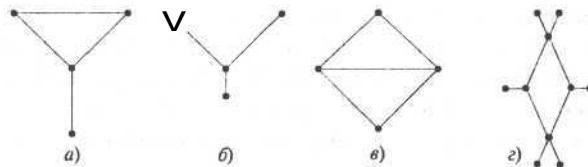
199. Пусть G — циклическая группа, $|G| = m$. Показать, что число образующих группы равно $\varphi(m)$ — функция Эйлера.

200. Пусть G — циклическая группа. Показать, что элементы группы одинакового порядка являются образующими одной и той же подгруппы $H \subset G$.

201. Пусть G — группа порядка $|G| = p$, p — простое число. Показать, что G — циклическая группа.

202. Показать, что циклическая группа — коммутативна.

203. Пусть G — группа. Показать, что для всякого $g \in G$ найдется такое целое k , что $g^k = e$.
204. Показать, что число образующих циклической группы $G = \{g, g^2, \dots, g^n = e\}$ равно значению функции Эйлера $\varphi(n)$ — количество чисел из множества $\{1, 2, \dots, n-1\}$ взаимно простых с n .
205. Пусть G — конечная абелева группа порядка $|G| = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, где p_1, p_2, \dots, p_k — простые различные числа; множество $A(p) = \{x \in G \mid |x| = p^\alpha\}$, где α принимает произвольные целые значения. Показать, что $A(p)$ — подгруппа.
206. 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.
207. Определить множество левых и правых смежных классов симметрической группы S_3 по подгруппе H , где H — циклическая подгруппа с образующим элементом $\begin{pmatrix} 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix}$. Доказать что H — нормальный делитель.
208. Найти цикловой индекс группы самосовмещений (автоморфизмов), действующей на множестве $a), б), в), г)$. Отдельно рассмотреть случаи действия группы на плоскости и в пространстве.



209. Применить теорему Пойа для определения количества возможных раскрасок двумя и тремя красками вершин графов а), б), в), г) предыдущей задачи.
210. 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

211. *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» $k, k + 1, \dots, m$ заняты), то муж приносит свои извинения и едет дальше.

Предположим, что это происходит с я различными машинами, причем j -я жена просыпается в момент, когда машина находится перед местом a_j . Сколько имеется таких последовательностей a_1, a_2, \dots, 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, \dots, a_n и соответствующие им расположения автомобилей. Исходные данные — m, n . Например, если $m = n = 9$ и $(a_1, a_2, \dots, a_n) = (3, 1, 4, 1, 5, 9, 2, 6, 5)$, то автомобили расположатся следующим образом: 2, 4, 1, 3, 5, 7, 8, 9, 6.

212. *Фальшивая монета.* Банк «Золотой Ящик» получил информацию, что в последней партии из n монет 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, какие из них больше, меньше или равны.

Ваша задача — написать алгоритм—программу, которая по результатам взвешиваний, выполненных 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 ($n > 2$) 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 ($1 \leq p_i \leq n/2$), 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 чашки весов.
- Знак $>$ означает, что вес левой чашки больше веса правой чашки весов.
- Знак $=$ означает, что вес левой чашки равен весу правой чашки весов.

Ниже приведен пример файла входных данных.

Результаты определения номера фальшивой монеты сохранить в текстовом файле. Если по выполненным взвешиваниям нельзя выявить фальшивую монету, то в качестве результата сохранить 0. Ниже приведен пример файла результатов.

<u>ФАЙЛ</u> <u>ИСХОДНЫХ</u> <u>ДАнных</u>	<u>ФАЙЛ</u> <u>ИСХОДНЫХ</u> <u>ДАнных</u>
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
=	<
<u>ФАЙЛ</u> <u>РЕЗУЛЬТАТОВ</u>	2 4 5 2 6
3	>
	<u>ФАЙЛ</u> <u>РЕЗУЛЬТАТОВ</u>
	0

213. *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:*

- 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;
- 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 имеет вид:

3.

214. Спуск с горы.

```
      1
     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 налево или вниз направо. 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 файла исходных данных:

```
5
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5
```

Пример файла выходных данных:

```
30
7 3 8 7 5
```


215. Перестановка корзин. Даны 2л корзин с шарами *A* и *B*, расположенные в ряд, как на рисунке ($n < 5$).

$$\backslash A \backslash B \backslash B \backslash A \backslash \quad \backslash \quad | A | B | A | B |$$

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.

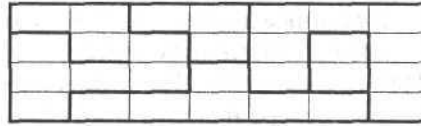
Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: 2л — число корзин с пустыми местами. Вторая строка: $| A | 2? | B | A | | | A | 5 | A | B |$ — исходное расположение корзин. Расчетные данные сохранить в текстовом файле со следующей структурой. Каждая строка — состояние корзин после каждой перестановки.

216. Комнаты музея. Составьте алгоритм—программу определения числа комнат в музее и площади 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: l 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 файла исходных данных:

```
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
```

Пример файла выходных данных:

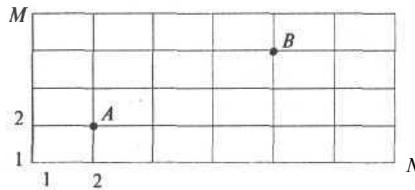
```
5
9 6 3 2 8.
```

217. *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 ($1 < M \leq N$) любых бокалов переворачиваются так, что те бокалы, которые стояли дном вниз, оказываются перевернутыми вверх дном, а остальные из M бокалов ставятся вниз дном.

Составить алгоритм-программу, которая за минимальное количество шагов позволяет перевернуть все бокалы вверх дном или определяет, что это сделать невозможно.

Исходные данные N, M задаются в текстовом файле. Результаты последовательных переверотов сохранить в текстовом файле.

218. *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: $H[i, j]$, $i = 1, M, j = 1, N$ — 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).

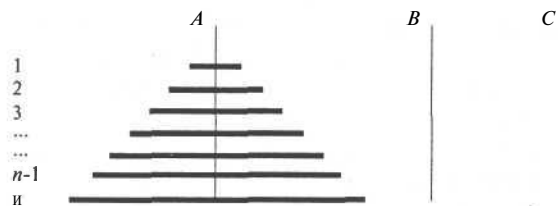


Исходные данные представлены в текстовом файле со следующей структурой. Первая строка: M, N — размерность матрицы и

улиц. Вторая строка: $A = (ia, ja), B = (ib, jb)$ — координаты двух перекрестков. Третья и следующие строки определяют значения элементов матрицы высот $H[i, j], i = 1, M, j = 1, N$. Результаты расчетов сохранить в выходном текстовом файле со следующей структурой. Каждая строка — координаты и высоты перекрестков, через которые пролегает маршрут.

219. *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 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ 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) .

220. *Ханойская башня*. Задача о ханойской башне проиллюстрирована на следующем рисунке:



Имеются три колышка A, B, C . На колышек A нанизано n дисков радиуса $1, 2, \dots, n$ (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.

221. *Числа*. Дана последовательность n различных между собой целых чисел a_1, a_2, \dots, a_n . Трансформацией называется следующая последовательность действий: некоторые из чисел увеличивают-

ся на 1, если в последовательности оказываются два одинаковых числа, то одно из них исключается из последовательности. Выполним данную трансформацию k раз, с целью достигнуть минимального количества чисел в последовательности a_1, a_2, \dots, a_n . Составить алгоритм–программу, которая вычисляет p — количество чисел, оставшихся в последовательности. Исходные данные n, k и последовательность a_1, a_2, \dots, a_n задаются в текстовом файле. Найденное число p сохранить в текстовом файле.

Пример файла исходных данных:

```
5 2
7 1 15 8 3.
```

Пример файла выходных данных:

```
3.
```

222. *Знакомства.* Имеется n человек, где $1 < n < 1000$. 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 познакомиться 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 находиться 2 числа, разделенные пробелом — номера людей, которым необходимо познакомиться. Приведем примеры расчетов для конкретных исходных данных.

Пример входного файла:	Пример выходного файла:
4	3
3	1 2
2	1 3
2	1 4
1	2 3

223. *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. Первая строка содержит целое число n , $1 < n < 1000$, равное количеству жителей городка. Следующие n строк содержат по одному числу — номеру жетона, т.е. в $(i + 1)$ -й строке, соответствующей i -му жителю, находится номер того жетона, который он получил по почте. Выходной файл данных содержит одну строку с найденным числом дней m .

224. *Кокосы.* В газетах 9 октября 1926 года 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.

Файл исходных данных	Файл результатов
25 кокосов	3 человека и 1 обезьянка.
20 кокосов	Нет решений.
3121 кокосов	5 человека и 1 обезьянка.
-1	

225. *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-GreenYellowRed (LGYR)* для каждого светофора, где L — положительное действительное число, указывающее место расположения светофора от начала маршрута Филадельфии; G, Y, R — интервал продолжительности времени в секундах непрерывного цвета светофора: зеленого, желтого и красного. Результаты расчетов допустимых целочисленных скоростей движения Филадельфии сохранить в выходном текстовом файле:

Файл исходных данных	Файл выходных данных
1	30, 32, 33, 36, 37, 38, 41, 42, 43, 44, 45,
5.5 40 8 25	48, 49, 50, 51, 52, 53, 54, 59, 60
3	0 — признак отсутствия такой скорости.
10.7 10 2 75	
12.5 12 5 57	
17.93 15 4 67	
— 1	

226. *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 $N = 31$.

Файл исходных данных	Файл выходных данных
7	3 4
31	2 3 5 6 7 8

227. *Кот и Лиса.* Дан ориентированный граф $G = (X, U, \Phi)$, 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 $X = \{1, 2, \dots, n\}$, $1 \leq n \leq 1000$.

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, \dots, b_n и $x_1y_1 x_2y_2 \dots x_my_m$, 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.

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
```

Результат представить в текстовом файле, в котором записать n целых чисел c_1, c_2, \dots, c_n , где $c_i = 1$, если позиция i выигрышная для Лисы; иначе $c_i = 0$.

Для примера результаты расчетов представляются строкой:

```
0 1 0 0 1 1 1
```

228. *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 пропала. 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) 01210	4	1	10	7	5	
2) 21120						
3) 21001	6	1	0	1	0	1
4) 12110	9	1	1	0	0	1
5) 12101	3	1	1	0	0	1
6) 12101	2	1	1	1	1	0
7) 00011	8	0	0	0	1	1
8) 22222						
9) 11001						
10) 10010						

229. Длина объединения. Текстовый файл содержит целые числа: $a_1, b_1, a_2, b_2, \dots, a_n, b_n$. 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 файла исходных данных:

```
3
0 2
-1 1
0 1.
```

Пример файла выходных данных:

```
3.
```

230. 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.

231. Прямоугольные области. Условие данной задачи — это условие предыдущей задачи. Однако требуется составить алгоритм–программу определения количества областей, на которые границы прямоугольников разбивают плоскость.

232. *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 строк 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 файла исходных данных:

```
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 для данного примера:

3.

233. *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 файла исходных данных:

```
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
```

Пример файла результатов:

11 — площадь участка

234. *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 файла исходных данных:

```
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
```

Пример файла результатов:

1 — ломаные совпадают.

235. *Деление на равные половины.* Текстовый файл содержит последовательность целых чисел — веса элементов: a_1, a_2, \dots, a_n . Составить алгоритм—программу деления этих предметов на две группы так, чтобы общие веса двух групп были максимально 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 файла исходных данных:

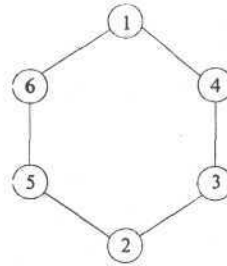
```
5
6 3 1 4 5.
```

Пример файла выходных данных:

```
6 3 1 — первая группа
4 5 — вторая группа.
```

236. *Простой круг.* Натуральные числа $1, 2, \dots, y$ (y — чётное число) располагают по кругу как на диаграмме. Первое число по кру-

гу 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 n задается в текстовом файле. Результаты расчетов сохранить в текстовом файле.



Пример файла исходных данных	Пример файла результатов
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

237. *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 «с» и «т», 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 начинаться и закончиться на одном и том же перекрестке. Стоимость проезда по улице равна ее длине.

Пример входного файла	Пример выходного файла
3 one two three	11

238. *Пересечение с отрезками.* Имеются N отрезков, концы которых задаются двумя парами точек на плоскости: $(x1[i], y1[i])$ и $(x2[i], y2[i])$. Требуется составить алгоритм—программу определе-

НИЯ 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.

239. Простые суммы. Даны числа от 1 до n , $1 < n < 5000$. 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 файла исходных данных:

8.

Пример файла выходных данных:

1 2 2 1 1 1 1 2.

240. 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 M и N . 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 файла исходных данных:

```
8 2
-#---
-----
---#-
#-##-
-----
---##
-----
-----
```

Выходной файл для данного примера:

18.

241. 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 с 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 файла исходных данных:

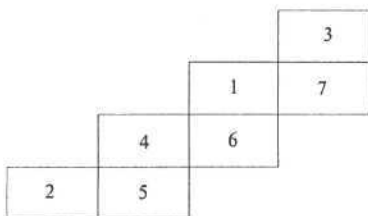
```
3 4 7
1 1 1 1
1 1 2 1
1 1 2 3
```

Выходной файл для данного примера:

12.

242. Оставить условие задачи №241 и предписать пирату, чтобы он за K шагов еще и вернулся в начальную комнату лабиринта.

243. *Задача Д.Андре.* Составить алгоритм—программу поиска всех способов заполнения числами $1, 2, \dots, n$ массива из n ячеек



так, чтобы во всех строках и столбцах они располагались в возрастающем порядке (слева-направо и сверху-вниз). Исходное число n задается в текстовом файле. Результаты заполнения сохранить в файле результатов.

244. *Сумма чисел.* Дано натуральное число m . Вставить между некоторыми цифрами: $1, 2, 3, 4, 5, 6, 7, 8, 9$, 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 .

245. *Обслуживание авиалиний.* Имеется некоторый город M , который связан маршрутами с городами A_1, A_2, \dots, 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 $[a_i, b_i], i = 1, 2, \dots, n$. Требуется составить алгоритм—программу определения минимального числа самолетов, достаточного для обслуживания всех рейсов.

Файл исходных данных	Файл результатов
3 — количество городов	1 самолет на все рейсы
1 2 — интервалы времени	
2 3	
3 5	

5 — количество городов	2 самолета на все рейсы
1 3 — интервалы времени	
7 12	
6 8	
2 4	
9 10	

246. 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 возможного числа гостей на юбилейном вечере и сохранить его в выходном файле результатов.

Файл исходных данных	Файл результатов
5 — число знакомых у генерала	2 приглашенных на вечер
6 — число симпатизирующих пар	
1 2	
1 3	
2 4	
2 5	
3 4	
3 5	

247. Таблица инверсий. Составить алгоритм—программу определения таблицы инверсий $d_1 d_2 \dots d_n$ перестановки (a_1, a_2, \dots, a_n) , где d_j — число элементов, больших j и расположенных левее j (см. п. 1.13). Исходные данные — число n и произвольная перестановка чисел $1, 2, \dots, n$ — определяются в текстовом файле. Выходной файл результатов должен содержать найденную таблицу инверсий.

Пример файла исходных данных:

5 — число n
5 3 4 2 1 — перестановка

Пример файла результатов:

4 3 1 1 0 — таблица инверсий.

1. Множество D всех сочетаний C_n^k разобьем на два непересекающихся подмножества $D = D_1 \cup D_2$, где $D_1 \cap D_2 = \emptyset$. 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, $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$.

2. Используйте бином Ньютона $(1+x)^n = \sum_{k=0}^n C_n^k x^k$.

3. Применить индукцию по n . 7. 2^n .

8. Воспользуйтесь тождеством $(a+b)^n = \sum_{k=0}^n C_n^k a^k b^{n-k}$.

9. Воспользуйтесь тождеством $(1+x)^m(1+x)^n = (1+x)^{m+n}$.

10. Воспользуйтесь тождеством $(1+(1+1))^n = 3^n$.

11–13. Воспользуйтесь тождеством $(1+x)^n = \sum_{k=0}^n C_n^k x^k$.

14. *Указание.* Воспользуйтесь полиномиальным разложением формулы $(1+1+\dots+1)^n = k^n$.

15. 21 . 16. C_{n+2}^2 . 17. $2n^2 \times 2n^2$. 18. $2n^2 \times (2n^2 - 2n)$. 19. 2^{nm} .

20. $A_5^3, C_4^2 3!$. 21. 3^6 . 22. 63 . 23. C_n^2 — двусторонних словарей, A_n^2 — односторонних словарей, n — словарей при переводе по циклу.

24. A_m^4 . 25. $P(5,7,3) = \frac{15!}{5!7!3!} = C_{15}^5 C_{10}^7 C_3^3$. 26. C_{n+m}^m . 27. C_n^r .

28. $A_m^1 + A_m^2 + A_m^3$ — учитывается порядок имен. 29. $P(2,2,2,1,1)$.

30. $C_n^k C_m^k k!$. 31. $2n!n!$ и $2n!n!/2n = n!(n-1)!$. 32. $C_{12}^4 C_{15}^4 4!$.

33. $n!/2n$, где деление на число n обозначает совпадение соседей при циклическом перемещении исходной перестановки; в цифре 2 учитываются одни и те же соседи при отраженном (зеркальном) расположении исходной перестановки.

34. $C_{10}^6 (C_{10}^6 - 1)(C_{10}^6 - 2) = A_{C_{10}^6}^3$. 35. $C_{3n}^n C_{2n}^n C_n^n$.

36. $C_n^6 ((C_6^2 C_4^2 C_2^2) / 3!) = 15C_n^6$. 37. $C_n^6 ((C_6^2 C_4^2 C_2^2) / 3!) = 15C_n^6$.

38. $((2n)! / (2!)^n) / n!$ — неупорядоченное разбиение множества.

39. См. решение задач 7 и 8. 40. $9! - C_3^1 7!3! + C_3^2 5!3!3! - C_3^3 3!3!3!$
 41. $C_4^1 C_{48}^1, C_{52}^9 C_{48}^{10}, C_{52}^{10} C_{48}^{10}, C_4^1 C_{48}^9, C_4^2 C_{48}^8$.
 42. $(C_4^1 C_{13}^3)(C_{13}^1)^3 + (C_4^2 (C_{13}^2)^2)(C_{13}^1)^2$. 43. 3^m 44. 3^{17} . 45. m^n .
 46. C_{n+m-1}^{m-1} . 47. $C_{(n-m)+m-1}^{m-1} = C_{n-1}^{m-1}, C_{(n-k)+m-2}^{m-2}, C_{(n-\sum a_i)+m-k-1}^{m-k-1}$.
 48. $C_{n_1+m-1}^{m-1} C_{n_2+m-1}^{m-1} C_{n_3+m-1}^{m-1}$. 49. $C_8^2 (C_3^2)^6$. 50. C_{n+m-1}^{m-1} .
 51. $5^4, A_5^4$. 52. C_{36}^5 . 53. $C_5^3 C_{31}^2, C_{36}^5 - C_{31}^5 - C_5^1 C_{31}^4 - C_5^2 C_{31}^3$
 55. $P(3, 2, 3, 1)$. 56. $C_{10}^3 C_7^4 C_3^3$. 57. $(1 + \alpha_1)(1 + \alpha_2) \dots (1 + \alpha_n)$,
 $\frac{1-p_1^{\alpha_1}}{1-p_1} \frac{1-p_2^{\alpha_2}}{1-p_2} \dots \frac{1-p_n^{\alpha_n}}{1-p_n}$. 59. $n \lfloor n/p \rfloor$ — столько чисел среди $1, 2, \dots$,
 и не кратных p . Всего произведений C_n^k , тогда число произведе-
 ний, кратных p , равно $C_n^r - C_{n-\lfloor n/p \rfloor}^k$. 60. $l! - m!(n - m + 1)!$ (объеди-
 нить m элементов в один элемент).
 61. $2(C_n^2)^2 = \frac{n^2(n-1)^2}{2}$ — способов поставить две ладьи так, чтобы
 они не угрожали друг другу. $(C_n^1)^2(n-1) = n^2(n-1)$ — способов по-
 ставить две ладьи так, чтобы они угрожали друг другу. Для ответа
 на вопрос задачи осталось сравнить указанные числа.
 63. $C_{10}^2 + C_{10}^2 = 90$. 66. 18. 68. 31. 71. $C_{n+k-1}^{k-1}, C_{n-1}^{k-1}$.
 72. 205223000, см. п. 1.13. 73. 27354186, см. п. 1.13.
 75. Указание: воспользуйтесь тем, что перестановка двух соседних
 меток N и M не оказывает влияния на произведение 2^{n-m} .
 76. $3C_{3n}^n 2^n$. 77. $(kn)! / ((k!)^n n!)$. 78. $(30)! / ((10!)^3 3!)$, $(30)! / ((3!)^{10} 10!)$.
 79. $(C_4^2 C_2^2 / 2!) \cdot (C_{32}^{16} C_{16}^{16})$. 80. $(C_{10}^2 C_8^2 C_6^2 C_4^2 C_2^2) / 5! = 10! / ((2!)^5 5!)$.
 81. $9! / ((1!)^1 (2!)^4 (4!)^1)$. 82. $9! / ((3!)^3 3!)$. 83. $P(2, 2, 2, 2, 2, 1, 1, 1, 1)$.
 84. $C_{15}^8 C_{10}^8$. 85. $4^8 - C_4^1 3^8 + C_4^2 2^8 - C_4^3 1^8$ — воспользоваться форму-
 лой включений и исключений для свойств: P_i — пустой i -й этаж,
 $i = \overline{1, 4}$. 87. $\frac{n!}{n_1! n_2! \dots n_k!}$. 88. $(9^0 + 9^1) + 9^2 + 9^3 + \dots + 9^n = \frac{9^{n+1} - 1}{9 - 1}$.
 89. C_{n+9-1}^{9-1} . 90. $C_{n+10-1}^{10-1} - 1$, где -1 обозначает число 0.
 91. C_{n+1}^k . 92. C_{n+k}^k . 93. Схема посева сортов пшеницы должна соот-
 ветствовать латинскому квадрату $m \times m$.
 94. C_{m+6-1}^{6-1} . 95. k^m . 96. $\sum_{i=0}^k (-1)^i C_k^i (k-i)^m$.

$$97. \sum_{i=0}^{k-r} (-1)^i C_{r+i}^r C_k^{r+i} (k-r-i)^m = C_k^r \sum_{i=0}^{k-r} (-1)^i C_{k-r}^i (k-r-i)^m.$$

$$98. \sum_{k=0}^m (-1)^k C_m^k m^{m-k} = m^m \sum_{k=0}^m (-1)^k \frac{m^k}{m^k}.$$

$$99. \sum_{k=0}^m (-1)^k C_m^k (m-k)! = m! \sum_{k=0}^m \frac{(-1)^k}{k!} \approx m! e^{-1}.$$

$$100. \sum_{k=0}^{m-r} (-1)^k C_{r+k}^r C_m^{r+k} (m-r-k)! = \frac{m!}{r!} \sum_{k=0}^{m-r} \frac{(-1)^k}{k!} \approx \frac{m!}{r!} e^{-1}.$$

102. Введем обозначения. P_k — свойство, что k -я пара враждующих рыцарей сидит рядом, $k = 1, 2, \dots, l$. ДО — размещение рыцарей, которые не обладают ни одним из указанных свойств, т.е. требуемые размещения по условию задачи. ДО) = $W(0) - W(1) + W(2) - \dots + (-1)^n W(n)$, 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 $2l - k$ объектов, которые можно расположить $(2l - k)!$ способами. В каждой из k заданных пар врагов можно поменять местами 2^k способами. Выбор k враждующих пар можно сделать C_m^k способами. Следовательно, искомое число равно ДО) = $\sum_{k=0}^n (-1)^k C_m^k 2^k (2l - k)!$.

103. 542. 104. 734. 106. 20%, 60%, 70%. 107. 2, 6, 3.

109–110. Решение подобного уравнения рассмотрено в п.1.7.

111. $\frac{10!}{(1!)^5 \cdot 1!} \cdot \frac{10!}{(2!)^5}$. 112. C_{n-2}^3 . Действительно, выбирая из $l-2$ 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 $k_1 < k_2 + 1 < k_3 + 2$.

$$113. \sum_{i=0}^n (-1)^i C_n^i \frac{(2n-i)!}{(2!)^{n-i}}. \quad 114. \left(\frac{4!}{(2!)^2 2!} \cdot 2 \cdot 2 \right)^n (2n)! = (12)^n (2n)!$$

115. $2^n, \frac{(2n)!}{n!}$. 116. $n + 1$. 117. Количество прямоугольников размера $i \times j$ равно $(l - i + 1) \times (n - j + 1)$. Каждый прямоугольник учитывается в сумме столько раз, какова его площадь. Тогда сумма равна

$$\sum_{i=1}^n \sum_{j=1}^n (n-i+1)(n-j+1) \cdot i \cdot j = \left(\sum_{j=1}^n (n-i+1)i \right)^2 = \left(\frac{n(n+1)(n+2)}{6} \right)^2.$$

118. 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) $n+1$.
119. $\frac{4x}{1-x} - \frac{10}{1-7x} + \frac{4y}{1-7x}$. 120. $(n(n+1)/2)^2$. 121. 1) $7+3^n$;
- 2) $u_{2n} = [9 \cdot 3^{2n} + (-1)^n]/10, u_{2n+1} = [9 \cdot 3^{2n+1} + 3 \cdot (-1)^{n+1}]/10, n > 0$;
- 3) $(\mp 9)^n$; 4) $(\mp 4)^n$; 5) $(-1)^n(n-1) + 2$; 6) 2^n .
122. 1) $C_{n+2}^2 - 2C_{n+1}^2 + 2C_n^2$; 2) $-2(-4)^n + 3 \cdot 2^n + 5^n$;
- 3) $2^n - \frac{n^2+n}{2}$; 4) $2^n + 2^{n-2}C_n^2$; 5) $\frac{(k+2)(k+1)}{2^{k+1}}$; 6) $\frac{2^{n+3} - 3 + (-1)^n}{6}$.
124. 1) $A \frac{n}{2^{n-1}} - B \frac{n-1}{2^n}$; 2) $A + B \cdot 3^n$; 3) $A \left(\frac{1+\sqrt{5}}{2} \right)^n + B \left(\frac{1-\sqrt{5}}{2} \right)^n$.
125. $a_n = 2^{n+1}(2n+7), b_n = 2^{n+1}(-2n-3)$.
126. $\frac{2n-3+(-1)^n}{4}$. 128. $\frac{\alpha^{n+1} + \beta^{n+1}}{\alpha + \beta}$.
129. Использовать формулу включений и исключений.
130. а) $R(x, C) = x(1+x)^3 + (1+3x+x^2)(1+x)$.
133. а) 2^n , с) $2 \cdot 3^{n-1}$. Составить рекуррентное соотношение (см. п.3.3).
134. а) $\frac{2^n + (-1)^n 2}{3}$, б) $\frac{3}{4}(3^{n-1} - (-1)^{n-1})$. Составить рекуррентное соотношение (см. п.3.3).
135. 4^n .
136. $C_{2n}^n \sum_{k=0}^n (C_n^k)^2 = (C_{2n}^n)^2$. Воспользуйтесь производящей функцией $\left(x + \frac{1}{x} + y + \frac{1}{y} \right)^{2n} = \sum_{k=0}^{2n} C_{2n}^k \left(x + \frac{1}{x} \right)^k \left(y + \frac{1}{y} \right)^{2n-k}$ всех ломаных маршрутов длины $2n$ по рассматриваемой сетке. Свободный член в правой части является искомым числом для замкнутых маршрутов (см. п.3.4).
137. C_{2n}^n (см. указание к предыдущей задаче).
183. а) 14; б) 18; в) 9. 184. а) 43; б) 61. 185. а) 37; б) 100; в) 158.

Литература

1. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов. — М.: Мир, 1979.
2. Берж К. Теория графов и ее применения. — М.: ИЛ, 1962.
3. Виленкин Н.Я. Комбинаторика. - М.: Наука, 1969.
4. Виноградов И.М. Основы теории чисел. - М.: Наука, 1981.
5. Гаврилов Г.П., Сапоженко А.А. Сборник задач по дискретной математике. - М.: Наука, 1977.
6. Грин Д., Кнут Д. Математические методы анализа алгоритмов. — М.: Мир, 1987.
7. Гроссман И., Магнус В. Группы и их графы. - М.: Мир, 1971.
8. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. — М.: Мир, 1981.
9. Иванов Б.Н. Подсчет и оценивание. Алгоритмы на графах: Метод указания для студентов. — Владивосток, 1991.
10. Кнут Д. Искусство программирования для ЭВМ. Сортировка и поиск. Т.3. - М.: Мир, 1978.
11. Комбинаторный анализ. Задачи и упражнения. Учебное пособие / Под ред. Рыбникова К.А.— М.: Наука, 1980.
12. Кофман А. Введение в прикладную комбинаторику. — М.: Наука, 1975.
13. Кристофидес М. Теория графов. - М.: Мир, 1978.
14. Курош А.Г. Лекции по общей алгебре. - М.: Наука, 1973.
15. Нефедов В.Н., Осипова В.А. Курс дискретной математики. — М.: МАИ, 1992.
16. Оре О. Теория графов. - М.: Наука, 1980.
17. Препарата Ф., Шеймос М. Вычислительная геометрия. — М.: Мир, 1989.
18. Райзер Дж. Комбинаторная математика. — М.: Мир, 1966.
19. Рейнголд Э., Нивергельт Ю., Део Н. Комбинаторные алгоритмы. Теория и практика. - М.: Мир, 1980.
20. Риордан Дж. Введение в комбинаторный анализ. — М.: ИЛ, 1963.
21. Рыбников К.А. Введение в комбинаторный анализ. — М.: МГУ, 1972.
22. Уилсон Р.Дж. Введение в теорию графов. — М.: Мир, 1977.
23. Форд Л.Р., Фалкерсон Д.Р. Потоки в сетях. - М.: Мир, 1966.
24. Харрари Ф. Теория графов. - М.: Мир, 1973.
25. Харрари Ф., Палмер Э. Перечисление графов. — М.: Мир, 1977.
26. Холл М. Комбинаторика. — М.: Мир, 1970.
27. Холл П. Вычислительные структуры. Введение в нечисленное программирование. - М.: Мир, 1978.

Предметный указатель

- Алгоритмы
Евклида 227-228
поиска с возвращениями 66-68
порождения (=генерация)
 композиций 84
 перестановок 68-75
 случайных 89-90
 подмножеств 76
 разбиений 85-88
 размещений с повторениями 79
 сочетаний 80
- Алгоритмы на графах
выделение компонент связности 126-129
кратчайшие пути на графе 151-155
максимальное паросочетание 186-188
поиск блоков в глубину 182-184
поиск в глубину (общий) 117-123
поиск клик 160-171
поиск эйлеровой цепи 131-136
потoki в сетях 156-159
остовное дерево 137-138
 ближайшего соседа 145-149
 жадный 139-144
фундаментальные циклы 172-176
чередующиеся цепи 185-188
- Бернсаанда лемма 216
Вильсона теорема 237
Включение и исключение
 правило (принцип) 56
Выпуклая оболочка многоугольника (задача) 81
Вычеты 233
 полная система 233
 приведенная система 234
Гамильтонов цикл 113
Граф НО
 блоки 180-184
 вершины
 висячие 112
 изолированные 112
 смежные 112
 двудольный 185
 двусвязные 183
 диаметр, радиус, центры 196
 дополнение 112
 изоморфный 111
 клики 160
 компоненты связности 125
 листья 177-179
 маршрут 113
 матрица
 весов 114
 инцидентности 115
 смежности 114
 мост (разделяющее ребро) 178
 мультиграф 111
 остовное дерево 137
 паросочетания 186
 чередующиеся цепи 186-187
 пути на графе 151-155
 плоский (=планарный) 112
 помеченный 113
 связный 125
 список ребер 116
 структура смежности 117
 ориентированный 111
 петля 110
 подграф 111
 полный 112
 простой 112
 псевдограф 111
 связный 113
 сильно 113
 слабо 113
 транспортная сеть 156-159
 фундаментальные циклы 172-176
 хроматический 194
 цепь ИЗ
 простая 113
 замкнутая 113
 гамильтонова 113
 эйлерова 130
 цикл 113
 простой 113
 замкнутый 113
 гамильтонов 113
 эйлеров 130
 эйлеров 130
- Группа 197
 абелевы 203
 вычетов 233-234
 гомоморфизм 198
 образ 199
 ядро 199
 изоморфизм 199
 теорема Кэли 212

- индекс 200
- коммутативная 197
- подстановок 208
- примарная 204
- прямое произведение 203
 - циклические подгруппы 206
- Силовая 207
- симметрическая 208
- смежные классы 199–200
- фактор 201–202
- циклическая 200–201
- цикловой индекс 217
- Действие групп на множестве 212–216
- Декремент подстановки 209–210
- Дерево (граф) 114
 - бинарное (двоичное) 31
 - корневое 31
 - остовное 137–144
 - поддерево 32
 - представление 31
 - на связной памяти 32
 - на смежной памяти 32
 - бинарное (двоичное) 33
 - регулярное 34
 - сравнений 104
- Доминирующее множество 160–161
 - минимальное 161
 - число доминирования 161
- Доска запрещенных позиций 60
- Задача о назначениях 190–193
- Инверсии перестановки 20
- Индекс подгруппы 200
- Клики (в графе) 160
- Коллизии 108
- Компоненты связности графа 125–126
- Корень дерева 32
- Коэффициенты полиномиальные 14
- Куб n -мерный (задача) 40
- Кэли теорема 212
- Лагранжа теорема 200
- Лес (=множество деревьев) 31, 114
- Линейный порядок 91
- Мёбиуса функция 238
- Многочлен
 - ладейный 60–62
 - попаданий 63
- Множество
 - весов элементов 53
 - дополнение 8
 - объединение 8
 - пересечение 8
 - представление 37
 - смежное 37–38
 - связанное 37–38
 - характеристический вектор 38
- прямое произведение 8
- пустое 8
- разность 8
- универсальное 8
- Мультимножество 14
- Наибольший общий делитель 227
- Наименьшее общее кратное 228
- Независимое множество вершин, ребер 162
- Независимые циклы подстановок 206
- Нормальный делитель 201
- Обобщенное правило произведения 53
- Образующий элемент группы 200
- Орграф (=ориентированный граф) 111
- Отношение эквивалентности 124
- Паросочетание максимальное 186–188
- Петля (в графе) ПО
- Перестановки 11
 - инверсии 20
 - обратные 21
 - порождение 68–73
 - с повторениями 13
- Подгруппа 198
 - индекс 200
 - нормальный делитель 201
 - циклическая 200
- Подстановки 208
 - группа симметрическая 208
 - декремент 209
 - транспозиции 210–211
 - цикловая структура 223–226
 - четность 209–210
- Поиск данных 91
 - закон Зипфа 103
 - логарифмический 104–106
 - последовательный 102–103
- Поиск с возвращениями (алгоритм) 66
- Пойа теория перечисления 218–223
- Порядок
 - элемента (в группе) 200
 - линейный 91
- Потоки в сетях (в графе) 156–160
- Правило
 - суммы 8, 9, 53
 - прямого произведения 8, 9, 53
- Представление последовательности 24
 - связанное 26
 - смежное 24
 - характеристический вектор 25

- Принцип включения и исключения 56
 Производящая функция 39
 операции 39
 дополнительные суммы 42
 изменение масштаба 43
 линейные 41
 сдвиг влево, вправо 41–42
 свертка 44
 частичные суммы 42
 Простые числа 228
 решето Эратосфена 229
 Прямая адресация 106–109
 Разбиение множества
 упорядоченное 15
 неупорядоченное 15
 Разделяющая вершина 181–182
 Размещение
 с повторениями 9
 без повторений 10
 Расстановка 9–12
 Ребро (в графе) НО
 Рекуррентные линейные соотношения
 49
 неоднородное 51
 Система различных представителей
 189
 двудольные графы 189
 теорема Холла 190
 Смежные вершины 112
 Смежные левые (правые) классы 199
 Списки
 связанные 26–27
 циклически связанные 27
 Сортировка 91
 внешняя 91–92
 внутренняя 91–92
 всплытием Флойда 95–100
 вставками 92
 отрезков (задача) 100–101
 перечислением 94
 прямой адресации 106–109
 пузырьковая 93
 полная 94
 сложность 91
 Сочетания 11
 с повторениями 12
 Стабилизатор (группа) 214
 Сумма (задачи)
 квадратов 47
 счастливая 77
 Теорема
 Вильсона 237
 включения и исключения 56
 Кэли 212
 Лагранжа 200
 о двудольных графах 185
 о максимальном паросочетании 188
 о максимальном потоке и минимальном разрезе
 Силова 207
 Ферма 237
 Форда и Фалкersona 158
 Холла и система различных представителей 189–190
 Эйлера
 о графах 130
 о числах 237
 Теория чисел 227–239
 Точки сочленения 181
 Транспозиции подстановки 210–212
 Транспортная сеть 156
 поток 156
 максимальный 158–160
 насыщенный 158
 пропускная способность 156
 разрез 157
 Факторгруппа 201
 Ферма теорема 237
 Формула
 бинома Ньютона 19
 обращения Мёбиуса 238–239
 полиномиальная 18
 Фундаментальное множество циклов 172–177
 Функция
 Мёбиуса (свойства) 238–239
 производящая 39
 Эйлера (свойства) 234–235
 Характеристический
 вектор последовательности 25
 полином (уравнение) 50
 Хроматические графы 194–195
 Циклическая группа 200–201
 Цикловой индекс группы 217
 Цикловая структура групп
 подстановок 224–226
 Эйлеров граф 130–136
 Эйлерова теорема о числах 237
 Эйлерова функция (свойства) 234–236
 Эратосфеново решето 228–231
 простые числа 228