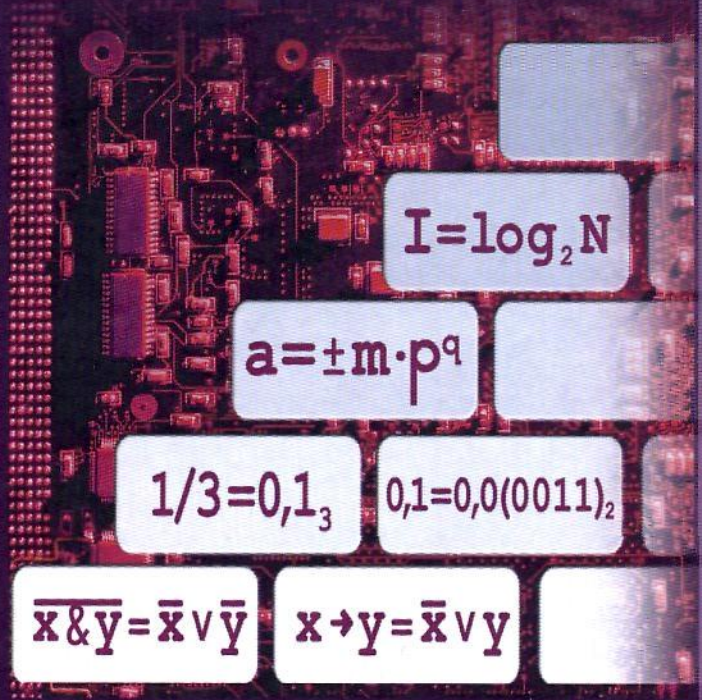
Е. В. Андреева Л. Л. Босова И. Н. Фалина

ЭЛЕКТИВНЫЙ КУРС

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие



БИНОМ

ЭЛЕКТИВНЫЙ КУРС

МАТЕМАТИКА

Е. В. Андреева Л. Л. Босова И. Н. Фалина

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ

Учебное пособие

Издание подготовлено при содействии
НФПК – Национального фонда подготовки кадров

Москва
БИНОМ. Лаборатория знаний
2005

УДК 004.9
ББК 32.97
А65

Андреева Е. В.

А65 Математические основы информатики. Элективный курс: Учебное пособие / Е. В. Андреева, Л. Л. Босова, И. Н. Фалина — М.: БИНОМ. Лаборатория знаний, 2005 — 328 с.: ил.

ISBN 5-94774-139-3

Учебное пособие входит в УМК для старших классов наряду с методическим пособием и хрестоматией. Материал раскрывает взаимосвязь математики и информатики, показывает, как развитие одной из этих научных областей стимулировало развитие другой. Дается углубленное представление о математическом аппарате, используемом в информатике, показывается, как теоретические результаты, полученные в математике, послужили источником новых идей и результатов в теории алгоритмов, программировании и в других разделах информатики.

Для учащихся старших классов информационно-технологического, физико-математического и естественно-научного профилей, желающих расширить свои теоретические представления о математике в информатике и информатике в математике.

УДК 004.9
ББК 32.97

Учебное издание

Андреева Елена Владимировна
Босова Людмила Леонидовна
Фалина Ирина Николаевна

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ИНФОРМАТИКИ
Элективный курс
Учебное пособие

Ведущий редактор *О. Полежаева*. Художник *Ф. Инфантэ*
Художественный редактор *О. Лапко*. Компьютерная верстка *В. Носенко*

Подписано в печать 03.10.05. Формат 60x90 1/16. Бумага офсетная.
Печать офсетная. Усл. печ. л. 20,5. Тираж 5000 экз. Заказ 3931

Издательство «БИНОМ. Лаборатория знаний». Телефон (095)955-0398.
E-mail: Lbz@aha.ru http:\\www.Lbz.ru

Отпечатано с готовых диапозитивов
в полиграфической фирме «Полиграфист».
160001, г. Вологда, ул. Челюскинцев, 3.

ISBN 5-94774-139-3

Оглавление

От авторов

Дорогие старшеклассники!

Задумывались ли вы, почему в современных компьютерах используется двоичная система счисления и можно ли заменить ее какой-либо другой?

Знаете ли вы, что такое машина Тьюринга и почему знакомству с ней придают такое важное значение в теории алгоритмов?

Вы, конечно, знаете, что для записи в компьютере графической информации используется растровое и векторное представление. А можно ли любую фотографию сохранить в компьютере так, чтобы ее цифровое представление было абсолютно идентичным оригиналу?

А из каких элементов построен компьютер, ведь известно, что компьютер обрабатывает только двоичные данные, но при этом является универсальным исполнителем?

До какого уровня можно улучшать алгоритмы, например, сортировки, чтобы они работали как можно быстрее?

На эти и многие другие вопросы вы найдете ответы в данной книге. Каждый ответ на подобный вопрос — это результат гениальной догадки и длительной работы ученых по разработке математической теории, на основе которой удалось обосновать выдвинутое предположение.

Наша книга — новый профильно-ориентированный курс «Математические основы информатики», который поможет вам продолжить образование в области математики, информатики и информационных технологий.

Сегодня, в начале XXI века, человечество входит в новую цивилизацию — цивилизацию, связанную с проникновением компьютеров во все сферы жизнедеятельности человека. Эту цивилизацию называют информационной, виртуальной, компьютерной...

Вы будете жить в новой цивилизации и должны научиться жить в ней, не теряя себя, пользоваться ее благами, но не становиться зависимыми от них. Мы убеждены, что человек может сохранить себя как *homo sapi-*

ens, только изучая основы фундаментальных знаний о мире, который его окружает. Особое значение в современных условиях приобретает такая учебная дисциплина, как информатика. В данной книге мы хотим показать, как математический аппарат используется в информатике, какие достижения математики повлияли на становление и развитие информатики с одной стороны, а с другой стороны, какие задачи информатики дали толчок к появлению новых идей и методов в математике. Это взаимовлияние двух наук продолжается до сих пор.

Мы хотели вам показать, как сложно выдвигать новые гипотезы, как часто ученые шли десятилетиями и даже столетиями к тем результатам, которые для вас сегодня привычны и очевидны.

В подготовке этой книги участвовал большой коллектив. Многие преподаватели СУНЦ МГУ (физико-математической школы–интерната им. А. Н. Колмогорова) использовали предварительные варианты книги на своих занятиях со школьниками и предложили различные улучшения. В написании книги большую помощь оказали наши коллеги: В. В. Усатюк (глава 2), Е. В. Щепин (глава 5), Ю. Е. Егоров (глава 6). Мы выражаем им глубокую благодарность. Мы благодарим за внимательное прочтение и научные консультации преподавателей факультета ВМиК МГУ им. М. В. Ломоносова В. Б. Алексеева (главы 3 и 4) и А. И. Фалина (глава 6). И, конечно же, мы благодарны самой многочисленной группе наших соавторов — ученикам СУНЦ МГУ, которые слушали лекции по материалам глав книги, самыми первыми решали все наши задачи и спрашивали, спрашивали, спрашивали... еще во время написания книги.

Книга, на наш взгляд, будет интересна и полезна тем, кто интересуется математикой, информатикой, физикой. Мы надеемся, что она поможет вам в выборе будущей профессиональной деятельности. Однако читать и изучать ее будет непросто, хотя мы максимально пытались структурировать материал, включили много иллюстраций, примеров. Не страшно, если вы не сможете разобраться в чем-то при первом чтении, возможно, к содержанию некоторых параграфов вам придется вернуться позже. Но, как говорится, дорогу осилит идущий.

Авторы

Как работать с книгой

Книга «Математические основы информатики» состоит из 6 глав, которые, вообще говоря, можно читать и изучать в любом порядке. Материал некоторых глав взаимосвязан, и в тексте есть соответствующие ссылки.

Главы состоят из параграфов, после каждого параграфа есть вопросы и задания для самостоятельной работы.

В тексте параграфов вам будут встречаться вопросы и задания, ответы на которые даны там же. Все эти вопросы не очень простые, но не спешите читать ответы и решения, попробуйте сначала самостоятельно найти ответы к поставленным проблемам.

В тексте много рисунков и таблиц, как правило, они содержат обобщающий материал в графическом виде, каждый раз старайтесь понять, почему приведен именно этот рисунок и именно в таком виде. Такая работа с иллюстративным материалом поможет вам лучше понять излагаемый материал.

Наиболее важный материал мы выделили навигационным знаком. Этот знак поможет вам быстро находить наиболее существенные факты, облегчит работу с книгой.

Текст книги непростой, некоторые разделы или параграфы, скорее всего, придется читать несколько раз, пусть вас это не смущает, именно так изучается серьезная профессиональная литература.

В тексте *курсивом* выделены вводимые термины и понятия, все они внесены в предметный указатель, который находится в конце книги. Обязательно используйте его для поиска нужных терминов и определений, это облегчит вам работу с книгой.

Успехов вам, дорогие ребята, в изучении книги!

Глава 1

Системы счисления

Мысль выражать все числа немногими знаками, придавая им, кроме значения по форме, еще значение по месту, настолько проста, что именно из-за этой простоты трудно понять, насколько она удивительна. Как нелегко было прийти к этому методу, мы видим на примере величайших гениев греческой учености Архимеда и Аполлония, от которых эта мысль осталась скрытой.

П. С. Лаплас

Владея развитой компьютерной теорией, компьютерные специалисты иногда забывают о той роли, которую сыграли системы счисления в истории компьютеров.

А. П. Стахов

Первые счетные приборы (абаки, счеты), прообразы современных компьютеров, начали создаваться задолго до возникновения и алгебры логики, и теории алгоритмов. И определяющую роль в их конструкции играли выбранные для них системы счисления.

Первые механические счетные машины (суммирующая машина Блеза Паскаля — 1642 г., счетная машина Вильгельма Лейбница — 1673 г., аналитическая машина Чарльза Бэббиджа — 1848 г.) были разработаны на основе десятичной системы счисления. Для реализации десяти устойчивых состояний использовались сложные системы зубчатых колес. Эти механические машины были очень громоздки, занимали много места. Так, если бы проект Аналитической машины Бэббиджа, которая явилась механическим прототипом появившихся спустя столетие ЭВМ, был реализован, то по размерам машина сравнялась бы с локомотивом, и чтобы привести в движение ее устройства, понадобился бы паровой двигатель. Причинами этого были механический принцип построения устройств и использование десятичной системы счисления, затрудняющей создание простой элементной базы.

Конрад Цузе (1910–1985)

Через 63 года после смерти Бэббиджа немецкий студент Конрад Цузе начал работу по созданию машины, основанной на принципах действия Аналитической машины Бэббиджа. В 1937 г. машина *Z*1 была готова. Работала она на основе двоичной системы счисления и была чисто механической, как у Бэббиджа. Но использование двоичной системы сотворило чудо — машина занимала всего два квадратных метра на столе в квартире изобретателя.

В современных компьютерах вся информация также хранится в виде последовательностей нулей и единиц. Однако двоичная система счисления в чистом виде обладает рядом принципиальных недостатков, которые становятся критичными в век бурного развития компьютерной техники. Главными из этих недостатков являются проблема представления отрицательных чисел и «нулевая избыточность» (т. е. отсутствие избыточности, из чего вытекает невозможность определения, произошло ли искажение информации при ее передаче — см. § 1.8). Практическая потребность в решении этих вопросов вызывает сегодня по-

вышенный интерес к способам представления информации в компьютере и новым компьютерным арифметикам. Так, например, родоначальник теории информации Джон фон Нейман доказал теорему о том, что троичная система счисления позволяет наиболее эффективно среди всех основных позиционных систем счисления «сворачивать» информацию о вещественном числе.

Какие же системы счисления рассматриваются математиками и инженерами в качестве «компьютерных»? Какими свойствами должна обладать система счисления, при помощи которой будет кодироваться информация в компьютерных системах? Мы попытаемся ответить на эти и другие вопросы, связанные с системами счисления. Наша задача — рассмотреть принципы построения позиционных систем счисления, познакомиться с неклассическими (нетрадиционными) позиционными системами счисления, с системами счисления, используемыми в компьютерах.

Начнем с повторения базовых определений.

§ 1.1. Позиционные системы счисления. Основные определения

Определение 1. *Система счисления* или *нумерация* — это способ записи (обозначения) чисел.

Определение 2. Символы, при помощи которых записываются числа, называются *цифрами*, а их совокупность — *алфавитом* системы счисления. Количество цифр, составляющих алфавит, называется его *размерностью*.

Определение 3. Система счисления называется *позиционной*, если количественный эквивалент цифры зависит от ее положения в записи числа.

В привычной нам десятичной системе значение числа образуется следующим образом: значения цифр умножаются на «веса» соответствующих разрядов и все полученные значения складываются. Например, $5047 = 5 \cdot 1000 + 0 \cdot 100 + 4 \cdot 10 + 7 \cdot 1$. Такой способ образования значения числа называется *аддитивно-мультипликативным*.

Определение 4. Последовательность чисел, каждое из которых задает «вес» соответствующего разряда, называется *базисом* позиционной системы счисления.

Основное достоинство практически любой позиционной системы счисления — возможность записи произвольного числа при помощи ограниченного количества символов.

Определение 5. *Позиционную систему счисления* называют *традиционной*, если ее базис образуют члены геометрической прогрессии, а значения цифр есть целые неотрицательные числа.

Так, базисы десятичной, двоичной и восьмеричной систем счисления образуют геометрические прогрессии со знаменателями 10, 2 и 8 соответственно. В общем виде базис традиционной системы счисления можно записать так:

$$\ldots P^{-3}, P^{-2}, P^{-1}, 1, P, P^2, P^3, \ldots, P^n, \ldots$$

Определение 6. Знаменатель P геометрической прогрессии, члены которой образуют базис традиционной системы счисления, называется *основанием* этой системы счисления. Традиционные системы счисления с основанием P иначе называют *P-ичными*.

В P-ичных системах размерность алфавита равна основанию системы счисления.

Так, алфавит десятичной системы составляют цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Алфавитом произвольной системы счисления с основанием P служат числа 0, 1, ... , $P-1$, каждое из которых должно быть записано с помощью одного уникального символа, младшей цифрой всегда является 0.

В класс позиционных систем счисления входят также системы, в которых либо базис не является геометрической прогрессией, а цифры есть целые неотрицательные числа, либо базис является геометрической прогрессией, но цифры не являются целыми неотрицательными числами.

К первым можно отнести *факториальную* и *фибоначчиеву* системы счисления, ко вторым — *уравновешенные* системы счисления. Такие системы будем называть *нетрадиционными*. Алфавитом фибоначчиевой системы являются цифры 0 и 1, а ее базисом — последовательность чисел Фибоначчи 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 ... .

Леонардо Пизанский Фибоначчи (1170–1250) — итальянский математик. Благодаря его книге «Liber Abaci» Европа узнала индо-арабскую систему чисел, которая позднее вытеснила римские числа

Базисом факториальной системы счисления является последовательность 1!, 2!, ... , $n!$, ... В отношении алфавита этой системы можно сделать замечание: количество цифр, используемых в разряде, увеличивается с ростом номера разряда.

В общем случае, если система счисления устроена таким образом, что основание как таковое в ней отсутствует, а базис представляет собой последовательность чисел ..., P_0, P_1, ..., P_n, ..., то количество N_k цифр, используемых в k-м разряде, определяется так:

$$N_k = \begin{cases} \dfrac{P_{k+1}}{P_k}, & \text{если } P_{k+1} \vdots P_k; \\ \left[\dfrac{P_{k+1}}{P_k}\right] + 1, & \text{в противном случае.} \end{cases} \qquad (1.1)$$

Знак «$\vdots$» означает «делится нацело».

Пример 1. Приведем сводную таблицу, характеризующую некоторые позиционные системы счисления.

Система счисления	Основание	Размерность алфавита	Цифры
Двоичная	2	2	0, 1
Троичная	3	3	0, 1, 2
Восьмеричная	8	8	0, 1, 2, 3, 4, 5, 6, 7
Шестнадцатеричная	16	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
Факториальная	Нет	Увеличивается с ростом номера разряда	1-й разряд: 0, 1 2-й разряд: 0, 1, 2 3-й разряд: 0, 1, 2, 3 ...
Фибоначчиева	Нет	2	0, 1
Уравновешенная троичная (см. § 1.8)	3	3	$\bar{1}$, 0, 1

□

Основанием P-ичной системы счисления может быть любое натуральное число, большее единицы. Системой счисления с минимальным основанием является двоичная система, все числа в которой записываются с помощью 0 и 1.

Пример 2. Приведем запись некоторых десятичных чисел в различных нетрадиционных позиционных системах счисления.

Десятичная система счисления	Факториальная система счисления	Фибоначчиева система счисления	Уравновешенная троичная система счисления
10	120	10010	101
25	1001	1000101	$10\bar{1}1$
100	4020	1000010100	$11\bar{1}01$

□

Мы видим, что для описания системы счисления используются понятия *«базис»*, *«алфавит»*, *«основание»*.

Вопрос. *Какое множество понятий однозначно определяет позиционную систему счисления*:
1) {*базис, алфавит, основание*};
2) {*базис, алфавит*};
3) {*базис*}?

Ответ. Оказывается, что для однозначного определения позиционной системы счисления, у которой в качестве цифр используются натуральные числа и 0, необходимо и достаточно указать только ее базис: последовательность чисел ..., P_0, P_1, ..., P_n, ... Все остальные компоненты системы являются производными от базиса. Покажем это.

Формулировка необходимого условия имеет вид: если задана позиционная система счисления, то, следовательно, задан базис. Это утверждение очевидно следует из определений 3 и 4.

Формулировка достаточного условия имеет вид: если задан базис, то задана позиционная система счисления.

Рассмотрим отдельно два случая: 1) базис является геометрической прогрессией, 2) базис не является геометрической прогрессией.

В первом случае отношение $\frac{P_{k+1}}{P_k}$ для любого k постоянно и равно P — основанию системы счисления. Количество цифр в алфавите также равно P, так как максимальное число единиц, которое можно записать в любом разряде, равно $P-1$, а минимальным числом является 0.

Во втором случае основание в системе отсутствует, а количество цифр алфавита в каждом разряде определяется по базису согласно формуле (1.1).

Если же в качестве цифр в системе счисления используются числа, отличные от целых неотрицательных, то для определения системы счисления необходимо еще описать и ее алфавит. □

Вопрос. *Какая последовательность чисел может быть использована в качестве базиса позиционной системы счисления?*

Ответ. Последовательность чисел может являться базисом позиционной системы счисления только тогда, когда в соответствующей этому базису системе может быть представлено любое число (если система предназначена только для нумерации целых чисел, то любое целое число).

Для представления целых чисел достаточно взять любую бесконечную монотонно возрастающую числовую последовательность (P_0, P_1, ..., P_i, ...), начинающуюся с единицы ($P_0 = 1$). В качестве цифр k-го разряда этой системы следует использовать символы, обозначающие числа 0, 1, 2, ..., N_k-1, где N_k вычисляется по формуле (1.1). Числа 0 и 1 в любой из таких систем представляются соответствующими цифрами. Пусть числа 2, ..., a представимы в системе с описанным базисом. Покажем, что тогда и число $a + 1$ также представимо.

Если младшая цифра числа a меньше, чем $N_0 - 1$, то в представлении числа $a + 1$ все старшие цифры совпадают с цифрами a, а младшая цифра на единицу больше, чем соответствующая цифра числа a. Так как $P_0 = 1$, то, согласно аддитивно-мультипликативному принципу построения подобных систем, числа a и $a + 1$ в системе с данным базисом в этом случае действительно отличаются на единицу. Пусть теперь в представлении числа a в разрядах 0, 1, 2, ..., i стоят максимально допустимые

цифры этих разрядов, а цифра в $(i + 1)$-м разряде меньше, чем $N_{i+1} - 1$. Увеличим значение числа, состоящего из цифр, стоящих в разрядах 0, 1, 2, ..., i числа a, на единицу. Получившееся число можно оценить снизу и сверху так:

$$P_{i+1} \le 1 + N_0 - 1 + (N_1 - 1)P_1 + (N_2 - 1)P_2 + \ldots + (N_i - 1)P_i < \\ < P_0 + P_1 + P_2 + \ldots + P_{i+1}.$$

Здесь при оценке снизу учтено, что согласно формуле (1.1) $N_0 \ge P_1$, $N_1 \cdot P_1 \ge P_2$, ..., $N_i \cdot P_i \ge P_{i+1}$, а при оценке сверху используются следующие из той же формулы неравенства $N_k - 1 < \frac{P_{k+1}}{P_k}$. Таким образом, для числа $a + 1$ оказывается возможным следующее представление: цифра в $(i + 1)$-м разряде числа a увеличивается на единицу, а число

$$0 \le 1 + N_0 - 1 + (N_1 - 1)P_1 + (N_2 - 1)P_2 + \ldots + (N_i - 1)P_i - P_{i+1} < \\ < P_0 + P_1 + P_2 + \ldots + P_i$$

представляется с использованием элементов базиса P_0, P_1, ..., P_i. Заметим, что если для любых соседних элементов базиса справедливо точное равенство $N_k = \frac{P_{k+1}}{P_k}$, то все цифры в представлении $a + 1$ в разрядах 0, 1, ..., i равны 0. В этом случае можно говорить и о единственности представления чисел в позиционной системе с соответствующим базисом. Соответствующая теорема доказана в следующем параграфе. □

Вопрос. *Какие символы могут быть использованы в качестве цифр системы счисления?*

Ответ. В качестве цифр систем счисления могут быть использованы любые символы, это наглядно демонстрируют нам ученые, занимающиеся историей математики: вавилоняне использовали клиновидные цифры (у них не было бумаги, и «писали» они на мягких глиняных дощечках); китайцы использовали иероглифы; мы используем арабские цифры. Однако в математике придерживаются следующих договоренностей в отношении вида используемых цифр.

Если основание системы счисления P меньше 10, то для символьного представления цифр в ней, как правило, используются первые P десятичных цифр (от 0 до $P-1$). Например, в пятеричной системе счисления будут использоваться пять цифр: 0, 1, 2, 3, 4.

Для $10 < P < 37$ в качестве первых десяти цифр также обычно используют их десятичное представление, а для остальных цифр — буквы латинского алфавита.

Для систем счисления с основаниями, большими 36, единых правил для формы записи цифр не существует. В дальнейшем, если при описании произвольной P-ичной системы счисления вид ее цифр указан не будет, то мы будем считать, что первые десять цифр совпадают с десятичными, а следующие 26 — с латинскими буквами. Остальные цифры будем записывать в виде их десятичных значений, заключенных в квадратные скобки. Так, [50] в системах счисления с основаниями, бóльшими 50, будет обозначать 51-ю по счету от нуля цифру. Максимальную цифру в произвольной P-ичной системе счисления мы будем обозначать $[P-1]$. □

Вопросы и задания

1. Сформулируйте определение аддитивно-мультипликативной системы счисления.
2. Сформулируйте правила, по которым вычисляется значение числа в римской системе счисления. Является ли она аддитивно-мультипликативной?
3. Сколько цифр нужно для записи чисел в двенадцатеричной системе счисления?
4. Верно ли записаны числа в семеричной системе счисления: 2360_7, 35721_7, 608512_7?
5. Предложите собственную классификацию систем счисления.
6. Придумайте и выпишите алфавит для пятидесятеричной системы счисления.
7. Опишите позиционную систему счисления, основанную на разложении числа по степеням простых чисел. Является ли она аддитивно-мультипликативной?
8. Докажите, что для P-ичных систем счисления минимальным основанием является число 2.

§ 1.2. Единственность представления чисел в P-ичных системах счисления

В примере 2 (§ 1.1) были приведены представления чисел 10, 25 и 100 в системах счисления, отличных от десятичной.

Вопрос. *Можно ли эти числа записать в указанных системах еще и другим способом или это представление единственно?*

Ответ. Оказывается, что любое десятичное число можно представить в любой позиционной системе счисления, а для целых чисел в большинстве систем это можно сделать единственным способом. Докажем это утверждение для натуральных чисел в P-ичных системах счисления. □

Теорема 1. Пусть P — произвольное натуральное число, большее единицы. Существует и единственно представление любого натурального числа X в виде степенного ряда

$$X = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0, \qquad (1.2)$$

где $0 \le a_i < P$, $0 \le i \le n$, $a_n \ne 0$.

Доказательство

Существование. Доказательство основано на методе построения, т. е. для произвольного натурального числа мы просто построим представление вида (1.2).

Так как числа P^0, P^1, P^2, P^3, ... образуют монотонно возрастающую числовую последовательность, то существует такое натуральное число n, что

$$P^n \le X < P^{n+1}. \qquad (1.3)$$

Разделим интервал $[P^n; P^{n+1})$ на $P - 1$ равную часть, тогда границами полученных интервалов окажутся числа $x_1 = 1 \cdot P^n$, $x_2 = 2 \cdot P^n$, ..., $x_P = P \cdot P^n = P^{n+1}$. Длина каждого промежутка $[x_k; x_{k+1})$, где $k = 1, \ldots, P - 1$, равна P^n.

Из (1.3) и проведенного построения следует, что число X попадет в один из интервалов $[x_k; x_{k+1})$, т. е. существует такое натуральное k, $1 \le k < P$, что

$$kP^n \le X < (k+1)P^n. \qquad (1.3a)$$

Положим $a_n = k < P$. Тогда $a_n \ne 0$.

Обозначим разницу между числом X и левой границей x_k соответствующего интервала как $Y = X - a_n P^n$. $0 \le Y < P^n$ по построению.

Если $Y = 0$, то построение закончено, в противном случае, опять сравнивая уже величину Y с членами возрастающей последовательности 1, P, P^2, P^3, ... P^n, найдем целое число m такое, что $P^m \le Y < P^{m+1}$, $0 \le m$.

Для номеров $m + 1 \le i \le n - 1$ положим $a_i = 0$ (таких номеров может и не оказаться, если $m + 1 = n$) и вычислим a_m. Для этого, как и ранее, разделим интервал $[P^m; P^{m+1})$ на $P - 1$ равные части длиной P^m и определим, в какую из них попадает число Y, т. е. среди целых значений h: $0 \le h < P$ найдем такое, что

$$hP^m \le Y(h + 1)P^m. \qquad (1.3b)$$

Положим $a_m = h < P$ и обозначим $Z = Y - a_m P^m$. И так далее.

Процесс обязательно завершится, так как на каждом шаге мы сравниваем оставшееся число все с меньшим конечным количеством различных неотрицательных степеней числа P. А как только результат очередного вычитания окажется меньше, чем P, мы положим a_0 равным ему, и построение закончится.

В результате получим, что

$$X = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0,$$

где $0 \le a_i < P$, $0 \le i \le n$, $a_n \ne 0$.

Единственность. Для доказательства воспользуемся методом от противного.

Предположим, что некоторое натуральное число имеет два различных представления вида (1.2):

$$X_1 = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0, \qquad (1.4a)$$

и

$$X_2 = b_m P^m + b_{m-1} P^{n-1} + \ldots + b_1 P + b_0. \qquad (1.4b)$$

Покажем, что если $m > n$, то $X_2 > X_1$. Для доказательства оценим X_2 снизу наименьшим возможным числом ($b_m = 1$; $b_{m-1} = \ldots = b_1 = b_0 = 0$):

$$X_2 \ge 1 \cdot P^m + 0 \cdot P^{m-1} + \ldots + 0 \cdot P + 0 = P^m.$$

Оценим X_1 сверху наибольшим возможным числом ($a_n = \ldots = a_1 = a_0 = P-1$):

$$X_1 \le (P-1)P^n + (P-1)P^{n-1} + \ldots + (P-1)P + (P-1) = \\ = P^{n+1} - 1 < P^{n+1}.$$

Здесь для вычисления суммы использовалась формула суммы членов конечной геометрической прогрессии.

Так как по предположению $m \ge n+1$, то $X_1 < P^{n+1} \le X_2$, т. е. $X_1 < X_2$. Следовательно, если $X_1 = X_2$, то $n = m$.

Покажем, что $a_n = b_n$. Опять воспользуемся методом от противного. Пусть $a_n > b_n$ (например, $a_n = b_n + 1$). Оценим разность $X_1 - X_2 = P^n - (a_{n-1} - b_{n-1})P^{n-1} - \ldots - (a_0 - b_0)$. Заменим все a_i ($0 \le i < n$) на их максимально возможные значения ($P-1$), а все b_i ($0 \le i < n$) — на минимально возможные значения (нули):

$$X_1 - X_2 \ge P^n - ((P-1)P^{n-1} + (P-1)P^{n-2} + \\ + \ldots + (P-1)) = 1.$$

Это противоречит тому, что $X_1 = X_2$, следовательно, $a_n = b_n$.

Пусть существует такое k, что $a_i = b_i$ при $k+1 \le i \le n = m$, но $a_k \ne b_k$. Сравним числа

$$Y_1 = X_1 - a_n P^n - \ldots - a_{k+1}P^{k+1}$$

и

$$Y_2 = X_2 - b_n P^n - \ldots - b_{k+1}P^{k+1}.$$

После преобразования получаем:

$$Y_1 = a_k P^k + \ldots + a_1 P + a_0,$$
$$Y_2 = b_k P^k + \ldots + b_1 P + b_0,$$

где $a_k \ne b_k$. Повторив проведенные ранее рассуждения, получим, что $Y_1 \ne Y_2$, и, следовательно, $X_1 \ne X_2$.

Получили противоречие с исходным предположением о равенстве представлений (1.4, *а*) и (1.4, *b*). Следовательно, $a_i = b_i$ при $0 \le i \le n = m$, т. е. представление вида (1.2) для любого натурального числа единственно. *Теорема доказана.*

На основании теоремы 1 можно утверждать, что любое натуральное число можно записать в какой угодно P-ичной системе счисления, причем единственным образом.

Пример 3. Построим представление десятичного числа $X = 3056$ в виде степенных рядов при различных значениях P.

1) $P = 10$.

Очевидно, что $10^3 \le 3056 < 10^4$, следовательно, в представлении (1.2) $n = 3$.

Разделив интервал $[10^3; 10^4)$ на 9 равных частей, получим, что $3 \cdot 10^3 \le 3056 < 4 \cdot 10^3$, следовательно, $a_3 = 3$.

$Y = 3056 - 3 \cdot 10^3 = 56$ и, так как $10 \le 56 < 10^2$, то $a_2 = 0$.

Далее получаем $5 \cdot 10 \le 56 < 6 \cdot 10$, следовательно, $a_1 = 5$.

Оставшееся число $Z = Y - 5 \cdot 10 = 56 - 50 = 6 < 10$, следовательно, $a_0 = 6$, и построение закончено.

В результате получаем: $3056 = 3 \cdot 10^3 + 5 \cdot 10 + 6$.

2) $P = 16$.

$16^2 \le 3056 < 16^3$, следовательно, в представлении (1.2) $n = 2$.

Разделив интервал $[16^2; 16^3)$ на 15 равных частей, получим, что $11 \cdot 16^2 \le 3056 < 12 \cdot 16^2$, следовательно, $a_2 = 11$.

$Y = 3056 - 11 \cdot 16^2 = 240$. Но $15 \cdot 16 \le 240 < 16^2$, следовательно, $a_1 = 15$.

И, так как $240 - 15 \cdot 16 = 0$, то построение окончено, а $a_0 = 0$.

В результате получаем: $3056 = 11 \cdot 16^2 + 15 \cdot 16$. □

Пример 4. Десятичное число 14 можно записать

в двоичной системе как	1110_2 $(14 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0)$;
в троичной системе как	112_3 $(14 = 1 \cdot 3^2 + 1 \cdot 3^1 + 2 \cdot 3^0)$;
в четверичной системе как	32_4 $(14 = 3 \cdot 4^1 + 2 \cdot 4^0)$;
в 14-ричной системе как	10_{14} $(14 = 1 \cdot 14^1 + 0 \cdot 14^0)$.

В системах счисления с основанием, бóльшим 14, данное число будет представлено одной цифрой (это будет буква латинского алфавита E или некий другой символ). □

В разделе математики «Теория чисел» доказывается, что и любую правильную дробь можно представить в

виде конечной или бесконечной суммы отрицательных степеней любого натурального числа $P > 1$. Например:

$$0{,}123 = 1\cdot10^{-1} + 2\cdot10^{-2} + 3\cdot10^{-3} = 0{,}1 + 0{,}02 + 0{,}003;$$

$$\frac{1}{6} = 0{,}1(6) = 1\cdot10^{-1} + 6\cdot10^{-2} + 6\cdot10^{-3} + \ldots = 0{,}1 + 0{,}06 + 0{,}006 + \ldots;$$

$$\pi - 3 = 0{,}1415\ldots = 1\cdot10^{-1} + 4\cdot10^{-2} + 1\cdot10^{-3} + 5\cdot10^{-4} + \ldots = 0{,}1 + 0{,}04 + 0{,}001 + 0{,}0005 + \ldots$$

Так как произвольное неотрицательное действительное число можно представить в виде суммы его целой и дробной частей (любая из этих частей может и отсутствовать), то полученные результаты можно обобщить.

Определение 7. В P-ичной системе счисления любое неотрицательное вещественное число можно записать в виде:

$$a = a_nP^n + a_{n-1}P^{n-1} + \ldots + a_1P + a_0 + a_{-1}P^{-1} + a_{-2}P^{-2} + \ldots = \sum_{i=-\infty}^{n} a_iP^i,\ 0 \le a_i < P,\ n \ge 0, \quad (1.5)$$

где $P > 1$ — *основание* позиционной системы счисления, a_i — *цифры* числа a в P-ичной системе счисления.

Отрицательные числа в P-ичных системах счисления представляются с помощью знака «минус» перед выражением вида (1.5) для модуля отрицательного числа. Далее мы будем рассматривать только положительные числа и их представление в P-ичных системах счисления.

Вопросы и задания

1. Постройте представление десятичного числа $X = 3056$ в виде степенного ряда при $P = 2$.
2. Покажите, что любое натуральное число может быть представлено в виде суммы различных неотрицательных степеней числа 2.
3. Во сколько раз увеличится число 325_6, если приписать к нему справа ноль?
4. Как изменится запись P-ичной дроби с нулевой целой частью, если ее разделить на P^2?

5. Выполните путем рассуждения следующие действия, не используя операцию деления:
$100000_P:1000_P$; $201_P:100_P$.

§ 1.3. Представление произвольных чисел в позиционных системах счисления

На основании теоремы 1 мы можем утверждать, что любое число может быть записано в виде суммы степеней числа P, где P — натуральное число, большее 1. Вместе с тем, если мы в качестве базиса позиционной системы счисления возьмем возрастающую последовательность степеней числа P и тем самым однозначно определим P-ичную систему счисления, то это разложение по степеням числа P будет являться представлением данного числа в P-ичной системе счисления.

1.3.1. Развернутая и свернутая формы записи

Договоримся представление числа в P-ичной системе счисления в виде (1.5) называть *развернутой формой* записи числа (эта форма в основном используется при решении задач).

Другим способом записи произвольного числа в позиционной системе счисления с основанием P является последовательное перечисление его значащих цифр, начиная со старшей, при этом целая часть отделяется от дробной запятой. То есть разложению вида (1.5) вещественного числа a по степеням P соответствует запись вида:

$$a = a_n \ldots a_1 a_0, a_{-1} \ldots a_{-k} \ldots \qquad (1.6)$$

Представление числа в P-ичной системе счисления в виде (1.6) называется *свернутой формой* записи числа.

Таким образом, натуральное число a в P-ичной системе счисления можно записать двумя равнозначными способами:

$$a = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0 = a_n a_{n-1} \ldots a_1 a_0. \qquad (1.7)$$

Правильную конечную P-ичную дробь b можно записать следующими способами:

$$b = b_{-1} P^{-1} + b_{-2} P^{-2} + \ldots + b_{-k} P^{-k} = 0, b_{-1} b_{-2} \ldots b_{-k}. \qquad (1.8)$$

При использовании развернутой формы для записи числа в P-ичной системе счисления основание P и его степени обычно записывают в десятичной системе, а цифры — в P-ичной. При использовании свернутой формы цифры также записывают в P-ичной системе, а основание P, записанное в десятичной системе, приписывают к числу в качестве его нижнего индекса. Исключение может составлять лишь десятичная система счисления, при записи чисел в которой индекс часто опускается.

1.3.2. Перечисление натуральных чисел

Вопрос. *Предположим, что мы работаем в 50-ричной системе счисления. Можно ли только по свернутой форме числа A, не производя никаких вычислений, определить, больше 50 или нет десятичный эквивалент числа A?*

Ответ. В любой P-ичной системе счисления натуральные числа, меньшие ее основания P, представляются с помощью одной цифры данной системы. Для чисел же, бóльших или равных P, требуются уже, по крайней мере, две цифры. Само число P в системе счисления с основанием P записывается в виде 10_P, что следует из развернутой формы записи числа P в P-ичной системе: $P = 1 \cdot P + 0$. Следовательно, если число A записано при помощи только одной цифры, то оно меньше 50, если число имеет вид 10_P, то оно равно 50, в остальных случаях число больше 50. □

Из ответа на предыдущий вопрос понятно, как в P-ичной системе счисления перечислять (выписывать в возрастающем порядке) числа, меньшие P. Само число P записывается в виде 10_P. Для перечисления чисел, больших P, воспользуемся следующим алгоритмом, описывающим, как по известной P-ичной форме записи натурального числа a_P получить запись следующего натурального числа $a_P + 1$.

Алгоритм перечисления натуральных чисел в P-ичных системах счисления

1. Если последняя (крайняя справа) цифра числа a_P меньше $P - 1$, то в следующем по порядку натуральном числе все цифры, кроме последней, будут совпадать с циф-

рами числа a_P, а последняя цифра числа $a_P + 1$ будет на единицу больше последней цифры числа a_P.

2. Если последняя цифра числа a_P равна $P - 1$, то последняя цифра числа $a_P + 1$ будет равна 0, а остальные цифры будут представлять число, состоящее из первых цифр числа a_P (начиная с крайней левой цифры и заканчивая предпоследней справа), увеличенное на единицу по правилам 1–2 данного алгоритма; если же первые цифры в записи a_P отсутствуют, то число $a_P + 1$ будет равно 10_P. △

Покажем, как перечислять натуральные числа в различных системах счисления.

Пример 5. В двоичной системе первые 16 чисел будут иметь следующий вид:

$1 = 1_2$;
$2 = 10_2$ (правило 2);
$3 = 11_2$ (правило 1);
$4 = 100_2$ (дважды примененное правило 2);
$5 = 101_2$ (правило 1);
$6 = 110_2$ (правила 2 и 1);
$7 = 111_2$ (правило 1);
$8 = 1000_2$ (трижды примененное правило 2);
$9 = 1001_2$ (правило 1);
$10 = 1010_2$ (правила 2 и 1);
$11 = 1011_2$ (правило 1);
$12 = 1100_2$ (дважды примененное правило 2, правило 1);
$13 = 1101_2$ (правило 1);
$14 = 1110_2$ (правила 2 и 1);
$15 = 1111_2$ (правило 1);
$16 = 10000_2$ (четырежды примененное правило 2). □

Пример 6. Приведем (без подробных комментариев) некоторые числа в 16-ричной системе счисления:

$10 = A_{16}$; $19 = 13_{16}$;
$11 = B_{16}$; $20 = 14_{16}$;
$12 = C_{16}$; ...
$13 = D_{16}$; $31 = 1F_{16}$;
$14 = E_{16}$; $32 = 20_{16}$;
$15 = F_{16}$; ...
$16 = 10_{16}$; $255 = FF_{16}$;
$17 = 11_{16}$; $256 = 100_{16}$.
$18 = 12_{16}$; □

1.3.3. Представление обыкновенных десятичных дробей в P-ичных системах счисления

В общем случае для представления десятичной дроби в P-ичной системе счисления надо воспользоваться специальными алгоритмами перевода. Однако для некоторых видов десятичных дробей мы можем указать их P-ичное представление, даже не зная алгоритмов перевода. Речь идет об *обыкновенных дробях*. Обыкновенные дроби записываются с помощью отношения числителя и знаменателя, наибольший общий делитель которых равен 1.

В десятичной системе счисления обыкновенная дробь будет точно представима *конечной* дробью, если существует такое натуральное число m, при умножении на которое знаменателя дроби можно получить некоторую натуральную степень числа 10. Если же такого числа не существует, то эту дробь можно представить только в виде бесконечной периодической дроби.

Вопрос. *Можно ли по виду десятичной обыкновенной дроби определить, представима ли она конечной дробью в P-ичной системе счисления?*

Ответ. Да, это возможно. Более того, для любой обыкновенной десятичной дроби, не являющейся конечной, можно найти систему счисления, в которой она будет представима конечной дробью. И наоборот, для любой обыкновенной десятичной дроби, являющейся конечной дробью, можно указать систему счисления, в которой она не будет представима конечной дробью. □

Пример 7. Известно, что правильную десятичную дробь $\frac{1}{3}$ нельзя записать в виде конечной десятичной дроби. Однако в троичной и 9-ричной системах счисления эта дробь будет записана в виде конечной P-ичной дроби.

В 3-ичной системе счисления: $\frac{1}{3} = 0{,}1_3$;

в 9-ричной системе счисления: $\frac{1}{3} = 0{,}3_9$.

□

Общее правило. Десятичная обыкновенная дробь будет точно представима конечной P-ичной дробью, если существует такое натуральное число m, при умножении на которое знаменателя дроби можно получить некоторую натуральную степень числа P. Если же такого числа не существует, то в P-ичной системе счисления дробь окажется бесконечной периодической.

Данный факт следует непосредственно из развернутой формы представления числа.

Кроме того, из развернутого представления дробной части числа следует, что в любой системе счисления с основанием P верны равенства

$$\frac{1}{P} = 0{,}1_P;\ \frac{1}{P^2} = 0{,}01_P;\ \ldots;\ \frac{1}{P^k} = 0{,}\underbrace{0\ldots0}_{k-1}1_P. \qquad (1.9)$$

Алгоритм записи обыкновенной десятичной дроби в виде конечной P-ичной дроби

Пусть для нашей дроби существует такое натуральное число m, что при умножении знаменателя на m получаем k-ю степень числа P.

1. Умножим числитель дроби на m.
2. Представим результат умножения в P-ичной системе счисления.
3. Дополним числитель, если потребуется, до k цифр нулями слева.
4. Полученное P-ичное число запишем после запятой. Оно является конечной P-ичной дробью для исходной обыкновенной. △

Пример 8. Запишем $\frac{5}{16}$ в двоичной системе. В знаменателе уже стоит четвертая степень двойки. Переведем числитель в двоичную систему ($5 = 101_2$) и дополним получившееся число до четырех цифр: 0101. В результате получим: $\frac{5}{16} = 0{,}0101_2$. □

Пример 9. Запишем $\frac{1}{18}$ в 6-ричной системе. Если знаменатель умножить на 2, то получим 6^2. Тогда умножим и числи-

тель на 2. Так как в знаменателе стоит вторая степень основания системы счисления, то после запятой мы должны записать 02. В результате получим: $\frac{1}{18} = 0{,}02_6$. □

Вопрос. *Можно ли описанный способ представления обыкновенных десятичных дробей в P-ичных системах счисления использовать для решения обратной задачи?*

Ответ. Так как все P-ичные системы равноправны, то описанный способ представления десятичных дробей в P-ичной системе счисления можно использовать и для решения обратной задачи: обыкновенную P-ичную дробь записать в десятичной системе счисления, не производя операции деления числителя на знаменатель. □

Пример 10. Запишем $0{,}011_2 = \frac{11_2}{1000_2}$ в десятичной системе.

Для того чтобы знаменатель, равный сейчас 2^3, оказался степенью десяти, его нужно домножить на 5^3. Таким образом, производя необходимые действия в десятичной системе счисления, получим:

$$\frac{11_2}{1000_2} = \frac{3}{8} = \frac{3 \cdot 5^3}{8 \cdot 5^3} = \frac{375}{1000} = 0{,}375.$$

□

Вопросы и задания

1. Запишите в развернутом виде числа 65_7; $10203{,}405_7$; $0{,}15A_{16}$; $1AF1H{,}A9_{20}$.
2. Какое из чисел больше: 5_{10} или 10_5; 1000_2 или 10_8?
3. Существуют ли системы счисления с основаниями P и Q, в которых $12_P > 21_Q$?
4. Для десятичного числа 371 найдите основание P системы счисления, в которой данное число будет представлено теми же цифрами, но записанными в обратном порядке, т. е. $371_{10} = 173_P$.
5. В каких системах счисления $5_P + 5_P \neq 10_P$?
6. В каких системах счисления $2_P + 2_P = 4_P$?
7. Во сколько раз увеличится число 32_4, если справа к нему приписать три нуля?
8. Докажите, что в любой позиционной системе счисления с основанием $P \geq 3$ число 121_P является полным квадратом.

9. Запишите в 6-ричной системе счисления число, следующее по порядку за числом 5.
10. Какое число следует за числом 111_{14} в 14-ричной системе счисления?
11. Какое число предшествует числу 10_{18} в 18-ричной системе счисления?
12. Выпишите в пятеричной системе счисления все четные числа из диапазона от 1 до 20.
13. Даны числа в четверичной системе счисления от 1 до 33_4. Выпишите все нечетные числа.
14. Запишите в системе счисления с основанием 234 число 235.
15. Запишите в системе счисления с основанием 240 числа 241, 242, 243, 250, 251.
16. Подсчитайте количество троичных чисел в диапазоне от 12_3 до 1000_3.
17. Назовем круглыми все числа, записываемые одной цифрой и несколькими нулями (быть может, одним). Выпишите все двузначные и трехзначные круглые числа в 5-ричной системе счисления.
18. В каких системах счисления 10_P является нечетным числом?
19. Как будет выглядеть в двоичной системе счисления десятичное число 0,125?

§ 1.4. Арифметические операции в *P*-ичных системах счисления

Во всех позиционных системах счисления арифметические операции выполняются по одним и тем же правилам согласно соответствующим таблицам сложения и умножения. Для всех систем счисления справедливы одни и те же законы арифметики: коммутативный, ассоциативный, дистрибутивный, а также правила сложения, вычитания, умножения и деления столбиком.

1.4.1. Сложение

В *P*-ичной системе счисления таблица сложения представляет собой результаты сложения каждой цифры алфавита *P*-ичной системы с любой другой цифрой этой же системы. Составить подобную таблицу нетрудно. Наиболее простыми являются таблицы сложения в двоичной и троичной системах счисления (индексы 2 и 3 опущены).

Таблицы сложения двоичной и троичной систем счисления

+	0	1
0	0	1
1	1	10

+	0	1	2
0	0	1	2
1	1	2	10
2	2	10	11

Приведем также таблицу сложения в шестнадцатеричной системе счисления (нижний индекс 16 в обозначении шестнадцатеричных чисел в таблице опущен).

Таблица сложения шестнадцатеричной системы счисления

+	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

Несложно показать, что если результат сложения двух цифр в P-ичной системе счисления больше $P - 1$ (т. е. полученное число двузначное), то старшая цифра результата всегда равна 1. Действительно, при сложении двух самых старших цифр алфавита мы имеем: $(P - 1) + (P - 1) = 2 \cdot P - 2 = = 1[P-2]_P$. Например, в четверичной системе счисления: $3 + 3 = 12_4$.

Следовательно, при сложении столбиком в любой системе счисления в следующий разряд может переходить только единица, а результат выполнения сложения в любом разряде будет меньше, чем $2 \cdot P$ (максимум $2 \cdot P - 1 = 1[P - 1]_P$, с учетом переноса единицы из предыдущего разряда). То есть результат сложения двух поло-

жительных P-ичных чисел либо имеет столько же значащих цифр, что и максимальное из двух слагаемых, либо на одну цифру больше, но этой цифрой может быть только единица.

Пример 11. Сложение столбиком в двоичной, троичной и шестнадцатеричной системах счисления.

$$\begin{array}{r} 101{,}01_2 \\ +\ 1{,}11_2 \\ \hline 111{,}00_2 \end{array} \qquad \begin{array}{r} 21_3 \\ +\ 2{,}1_3 \\ \hline 100{,}1_3 \end{array} \qquad \begin{array}{r} \mathrm{F2A}_{16} \\ +\ \mathrm{E9}_{16} \\ \hline 1013_{16} \end{array}$$ □

1.4.2. Вычитание

Вычитание из большего числа меньшего в P-ичной системе счисления можно производить столбиком аналогично вычитанию в десятичной системе. Для выполнения этой операции будем также использовать таблицу сложения в P-ичной системе счисления.

Если нам необходимо вычесть из цифры a цифру b и $a \geq b$, то в столбце «b» таблицы сложения ищем число a. Самая левая цифра в строке, в которой расположено число a, и будет результатом вычитания. Если же $a < b$, то, занимая единицу из левого разряда, мы придем к необходимости выполнения следующего действия: $10_P + a - b = 1a_P - b$. Для этого в столбце «b» таблицы сложения мы уже ищем число $1a_P$, левая цифра в соответствующей строке является результатом вычитания.

Пример 12. Вычитание в двоичной, троичной и шестнадцатеричной системах счисления.

$$\begin{array}{r} 101_2 \\ -\ 10{,}1_2 \\ \hline 10{,}1_2 \end{array} \qquad \begin{array}{r} 210_3 \\ -\ 102_3 \\ \hline 101_3 \end{array} \qquad \begin{array}{r} \mathrm{A10}_{16} \\ -\ 102_{16} \\ \hline \mathrm{90E}_{16} \end{array}$$ □

1.4.3. Умножение

Для выполнения умножения двух многозначных чисел в P-ичной системе счисления надо иметь таблицы умножения и сложения в этой системе.

Приведем таблицы умножения для двоичной, троичной и шестнадцатеричной систем, опуская нижние ин-

дексы, указывающие на принадлежность к соответствующей системе счисления.

Таблицы умножения двоичной и троичной систем счисления

×	0	1
0	0	0
1	0	1

×	0	1	2
0	0	0	0
1	0	1	2
2	0	2	11

Таблица умножения шестнадцатеричной системы счисления

×	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	2	4	6	8	A	C	E	10	12	14	16	18	1A	1C	1E
3	3	6	9	C	F	12	15	18	1B	1E	21	24	27	2A	2D
4	4	8	C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	5	A	F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	6	C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	7	E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	8	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	9	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

Приведем примеры выполнения умножения в двоичной, троичной и шестнадцатеричной системах. Действия производятся по правилам умножения столбиком (последовательное умножение цифр второго сомножителя на первый сомножитель и сложение промежуточных результатов), при этом используются соответствующие таблицы умножения и сложения.

Пример 13. Умножение в различных системах счисления.

$$\begin{array}{r} 10100_2 \\ \times\ 101_2 \\ \hline 101 \\ +\ 101 \\ \hline 1100100_2 \end{array}$$

Умножение на 0 не производится. Все оставшиеся справа нули, не участвующие в умножении, приписываются справа к результату умножения.

$$
\begin{array}{r}
212_3 \\
\times\ 1210_3 \\
\hline
212 \\
+1201 \\
\hline
12222 \\
+\ 212 \\
\hline
1111220_3
\end{array}
$$

При сложении столбиком трех и более слагаемых действия сложения целесообразно производить последовательно, так как сложные вычисления в непривычной системе счисления могут породить ошибки.

$$
\begin{array}{r}
\mathrm{FFA,3}_{16} \\
\times\ \mathrm{D,E}_{16} \\
\hline
\mathrm{DFAEA} \\
+\mathrm{CFB4\ 7} \\
\hline
\mathrm{DDAF,5A}_{16}
\end{array}
$$

При умножении P-ичных дробей количество цифр в дробной части результата равно сумме количеств цифр в дробных частях множителей (если одна или более крайних справа цифр результата окажутся равными нулю, то их можно опустить как незначащие). □

Возможен и другой подход к выполнению арифметических операций. Можно перевести каждый из сомножителей в десятичную систему счисления, произвести требуемое действие в десятичной системе, а результат записать в исходной P-ичной системе счисления. Аналогичным способом можно поступать и при выполнении операций сложения и вычитания.

1.4.4. Деление

При делении столбиком в P-ичной системе счисления приходится в качестве промежуточных вычислений выполнять действия умножения и вычитания, следовательно, используются таблицы умножения и сложения.

Пример 14. Наиболее просто деление организовать в двоичной системе, так как в ней необходимо лишь сравнивать два числа между собой и вычитать из большего числа меньшее.

$$
\begin{array}{r|l}
11110_2 & 110_2 \\
\cline{2-2}
-\ 110_2 & 101_2 \\
\cline{1-1}
110 & \\
-\ 110 & \\
\cline{1-1}
0 &
\end{array}
$$

□

Пример 15. Деление столбиком в шестнадцатеричной системе счисления.

$$
\begin{array}{l|l}
F127_{16} & 8_{16} \\
\underline{-\,8} & \underline{\phantom{8_{16}}} \\
71 & 1E24{,}E_{16} \\
\underline{-\,70} & \\
12 & \\
\underline{-\,10} & \\
27 & \\
\underline{-\,20} & \\
70 & \\
\underline{-\,70} & \\
0 &
\end{array}
$$

□

Однако результат деления не всегда является конечной P-ичной дробью (или целым числом). Тогда при осуществлении операции деления обычно требуется выделить непериодическую часть дроби и ее период. Продемонстрируем это на нескольких примерах.

Пример 16. Деление в троичной системе счисления.

$$
\begin{array}{l|l}
10_3 & 2_3 \\
\underline{-\,2} & \underline{} \\
10 & 1{,}(1)_3 \\
\underline{-\,2} & \\
1 &
\end{array}
$$

Так как результат последнего вычитания совпал с предыдущим, то все остальные цифры дробной части результата совпадут с последней найденной цифрой. Повторяющаяся цифра образует период троичной дроби.

Деление в двоичной системе счисления.

$$
\begin{array}{l|l}
1010_2 & 11_2 \\
\underline{-\,11} & \underline{} \\
100 & 11{,}0101\ldots_2 = 11{,}(01)_2 \\
\underline{-\,11} & \\
100 & \\
\underline{-\,11} & \\
100 & \\
\underline{-\,11} & \\
1 &
\end{array}
$$

В этом примере период дроби состоит из двух цифр. Для определения периода дроби деление выполняется до тех пор, пока не будет заметно повторение группы цифр в результате. Точнее, должно обнаружиться, что на каком-то этапе вычислений результат последнего вычитания совпал с неким предыдущим, встречавшимся ранее при подсчете именно дробной части. Следовательно, все остальные цифры дробной части результата будут повторяться такими же группами. Повторяемая группа и образует период дроби, в данном случае двоичной. □

Вопросы и задания

1. Подсчитайте сумму троичных чисел в диапазоне от 10_3 до 100_3, включая границы диапазона. Ответ запишите в троичной системе счисления.
2. Найдите сумму шестнадцатеричных чисел:
   F,1 + E,2 + D,3 + C,4 + B,5 + A,6 + 6,A + 5,B + 4,C + + 3,D + 2,E + 1,F.
   Ответ запишите в десятичной системе счисления.
3. Выпишите таблицы сложения и умножения в двенадцатеричной системе счисления.
4. Объясните, почему любая таблица сложения (и умножения) симметрична относительно главной диагонали (линии, проведенной из левого верхнего угла таблицы в ее правый нижний угол).
5. Число, записанное в десятичной системе счисления, оканчивается цифрой 5. Будет ли оно делиться на 5, если его записать в троичной системе счисления?
6. Будут ли справедливы признаки делимости натуральных чисел на 2, 3, 5, 9, 10, сформулированные для десятичной системы счисления, и в других системах?
7. В каком случае при прибавлении единицы к числу в *P*-ичной системе счисления количество цифр в числе-результате возрастет по сравнению с исходным числом? Может ли количество цифр возрасти больше, чем на одну?
8. Выполните операции сложения и вычитания над следующими парами чисел:
   11010101_2 и 1110_2; 1234_5 и 4321_5; $BABA_{16}$ и $ABBA_{16}$.
9. Выполните операцию умножения над следующими парами чисел:
   11010101_2 и 1110_2; 4321_5 и 123_5; $ABBA_{12}$ и $10A_{12}$.
10. Выполните операцию деления над следующими парами чисел:
    10010000_2 и 1100_2; 4322_5 и 3_5; $AB06_{12}$ и A_{12}.
11. В следующих примерах найдите пропущенные цифры, обозначенные знаком «*», определив вначале, в какой системе счисления выполняются действия:

а)
$$\begin{array}{r} 2*21 \\ +\ 123* \\ \hline *203 \end{array}$$

б)
$$\begin{array}{r} 5*55 \\ +\ *327 \\ \hline *16*4 \end{array}$$

в)
$$\begin{array}{r} 21*02 \\ +\ *1212 \\ \hline *2*021 \end{array}$$

г)
$$\begin{array}{r} 1*01 \\ +\ 1** \\ \hline 10100 \end{array}$$

§ 1.5. Перевод чисел из *P*-ичной системы счисления в десятичную

1.5.1. Перевод целых *P*-ичных чисел

Дано число в *P*-ичной системе счисления $a = a_n a_{n-1} \ldots a_1 a_0$. Требуется получить запись этого числа в десятичной системе счисления. Для решения задачи представим число в развернутой форме: $a = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0$ (формула (1.7)). Для того чтобы получить значение этого многочлена в десятичной системе счисления, следует число P и коэффициенты при степенях P (цифры *P*-ичного числа) записать в виде десятичных чисел и все вычисления провести в десятичной системе. Данный способ можно сформулировать в виде следующего алгоритма.

Алгоритм перевода целых чисел из *P*-ичной системы счисления в десятичную

1. Каждая цифра *P*-ичного числа переводится в десятичную систему.
2. Полученные числа нумеруются справа налево, начиная с нуля.
3. Число P переводится в десятичную систему.
4. Десятичное число, соответствующее каждой *P*-ичной цифре, умножается на P^k, где k — номер этого числа (п. 2), результаты складываются, причем все арифметические действия проводятся в десятичной системе. △

Пример 17. Переведем число $B0F9_{16}$ в десятичную систему счисления.

$$B0F9_{16} = [11_{10}][0][15_{10}][9] = \\ = 11_{10} \cdot 16_{10}^3 + 15_{10} \cdot 16_{10} + 9 = 45\ 305_{10}.$$ □

При вычислении десятичного значения *P*-ичного числа по развернутой форме удобно пользоваться *схемой Горнера*, которая позволяет получить результат с использованием минимального числа арифметических операций сложения и умножения/деления (операция возведения в степень не используется). Схема Горнера основана на следующих тождественных преобразованиях исходного степенного ряда (многочлена):

$$
\begin{aligned}
R &= a_n P^n + a_{n-1}P^{n-1} + \ldots + a_1 P^1 + a_0 = \\
&= (a_n P^{n-1} + a_{n-1}P^{n-2} + \ldots + a_1)P + a_0 = \\
&= ((a_n P^{n-2} + a_{n-1}P^{n-3} + \ldots + a_2)P + a_1)P + a_0 = \qquad (1.10)\\
&= (((a_n P^{n-3} + a_{n-1}P^{n-4} + \ldots + a_3)P + a_2)P + a_1)P + a_0 = \\
&= (\ldots(((a_n P + a_{n-1})P + a_{n-2})P + a_{n-3})P + \ldots + a_1)P + a_0.
\end{aligned}
$$

Пример 18. Применим схему Горнера для перевода в десятичную систему числа 2143_5.

$$
\begin{aligned}
2143_5 &= 2\cdot 5^3 + 1\cdot 5^2 + 4\cdot 5 + 3 = \\
&= ((2\cdot 5 + 1)\cdot 5 + 4)\cdot 5 + 3 = 298_{10}.
\end{aligned}
$$ □

Задание. *Подсчитайте, сколько операций сложения и умножения потребуется при переводе числа* $B0F9_{16}$ *в десятичную систему «в лоб» и при использовании схемы Горнера.*

Для *двоичной* системы описанный выше алгоритм перевода чисел из P-ичной системы в десятичную выглядит следующим образом.

Алгоритм перевода целых чисел из двоичной системы счисления в десятичную

Для того чтобы перевести число из двоичной системы счисления в десятичную, надо в десятичной системе счисления сложить все степени двоек, которые соответствуют единицам в записи исходного двоичного числа. Нумерация степеней ведется справа налево, начиная с нулевой. △

Пример 19. Переведем двоичное число 1001101_2 в десятичное.

$$1001101_2 = 2^0 + 2^2 + 2^3 + 2^6 = 77_{10}.$$ □

Двоичная система счисления широко используется в информатике и вычислительной технике, поэтому очень полезным оказывается знание по крайней мере первых шестнадцати степеней двойки:

$2^3 = 8$;	$2^7 = 128$;	$2^{11} = 2048$;	$2^{15} = 32\ 768$;
$2^4 = 16$;	$2^8 = 256$;	$2^{12} = 4096$;	$2^{16} = 65\ 536$.
$2^5 = 32$;	$2^9 = 512$;	$2^{13} = 8192$;	
$2^6 = 64$;	$2^{10} = 1024$;	$2^{14} = 16\ 384$;	

1.5.2. Перевод конечных *P*-ичных дробей

Дана правильная конечная дробь b в P-ичной системе счисления: $b = 0,b_{-1}b_{-2}...b_{-k}$. Требуется получить запись этой дроби в десятичной системе счисления.

Способ 1

Для решения этой задачи представим дробь в развернутой форме $b = b_{-1}P^{-1} + b_{-2}P^{-2} + ... + b_{-k}P^{-k}$ (формула (1.8)). Для того чтобы вычислить значение многочлена в десятичной системе счисления, следует число P и коэффициенты многочлена (цифры P-ичного числа) записать в виде десятичных чисел и все вычисления проводить в десятичной системе. Запишем эти правила в виде алгоритма.

Алгоритм перевода конечной *P*-ичной дроби в десятичную

1. Целая часть числа переводится в десятичную систему отдельно (см. п. 1.5.1).
2. Каждая цифра дробной части P-ичного числа переводится в десятичную систему.
3. Полученные числа нумеруются слева направо, начиная с единицы.
4. Число P переводится в десятичную систему.
5. Десятичное число, соответствующее каждой P-ичной цифре, умножается на P^{-k}, где k — номер этого числа, результаты складываются, причем все арифметические действия проводятся в десятичной системе. △

Пример 20. Переведем число $0,B0F9_{16}$ в десятичную систему счисления.

$$0,B0F9_{16} = 0,[11_{10}][0][15_{10}][9] = 11\cdot16^{-1} + 15\cdot16^{-3} + 9\cdot16^{-4} =$$
$$= 0,6912994384765625_{10}.$$

Здесь, согласно пункту 3 алгоритма, числа были пронумерованы так: 11 — номер один (коэффициент при P в степени минус один), 15 — номер три, 9 — номер четыре. □

При вычислении десятичного значения P-ичной дроби по развернутой форме также рекомендуется пользоваться *схемой Горнера*, это минимизирует количество необходимых арифметических действий. Для того чтобы получить схему Горнера для вычисления значения P-ич-

ной дроби, выпишем цифры дроби в представлении (1.8) в обратном порядке:

$$b_{-k}P^{-k} + b_{-k+1}P^{-k+1} + \ldots + b_{-1}P^{-1} =$$
$$= (b_{-k}P^{-k} + 1 + b_{-k+1}P^{-k+2} + \ldots + b_{-1})P^{-1} =$$
$$= ((b_{-k}P^{-k+2} + b_{-k+1}P^{-k+3} + \ldots + b_{-2})P^{-1} + b_{-1})P^{-1} =$$
$$\ldots$$
$$= (\ldots(((b_{-k}P^{-1} + b_{-k+1})P^{-1} + b_{-k+2})P^{-1} + b_{-k+3})P^{-1} +$$
$$+ \ldots + b_{-1})P^{-1}. \qquad (1.11)$$

Пример 21. Вычислим значение двоичной дроби $0{,}1101_2$, используя схему Горнера.

$$0{,}1101_2 = 1\cdot 2^{-4} + 0\cdot 2^{-3} + 1\cdot 2^{-2} + 1\cdot 2^{-1} =$$
$$= (((1\cdot 2^{-1} + 0)\cdot 2^{-1} + 1)\cdot 2^{-1} + 1)\cdot 2^{-1} =$$
$$= \left(\left(\left(\frac{1}{2} + 0\right):2 + 1\right):2 + 1\right):2 = \left(\left(\frac{1}{2}\cdot\frac{1}{2} + 1\right):2 + 1\right):2 =$$
$$= (1{,}25:2 + 1):2 = (0{,}625 + 1):2 = 0{,}8125. \qquad \square$$

Способ 2

Представим конечную P-ичную дробь в виде обыкновенной дроби. Числителем этой дроби будет число, стоящее после запятой, а знаменателем — P^n, где n — количество значащих цифр в дробной части. Знаменатель записывается в десятичной системе. Далее числитель запишем в десятичной системе, и мы получим обыкновенную дробь в десятичной системе. При необходимости ее можно записать в виде десятичной дроби (конечной или периодической, выделяя непериодическую часть и период).

Пример 22. Переведем число $0{,}13_{15}$ в десятичную систему счисления.

$$0{,}13_{15} = \frac{13_{15}}{15^2_{10}} = \frac{18_{10}}{225_{10}} = \frac{2_{10}}{25_{10}} = 0{,}08_{10}.$$

$\square$

Вопрос. *В каких случаях целесообразно использовать способ 1, а в каких — способ 2?*

Ответ. Способ 2 наиболее эффективен в случаях, когда среди простых делителей основания системы счисления P

содержатся какие-нибудь числа, кроме 2 и 5, и соответствующую конечную P-ичную дробь невозможно представить в виде конечной десятичной дроби. □

Пример 23. Переведем число $0,1A_{15}$ в десятичную систему счисления.

$$0,1A_{15} = \frac{1A_{15}}{15^2_{10}} = \frac{25_{10}}{225_{10}} = \frac{1_{10}}{9_{10}} = 0,(1)_{10}\ .$$ □

1.5.3. Перевод периодических P-ичных дробей

Существует несколько способов перевода бесконечной периодической дроби в десятичную систему, причем все они касаются лишь преобразования периода, а непериодическую дробную часть из P-ичной системы счисления в десятичную следует переводить отдельно, используя один из способов для конечных дробей.

Способ 1

Пусть непериодическая часть у P-ичной дроби отсутствует и дробь имеет вид $a = 0,(a_1a_2...a_k)_P$, т. е. a — чисто периодическая дробь. Умножим эту дробь на P^k, т. е. передвинем запятую на k позиций вправо. В результате мы получим некоторое число b:

$$b = aP^k = a_1a_2...a_k,(a_1a_2...\ a_k)_P = a_1a_2...a_k + a.$$

Отсюда получаем уравнение $aP^k = a + a_1a_2...a_k$, преобразовав его, находим $a = \dfrac{a_1a_2...a_k}{P^k - 1}$. В результате для записи числа a мы получили обыкновенную дробь. Знаменатель в ней записан в десятичной системе счисления. Переведя целое P-ичное число $a_1a_2...a_k$ (числитель) в десятичную систему, мы получим обыкновенную дробь, записанную в десятичной системе, равную исходному числу a. При необходимости ее можно перевести в десятичную дробь.

Пример 24. Пусть $a = 0,(1001)_2$, тогда

$$b = 2^4 \cdot a = 1001,(1001)_2 = a + 1001_2.$$

Мы получаем уравнение $16 \cdot a = a + 9$, из которого находим: $a = \dfrac{9}{15} = \dfrac{3}{5} = 0,6_{10}$. □

При наличии непериодической части вычисления изменятся незначительно. Пусть периодическая P-ичная дробь имеет вид: $b = 0,c_1c_2...c_n(a_1a_2...a_k)_P$. Обозначим через a следующую дробь: $a = 0,(a_1a_2...a_k)_P$. Тогда исходную дробь можно выразить через a так: $b = P^{-n}a + 0,c_1c_2...c_n$ (сдвигаем период на n P-ичных разрядов вправо и прибавляем непериодическую часть). Далее переводим отдельно в десятичную систему конечную дробь $0,c_1c_2...c_n$ и чисто периодическую дробь $a = 0,(a_1a_2...a_k)_P$.

Пример 25. $0,10(1001)_2 = 2^{-2}\cdot 0,(1001)_2 + 0,1_2 =$
$= 2^{-2}\cdot 0,6_{10} + 0,1_2 = 0,25_{10}\cdot 0,6_{10} + 0,5_{10} = 0,65_{10}$.
$0,00(1001)_2 = 2^{-2}\cdot 0,(1001)_2 = 2^{-2}\cdot 0,6_{10} = 0,15_{10}$.
Непериодическая часть последней двоичной дроби состоит из двух нулей, но ее нельзя считать отсутствующей, так как она определяет сдвиг периода. □

Способ 2

Запишем исходную бесконечную периодическую дробь в виде следующей бесконечной суммы:

$$0,c_1c_2...c_n(a_1a_2...a_k)_P = 0,c_1c_2...c_n + P^{-n-k}a_1a_2...a_k + P^{-n-2k}a_1a_2...a_k + P^{-n-3k}a_1a_2...a_k + ...$$

За исключением непериодической части данное выражение соответствует сумме бесконечной геометрической прогрессии со знаменателем $q = P^{-k} < 1$ и первым членом $P^{-n-k}\cdot a_1a_2...a_k$. Как известно, сумма такой прогрессии конечна и равна:

$$\frac{P^{-n-k}\cdot a_1a_2...a_k}{1-P^{-k}} = \frac{a_1a_2...a_k}{P^{n+k}\cdot\dfrac{P^k-1}{P^k}} = \frac{a_1a_2...a_k}{P^n\cdot(P^k-1)}.$$

Остается лишь записать числитель и знаменатель полученного отношения в десятичной системе счисления и прибавить к результату непериодическую часть дроби, предварительно переведенную в десятичную систему.

Пример 26. Переведем двоичную периодическую дробь $0,10(1001)_2$ в десятичную. Для этого представим исходную дробь в виде:

$$0,10(1001)_2 = 0,1_2 + 2^{-6}\cdot 1001_2 + 2^{-10}\cdot 1001_2 + 2^{-14}\cdot 1001_2 + ...$$

Легко увидеть, что слагаемые этой бесконечной суммы, начиная со второго, представляют собой бесконечную геометрическую прогрессию со знаменателем $q = 2^{-4}$ и первым членом $b = 2^{-6}\cdot 1001_2$. Сумма этой бесконечной геометрической прогрессии равна $\dfrac{2^{-6}\cdot 1001_2}{1-2^{-4}}$. Переведем числитель полученной величины в десятичную систему счисления: $2^{-6}\cdot 1001_2 = 2^{-6}\cdot(2^3+1) = 2^{-6}\cdot 9$.

Тогда $0{,}10(1001)_2 = 0{,}1_2 + \dfrac{9}{4\cdot(2^4-1)} = 0{,}5 + \dfrac{9}{4\cdot 15} = 0{,}65_{10}$. □

Вопросы и задания

1. Переведите в десятичную систему счисления двоичные числа:
   1_2; 101_2; 10000_2; 1000101010_2; 11001011_2.
2. Переведите в десятичную систему счисления числа, записанные в пятеричной системе счисления:
   1_5; 301_5; 121234_5.
3. Переведите из двоичной системы счисления в десятичную числа $0{,}0(0011)_2$ и $0{,}(001)_2$. Здесь в скобках указаны периоды бесконечных двоичных дробей.
4. Переведите следующие числа в десятичную систему счисления: 123_4; $123{,}4_5$; $203{,}5_6$; $0{,}(C)_{16}$.
5. Подсчитайте количество используемых операций (сложения и умножения) при вычислении десятичного значения числа $ABCDA9_{16}$ по развернутой форме записи и по схеме Горнера. Вместо операции возведения в степень используются несколько операций умножения.
6. Дано действительное число x. Дайте словесную запись алгоритма, позволяющего не более чем за четыре умножения и четыре сложения вычислить значение следующего выражения: $2x^4 + 3x^3 + 4x^2 + 5x + 6$.

§ 1.6. Перевод чисел из десятичной системы счисления в *P*-ичную

1.6.1. Два способа перевода целых чисел

Способ 1. Перевод делением на P с остатком.

Запишем число a в P-ичной системе счисления в развернутой форме (1.7):

$$a = a_n P^n + a_{n-1} P^{n-1} + \ldots + a_1 P + a_0,$$

где a_n, a_{n-1}, ... a_1, a_0 пока неизвестны. Тогда, разделив a на P с остатком, получаем целое частное:

$$a_n P^{n-1} + a_{n-1} P^{n-2} + \ldots + a_1 \qquad (1.12)$$

и a_0 в остатке, не превышающее $P - 1$. Таким образом, мы определили последнюю цифру в P-ичной записи числа a. Разделив полученное частное (1.12) вновь на P, получим в остатке значение a_1. Новое частное: $a_n P^{n-2} + a_{n-1} P^{n-1} + \ldots + a_2$. Таким образом, мы определили предпоследнюю цифру в P-ичной записи числа a. Продолжаем этот процесс, пока частное при целочисленном делении *не станет равным нулю. Более формально данный* способ можно записать в виде следующего алгоритма.

Алгоритм перевода целого числа из десятичной системы счисления в P-ичную

1. Делим исходное число a на P нацело в десятичной системе счисления и записываем в качестве нового значения десятичного числа a целую часть результата от деления.
2. Остаток от деления заменяем на соответствующую цифру в P-ичной системе счисления и приписываем ее слева к полученным ранее цифрам P-ичной записи числа a (первая полученная цифра соответствует младшему разряду).
3. Выполняем пункты 1 и 2, до тех пор пока число a не станет равным 0. △

Пример 27. Переведем число 123 в троичную систему счисления.

123:3 = 41	(0)	В скобках указаны остатки от целочисленного деления, которые являются соответствующими цифрами в троичном представлении числа.
41:3 = 13	(2)	
13:3 = 4	(1)	
4:3 = 1	(1)	
1:3 = 0	(1)	

Ответ: $123 = 11120_3$.

Переведем это же число в шестнадцатеричную систему счисления.

123:16 = 7 (11)
7:16 = 0 (7)

Заменим число 11 на шестнадцатеричную цифру B.

Ответ: $123 = 7B_{16}$. □

Способ 2. Этот способ основан на выделении максимальной степени числа P в исходном десятичном числе.

Заменим в развернутой форме записи числа в P-ичной системе все цифры на максимальную (равную $P-1$) и покажем, что и в этом случае число строго меньше, чем P^{n+1}. Очевидно, что такое число также не меньше, чем P^n, так как $a_n \geq 1$.

$$P^n \leq a_nP^n + a_{n-1}P^{n-1} + \ldots + a_1P + a_0 \leq$$
$$\leq (P-1)\cdot P^n + (P-1)\cdot P^{n-1} + \ldots + (P-1)\cdot P + (P-1) =$$
$$= P^{n+1} - 1 < P^{n+1}.$$

Последнее равенство получено с использованием формулы суммы конечного числа членов геометрической прогрессии.

Для нахождения старшей цифры (a_n) в P-ичной записи числа необходимо найти такую степень числа P, для которой выполняются неравенства: $P^n \leq a \leq P^{n+1}$, т. е. a_n будет равно целой части от деления a на P^n. Остатком же от такого деления является число $a_{n-1}P^{n-1} + \ldots + a_1P + a_0 \geq 0$. Обозначим его, как и раньше, a. Если оно равно нулю, то и все цифры a_{n-1}, ... a_1, a_0 равны 0, и вычисления заканчиваются, в противном случае мы опять ищем максимальную степень k числа P, для которой справедливо:

$$P^k \leq a_{n-1}P^{n-1} + \ldots + a_1P + a_0P^{k+1} \leq P^n.$$

Тогда $n-1-k$ следующих за a_n цифр будут равны нулю (при $n-1=k$ нулевые цифры между a_n и a_k отсутствуют), а a_k получаем в результате деления нацело a на P^k. Пока остаток от такого деления не окажется равен нулю, будем продолжать описанные действия.

Задание. *Запишите описанный способ в виде алгоритма.*

Пример 28. Переведем вторым способом число 100 в двоичную систему счисления.

Используя таблицу степеней двойки, запишем неравенства: $2^6 < 100 < 2^7$. Следовательно, двоичная запись числа 100 будет состоять из 7 цифр. Целая часть от деления 100 на 2^6 равна 1, т. е. старшая цифра искомого числа равна 1. Остаток от деления 100 на 2^6 равен 36. Так как $2^5 < 36 < 2^6$, то и вторая слева цифра равна единице. Остаток же от деления 36 на 2^5 равен $4 = 2^2$, т. е. третья и четвертая, а также шестая и седьмая цифры в двоичной записи числа 100 нулевые.

Ответ: $100_{10} = 1100100_2$. □

Пример 29. Переведем в шестнадцатеричную систему счисления число 525.

Используя таблицу степеней числа 16, запишем неравенства: $16^2 < 525 < 16^3$. Следовательно, запись числа 525 в шестнадцатеричной системе будет иметь три цифры. Разделим 525 на 256, получим частное 2 и в остатке 13, таким образом, старшая цифра в шестнадцатеричной записи — 2. В силу того что $13 < 16^1$, вторая цифра в шестнадцатеричном представлении равна 0, а младшей цифрой является [13], она обозначается символом D.

Ответ: $525_{10} = 20D_{16}$. □

Заметим, что второй способ перевода эффективен лишь тогда, когда нам уже известны значения всех степеней числа P, которые не превосходят исходное число. Но преимущество такого метода состоит в естественном порядке записи получившихся P-ичных цифр (слева направо), что бывает важно при программировании: очередная полученная цифра сразу же может выводиться на печать. В первом же способе перевода все цифры надо предварительно запомнить для последующей распечатки результата в порядке, обратном их получению.

1.6.2. Перевод конечных десятичных дробей

В этом разделе мы рассмотрим перевод только конечных десятичных дробей.

Если в дроби есть ненулевая целая часть, то она переводится из десятичной системы в P-ичную отдельно. Сформулируем правила перевода дробной части.

Дана правильная конечная десятичная дробь b. Допустим, что в P-ичной системе наша дробь b имеет вид $b = 0,b_{-1}b_{-2}...b_k...$ (в P-ичной системе дробь может оказаться и бесконечной). Необходимо найти цифры b_{-1}, b_{-2}, ... , b_{-k}, ...

Запишем десятичную дробь b в развернутой форме в P-ичной системе счисления:

$$b = b_{-1}P^{-1} + b_{-2}P^{-2} + ... + b_{-k}P^{-k} + ... \qquad (1.13)$$

Умножим левую (само число) и правую части выражения (1.13) на P. В правой части получим:

$$b_{-1} + b_{-2}P^{-1} + ... + b_{-k}P^{-k+1} + ..., \qquad (1.14)$$

значит, первая цифра b_{-1} дробной части числа b в P-ичной системе равна целой части результата умножения десятичной дроби b на P. Дробную часть результата умножения обозначим через b, т. е. $b = b_{-2}P^{-1} + ... + b_{-k}P^{-k+1} + ...$, и опять умножим полученное равенство на P. В результате справа получим: $b_{-2} + b_{-3}P^{-1} + ... + b_{-k}P^{-k+2} ...$, и целая часть результата в левой части будет равна b_{-2} (вторая искомая цифра). Этот процесс необходимо продолжать до тех пор, пока дробная часть результата умножения левой части на P не будет равна нулю или не будет выделен период из повторяющихся цифр b_{-i}. Иногда процесс можно прервать раньше, когда уже достигнута необходимая *точность вычислений*.

Сформулируем описанные выше правила перевода десятичных дробей в P-ичную систему в виде алгоритма.

Алгоритм перевода правильной конечной десятичной дроби в P-ичную систему счисления

1. Умножим исходное число на P (основание системы счисления), целая часть полученного произведения является первой цифрой после запятой в искомом числе.
2. Если дробная часть произведения не равна 0, умножим ее на P, целую часть полученного числа заменим на цифру в P-ичной системе и припишем ее справа к результату.
3. Выполняем пункт 2 до тех пор, пока дробная часть произведения не станет равной нулю или не выделится период (дробная часть окажется равной уже получавшейся ранее дробной части произведения). △

Пример 30. Переведем число 0,375 в двоичную систему.

$0{,}375 \cdot 2 = 0{,}75 \mid 0$ — первая цифра результата
$0{,}75 \cdot 2 = 1{,}5 \mid 1$ — вторая цифра результата
$0{,}5 \cdot 2 = 1{,}0 \mid 1$ — последняя цифра результата

Ответ: $0{,}375 = 0{,}011_2$.

Переведем число 0,515625 в четверичную систему.

$0{,}515625 \cdot 4 = 2{,}0625 \mid 2$
$0{,}0625 \cdot 4 = 0{,}25 \mid 0$
$0{,}25 \cdot 4 = 1{,}0 \mid 1$

Ответ: $0{,}515625 = 0{,}201_4$.

Переведем число 0,109375 в шестнадцатеричную систему.

$0{,}109375 \cdot 16 = 1{,}75 \mid 1$
$0{,}75 \cdot 16 = 12{,}0 \mid 12_{10} = C_{16}$

Ответ: $0{,}109375 = 0{,}1C_{16}$. □

Пример 31. Переведем число 0,123 в пятеричную систему.

$0{,}123 \cdot 5 = 0{,}615 \mid 0$
$0{,}615 \cdot 5 = 3{,}075 \mid 3$
$0{,}075 \cdot 5 = 0{,}375 \mid 0$
$0{,}375 \cdot 5 = 1{,}875 \mid 1$
$0{,}875 \cdot 5 = 4{,}375 \mid 4$

Дробная часть последнего произведения равна уже встречавшейся ранее дробной части, следовательно, последние две цифры образуют период пятеричной дроби.

Ответ: $0{,}123 = 0{,}030(14)_5$. □

Задание. *Докажите, что при переводе конечных десятичных дробей в Р-ичную систему счисления можно получить или конечную Р-ичную дробь, или периодическую дробь (бесконечную непериодическую дробь получить нельзя).*

Вопросы и задания

1. Переведите десятичное число 52 в двоичную, восьмеричную и 11-ричную системы счисления.
2. Переведите число 2005 в систему счисления с основанием, равным вашему возрасту. Может ли в новой системе счисления получившееся число быть дробным?
3. Переведите следующие десятичные дроби в троичную и восьмеричную системы счисления: 0,1; 0,3; 0,8.

4. Требуется выбрать 5 различных гирь так, чтобы с их помощью можно было взвесить любой груз до 30 кг включительно при условии, что гири ставятся только на одну чашу весов. (Эта задача приведена в книге Л. Фибоначчи. Ею также интересовался Л. Эйлер.)
5. Переведите в восьмеричную систему счисления конечную шестнадцатеричную дробь $BF3,6_{16}$.
6. Найдите 1999-ю цифру после запятой в четверичной записи десятичного числа 20,45.

§ 1.7. Смешанные системы счисления

В некоторых случаях числа, заданные в системе счисления с основанием Q, приходится изображать с помощью цифр другой P-ичной системы счисления.

Определение 8. Системы счисления, в которых каждый коэффициент разложения числа по степеням Q (цифра Q-ичной системы счисления) записывается в P-ичной системе счисления, называются *смешанными*. Иначе такие системы называют *P–Q-ичными*.

Например, ранее широкое распространение в вычислительной технике имела двоично-десятичная система. В двоично-десятичной системе счисления основанием системы счисления является число 10, но все десятичные цифры отдельно кодируются четырьмя двоичными цифрами и в таком виде записываются последовательно друг за другом. Так, число 839_{10} в двоично-десятичной системе счисления будет записываться как 100000111001_{2-10}. Заметим, что такое представление обладает избыточностью, поскольку четыре двоичные цифры могут кодировать не 10, а 16 различных чисел.

Особый интерес представляет случай, когда $Q = P^m$, где m — натуральное число. Для таких систем вид числа в P–Q-ичной системе совпадает с видом числа в P-ичной системе. Тогда перевод чисел из P-ичной системы счисления в Q-ичную и наоборот может производиться по более простым алгоритмам (сформулируем и докажем их пока только для целых чисел).

Теорема 2. Для того чтобы перевести целое число из системы счисления с основанием P в систему счисления с основанием $Q = P^m$, где m — натуральное число, достаточно запись числа в P-ичной системе разбить на группы по m цифр, начиная с правой цифры, и каждую такую группу заменить одной цифрой в Q-ичной системе.

Например: $10101_2 = 10|101 = 25_8$ ($P = 2$; $Q = 8$; $m = 3$).

Доказательство. Запишем исходное число в развернутом виде в системах счисления с основаниями P и Q:

$$a_0 + a_1P + a_2P^2 + \ldots + a_{n_1}P^{n_1} = \\ = b_0 + b_1Q + b_2Q^2 + \ldots + b_{n_2}Q^{n_2}. \qquad (1.15)$$

Преобразуем выражение в левой части равенства следующим образом (разобем его на группы по m членов и заменим P^m на Q):

$$(a_0 + a_1P + \ldots + a_{m-1}P^{m-1}) + \\ + P^m(a_m + a_{m+1}P + \ldots + a_{2m-1}P^{m-1}) + P^{2m}(\ldots) + \ldots = \\ = Q^0(a_0 + a_1P + \ldots + a_{m-1}P^{m-1}) + \\ + Q(a_m + a_{m+1}P + \ldots a_{2m-1}P^{m-1}) + Q^2(\ldots) + \ldots .$$

Покажем, что любой многочлен в скобках строго меньше Q. Для этого каждую цифру P-ичной системы счисления a_i заменим на максимальную цифру $[P-1]$ алфавита этой системы:

$$a_0 + a_1P + \ldots + a_{m-1}P^{m-1} \le (P-1)(1 + P + P^2 + \ldots + P^{m-1}) = \\ = (P-1)\frac{P^m - 1}{P-1} < Q.$$

В приведенных преобразованиях была применена формула суммы конечного числа элементов геометрической прогрессии со знаменателем P и первым членом, равным единице.

Следовательно, каждый многочлен в скобках при степенях Q можно записать в виде одной цифры Q-ичной системы счисления. В силу единственности представления натуральных чисел в любой системе счисления:

$$a_0 + a_1P + \ldots + a_{m-1}P^{m-1} = b_0;$$

$$a_m + a_{m+1}P + \ldots + a_{2m-1}P^{m-1} = b_1 \text{ и т. д.}$$

Теорема доказана.

Теорема 3. Для того чтобы перевести целое число из системы счисления с основанием $Q = P^m$, где m — натуральное число, в систему счисления с основанием P, необходимо каждую Q–ичную цифру перевести в систему с основанием P и дополнить, если это необходимо, полученные числа слева нулями так, чтобы каждое число, за исключением самого левого, состояло ровно из m цифр.

Например:
$73_{16} = 111|0011 = 1110011_2$ ($P = 2$; $Q = 16$; $m = 4$).

Доказательство. Представим каждую цифру b_i, $i = 0, \ldots, n_2$ (формула (1.15)) в представлении исходного числа в Q-ичной системе счисления в P-ичной системе счисления. Так как $b_i < Q = = P^m$, то максимальное количество цифр в полученном представлении равно m.

$$b_{00} + b_{01}P + b_{02}P^2 + \ldots + b_{0(m-1)}P^{m-1} + \\ + P^m(b_{10} + b_{11}P + b_{12}P^2 + \ldots + b_{1(m-1)}P^{m-1}) + \ldots .$$

После раскрытия скобок в силу единственности представления чисел в P-ичной системе счисления получаем:

$$b_{00} = a_0;\ b_{01} = a_1;\ \ldots;\ b_{10} = a_m;\ b_{11} = a_{m+1};\ \ldots .$$

Теорема доказана.

Примечание. Аналогичные утверждения справедливы также и для правильных дробей. Перевод дробной части из Q-ичной системы в P-ичную осуществляется, как и для целых чисел. Незначащими в дробной части теперь являются правые нули в P-ичном представлении самой правой цифры дробной части Q-ичного числа. При обратном же переводе цифры P-ичной дроби группируются по m штук слева направо, начиная с первой цифры после запятой. Если последняя группа содержит менее m цифр, то к ней добавляют справа соответствующее количество нулей.

Пример 32. Переведем двоичное число $1010{,}00011011_2$ в восьмеричную систему счисления.

Для двоичной и восьмеричной систем счисления выполняется соотношение $Q = P^m$, а именно $8 = 2^3$. Следовательно, при переводе будем группировать цифры двоичного числа по три (в целой части — справа налево, в

дробной части — слева направо): $1|010,|000|110|11_2 = 12{,}066_8$ (последняя группа двоичных цифр была дополнена нулем справа).

Ответ: $1010{,}00011011_2 = 12{,}066_8$. □

Пример 33. Переведем число $A{,}1C_{16}$ из шестнадцатеричной системы счисления в четверичную.

Для шестнадцатеричной и четверичной систем счисления выполняется соотношение $Q = P^m$, а именно $16 = 4^2$. Поэтому заменим каждую 16-ричную цифру ее 4-ричным представлением, для чего используем десятичную систему в качестве промежуточной: $A_{16} = 10_{10} = 22_4$; $C_{16} = 12_{10} = 30_4$. Тогда $A{,}1C_{16} = 22{,}|01|30_4$ (последний незначащий 0 можно опустить).

Ответ: $A{,}1C_{16} = 22{,}013_4$. □

Полученные результаты для смешанных систем счисления, таких что $P^m = Q$, имеют ряд практических применений.

1. Арифметические действия над числами, записанными в одной из таких систем, вы можете выполнять в другой системе, если последняя более удобна для вас. Например, вычисления в 100-ичной системе заменяются на десятичную арифметику (100-ичные числа переводятся в десятичную систему, а результат при необходимости может быть снова записан в 100-ичной), а действия с шестнадцатеричными или восьмеричными числами легко заменяются на двоичную арифметику.
2. Замена системы счисления с меньшим основанием P на систему с бóльшим основанием $Q = P^m$ обеспечивает сокращение записи числа, уменьшая количество цифр в m раз. Например, при использовании двоичной системы счисления числа можно представлять в 16-ричной, сократив количество цифр в записи числа в 4 раза ($16 = 2^4$).
3. В некоторых случаях удается сделать более рациональным решение задачи перевода чисел из одной системы в другую, даже если непосредственно их основания не связаны соотношением $Q = P^m$. Например, при переводе чисел из восьмеричной системы в шестнадцатеричную и наоборот удобно сначала переписать число в двоичном виде ($8 = 2^3$ и $2^4 = 16$).

Пример 34. Переведем число $BF3,6_{16}$ в восьмеричную систему счисления:

$$BF3,6_{16} = 1011|1111|0011,0110_2 = 101|111|110|011,011_2 = 5763,3_8.$$ □

Вопросы и задания

1. Переведите десятичные числа $a = 645$, $b = 383$ в двоичную, восьмеричную и шестнадцатеричную системы счисления и заполните следующую таблицу:

Выражение	Система счисления			
	10-тичная	16-ричная	8-ричная	2-ичная
a	645			
b	383			
$a + b$				
$a - b$				

2. Переведите число $1234,5678_9$ в 27-ричную систему счисления, а число $ABCD,EF_{16}$ — в восьмеричную.
3. Сумму восьмеричных чисел $17 + 1700 + 170\,000 + 17\,000\,000 + 1\,700\,000\,000$ перевели в шестнадцатеричную систему счисления. Найдите в записи числа, равного этой сумме, пятую цифру слева.
4. Во сколько раз сократится количество цифр в записи числа, если его перевести из четверичной системы счисления в 64-ричную? А из десятичной в 10 000-ричную?
5. Выпишите четверичные представления для всех цифр алфавита шестнадцатеричной системы счисления и переведите число $12345678,9ABCDEF_{16}$ в четверичную систему, не используя его двоичное представление в качестве промежуточного.

§ 1.8. Системы счисления и архитектура компьютеров

В каждой области науки и техники существуют фундаментальные идеи или принципы, которые определяют ее содержание и развитие. В компьютерной науке роль таких фундаментальных идей сыграли принципы, сформулированные независимо друг от друга двумя крупней-

Дж. фон Нейман (1903-1957)

С. А. Лебедев (1902–1974)

шими учеными XX века — американским математиком и физиком Джоном фон Нейманом и советским инженером и ученым Сергеем Александровичем Лебедевым.

Центральное место среди «принципов Неймана—Лебедева», определяющих архитектуру ЭВМ, занимает предложение об использовании двоичной системы счисления. Это предложение было обусловлено рядом обстоятельств: простотой выполнения арифметических операций в двоичной системе счисления; ее «оптимальным» согласованием с булевой логикой; простотой технической реализации двоичного элемента памяти (триггера).

Однако на определенном этапе развития компьютерной техники было выявлено, что использование классической двоичной системы счисления для представления информации в компьютере имеет существенные недостатки. Первым из них является так называемая *проблема представления отрицательных чисел*. Второй недостаток двоичной системы счисления получил название *нулевой избыточности*.

Как известно, отрицательные числа непосредственно не могут быть представлены в двоичной системе счисления, использующей только две цифры 0 и 1. Перед модулем отрицательного числа необходимо ставить знак «минус». Это влечет за собой необходимость анализировать знаки операндов при выполнении арифметических операций, что снижает скорость обработки информации. Для того чтобы не выполнять анализ операндов, был разработан и реализован способ представления целых отрицательных чисел в виде дополнительного кода (см. главу 2), что существенно упростило схему выполнения арифметических операций, но затруднило восприятие записи отрицательных чисел.

Второй недостаток двоичной системы особенно неприятен при хранении и передаче двоичных кодов. Нулевая избыточность (т. е. отсутствие избыточности) двоичного представления означает, что в системе счисления отсутствует механизм обнаружения ошибок, которые, к сожалению, неизбежно возникают в компьютерных системах под влиянием внешних и внутренних факторов.

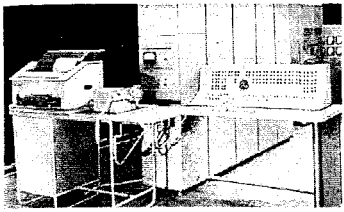
Суть этой проблемы состоит в следующем. Пусть в процессе передачи или хранения информации, представленной, например, двоичным кодом 10011010, под влиянием внешних или внутренних факторов произошло искажение информации, и она перешла в кодовую комбинацию 11010010 (искаженные разряды подчеркнуты). Поскольку комбинация 11010010 (как и любой другой двоичный код) является «разрешенной» в двоичной системе счисления, то без дополнительных действий невозможно определить, произошло искажение информации или нет. Для решения этой проблемы можно, например, для каждого байта (8 разрядов двоичного числа) подсчитывать количество единиц, или для группы байтов подсчитывать контрольную сумму и т. д. В любом случае должны быть использованы специальные методы избыточного кодирования, что замедляет работу компьютера и требует дополнительной памяти.

В условиях, когда человечество все больше и больше зависит от надежности работы компьютерных систем (управления ракетами, самолетами, атомными реакторами, банковскими системами), вопрос об эффективных механизмах обнаружения ошибок выдвигается на передний план. Ясно, что для компьютеров, основанных на двоичной системе счисления, не всегда можно эффективно решать эту проблему.

Попытка преодолеть эти и другие недостатки двоичной системы счисления стимулировала использование в компьютерах других систем счисления и развитие собственно теории систем счисления.

1.8.1. Использование уравновешенной троичной системы счисления

Для преодоления недостатков использования двоичной системы для кодирования информации уже на этапе зарождения компьютерной эры был выполнен ряд проектов и сделано несколько интересных математических открытий, связанных с системами счисления. Пожалуй, наиболее интересным проектом в этом отношении является троичный компьютер «Сетунь», разработанный в 1958 г. в Московском государственном университете им. М. В. Ломоносова под руководством Н. П. Брусенцова (Сетунь — название речки, протекающей неподалеку от МГУ).



ЭВМ «Сетунь»

В ЭВМ «Сетунь» применялась уравновешенная (симметричная) троичная система счисления для представления чисел, использование которой впервые в истории компьютеров поставило знак равенства между представлением отрицательных и положительных чисел, позволило отказаться от различных «ухищрений», используемых для представления отрицательных чисел. Это обстоятельство, а также использование «троичной логики» при разработке программного обеспечения привело к созданию весьма совершенной архитектуры компьютера.

ЭВМ «Сетунь» является наиболее ярким примером, подтверждающим влияние системы счисления на архитектуру компьютера!

Определение 9. Система счисления с основанием $P = 3$ и цифрами $\bar{1}$, 0, 1, где $\bar{1}$ означает «минус единица», называется *уравновешенной троичной* или *симметричной троичной* системой счисления.

Пример 35. Приведем примеры записи некоторых чисел в уравновешенной троичной системе:

Положительные десятичные числа	Положительные троичные уравновешенные числа	Отрицательные троичные уравновешенные числа	Отрицательные десятичные числа
1	1	$\bar{1}$	−1
2	$1\,\bar{1}$	$\bar{1}\,1$	−2
3	$1\,0$	$\bar{1}\,0$	−3
4	$1\,1$	$\bar{1}\,\bar{1}$	−4
5	$1\,1\,\bar{1}$	$\bar{1}\,1\,1$	−5
6	$1\,\bar{1}\,0$	$\bar{1}\,1\,0$	−6
7	$1\,\bar{1}\,\bar{1}$	$\bar{1}\,1\,\bar{1}$	−7
8	$1\,0\,\bar{1}$	$\bar{1}\,0\,1$	−8
9	$1\,0\,0$	$\bar{1}\,0\,0$	−9
10	$1\,0\,1$	$\bar{1}\,0\,\bar{1}$	−10
11	$1\,1\,\bar{1}$	$\bar{1}\,1\,1$	−11
12	$1\,1\,0$	$\bar{1}\,1\,0$	−12
13	$1\,1\,1$	$\bar{1}\,\bar{1}\,\bar{1}$	−13
14	$1\,1\,1\,1$	$\bar{1}\,1\,1\,1$	−14

□

Из приведенного примера понятно, почему эта система счисления называется уравновешенной или симметричной.

Главная особенность уравновешенных систем счисления — при выполнении арифметических операций не используется «правило знаков».

1.8.2. Использование фибоначчиевой системы счисления

На заре компьютерной эры было сделано еще два открытия в области позиционных способов представления чисел, которые, однако, малоизвестны и в тот период не привлекли особого внимания математиков и инженеров. Речь идет о свойствах фибоначчиевой системы счисления и системы счисления золотой пропорции.

В последние десятилетия XX века группой математиков под руководством профессора А. П. Стахова в СССР были получены чрезвычайно интересные результаты, связанные с решением проблемы надежности хранения, обработки и передачи информации в компьютерных системах. Математиками было предложено использовать в качестве системы счисления в компьютерах фибоначчиеву систему. Напомним, что алфавитом этой системы являются цифры 0 и 1, а базисом — последовательность чисел Фибоначчи: 1, 2, 3, 5, 8, 13, 21, 34 ... .

Основное преимущество кодов Фибоначчи для практических применений состоит в их «естественной» избыточности, которая может быть использована для целей контроля числовых преобразований. Эта избыточность проявляет себя в свойстве множественности представлений одного и того же числа. Например, число 30 в коде Фибоначчи имеет несколько представлений:

$$30 = 1001101_{fib} = 1010001_{fib} = 111101_{fib}.$$

При этом различные кодовые представления одного и того же числа могут быть получены друг из друга с помощью специальных фибоначчиевых операций *свертки* ($011 \rightarrow 100$) и *развертки* ($100 \rightarrow 011$), выполняемых над кодовым изображением числа. Если над кодовым изображением выполнить все возможные свертки, то мы придем к специальному фибоначчиевому изображению,

называемому *минимальной формой*, в которой нет двух рядом стоящих единиц. Если же в кодовом изображении выполнить все возможные операции развертки, то придем к специальному фибоначчиевому изображению, называемому *максимальной* или *развернутой формой*, в которой рядом не встречаются два нуля.

Анализ фибоначчиевой арифметики показал, что основными ее операциями являются операции свертки, развертки и основанная на них операция приведения кода Фибоначчи к минимальной форме.

Эти математические результаты стали основой для проекта создания компьютерных и измерительных систем на основе фибоначчиевой системы счисления.

При разработке элементной базы новой компьютерной техники основным операционным элементом стало устройство приведения кода Фибоначчи к минимальной форме. Это устройство реализовывалось через *RS*-триггеры и логические элементы И и ИЛИ. Были созданы опытные образцы микросхемы, выполняющей следующие операции: запись и чтение данных, свертка, развертка, перемещение, поглощение, приведение к минимальной форме, суммирование, вычитание, реверсивный сдвиг, логическое умножение, логическое сложение и сложение по модулю 2.

Отличительной особенностью микросхемы являлось наличие контрольного выхода, на котором формировалась информация о неправильной работе микросхемы.

Таким образом, основным результатом этой разработки было создание первой в истории компьютерной техники микросхемы для реализации самоконтролирующегося Фибоначчи-процессора со стопроцентной гарантией обнаружения сбоев, возникающих при переключении триггеров.

И хотя создать Фибоначчи-компьютер по разным причинам пока так и не удалось, теоретические основы данного направления представляют несомненный интерес и могут стать источником новых идей не только в компьютерной области, но и в области математики. Особенно эффективным считается использование «фибоначчиевых» представлений в измерительной технике и цифровой обработке сигналов.

1.8.3. Недвоичные компьютерные арифметики

При разработке вычислительной техники перед математиками всегда стоит сложнейшая проблема — создание эффективных (их часто называют «предельными») алгоритмов выполнения арифметических операций в компьютере. В рамках решения этой проблемы учеными были придуманы новые системы счисления и разработаны компьютерные арифметики на их основе, которые позволяют построить вычислительные устройства, быстродействие и надежность которых превосходят вычислители, основанные на двоичной арифметике. К таким системам счисления можно отнести непозиционную систему остаточных классов, некоторые иерархические системы счисления и др.

Иерархические системы счисления конструируются на основе идеи соединения позиционных и непозиционных систем счисления, при этом они должны сочетать в себе положительные стороны включенных в них систем и быть свободными от их недостатков. Принцип построения иерархических систем в целом прост. Выбирается некоторая внешняя система счисления A с алфавитом α. Цифры этой системы записываются в виде слов (кодов) другой (внутренней) системы счисления B с алфавитом β. В качестве примера такой системы можно привести известную вам двоично-десятичную систему, применяемую для представления десятичных чисел в некоторых компьютерах.

Система остаточных классов (СОК) — это непозиционная система счисления, числа в которой представляются остатками от деления на выбранную систему оснований $P_1, P_2, \ldots, P_n$ и являются взаимно простыми числами. Операции сложения, вычитания и умножения над числами в СОК производятся независимо по каждому основанию без переносов между разрядами.

Такие операции, как деление, сравнение и др., требующие информации о величине всего числа, в СОК выполняются по более сложным алгоритмам. И в этом заключается существенный недостаток данной системы счисления, сдерживающий ее широкое применение в качестве компьютерной. Однако сегодня в современных компьютерах при работе с большими и супербольшими числами используют СОК, ибо только СОК-арифметика позволяет получать результаты вычислений в реальном времени.

В таких случаях в качестве оснований СОК берут величины, близкие к 2^m (m — двоичная разрядность компьютера), например, $2^{m-1} - 1$, 2^{m-1}, $2^{m-1} + 1$ и т. д. Начиная с середины прошлого столетия ученые многих стран мира, включая и нашу, занимаются проблемой повышения скорости «неудобных» операций в СОК. Сама же система остаточных классов применяется в вычислительных системах достаточно широко уже несколько десятилетий.

Вопросы и задания

1. Перечислите первые 14 натуральных чисел в фибоначчиевой системе счисления в минимальной форме.
2. Сформулируйте правила перечисления натуральных чисел в фибоначчиевой системе.
3. Сформулируйте правила перечисления натуральных чисел в троичной уравновешенной системе счисления.
4. Сформулируйте правила сложения в фибоначчиевой системе счисления.
5. Сформулируйте правила сложения в троичной уравновешенной системе счисления.
6. Объясните, почему системы счисления, аналогичные используемой в ЭВМ «Сетунь», называют уравновешенными или симметричными.
7. Напишите алгоритм составления таблицы сложения в P-ичной системе счисления.
8. Напишите алгоритм составления таблицы умножения в P-ичной системе счисления.

Заключение

Сегодня многими учеными высказывается утверждение, что элементная база компьютерной техники, основанная на двоичном кодировании, скоро достигнет границы своих возможностей, и тогда, скорее всего, дальнейшее развитие этой области науки и техники будет связано с новыми математическими результатами в области кодирования информации. В XX веке в теории чисел системам счисления не уделялось должного внимания, и в этой части данная теория не намного ушла вперед по сравнению с периодом своего зарождения. Это было связано

с отсутствием серьезной потребности в использовании новых систем счисления в практике вычислений, которая в течение последних столетий всецело удовлетворялась десятичной системой, а в последние десятилетия — двоичной системой (в информатике).

Ситуация резко изменилась в результате появления современных компьютеров. Именно в информатике опять проявился интерес к способам представления чисел и к новым компьютерным арифметикам. Дело в том, что, как мы уже говорили, классическая двоичная система счисления обладает рядом принципиальных недостатков, главными из которых являются проблема представления отрицательных чисел и нулевая избыточность.

В связи с этим изучение основ построения систем счисления, свойств различных систем счисления представляет сегодня не только научный, но и практический интерес.

В главе «Системы счисления» вы получили представление о многообразии способов записи чисел, познакомились с различными типами позиционных систем счисления (традиционными, или P-ичными, смешанными, нетрадиционными), подробно рассмотрели P-ичные системы счисления: способы представления произвольных чисел и арифметические операции в P-ичных системах счисления, алгоритмы перевода чисел из одной системы счисления в другую и т. д. Вы имели возможность разобрать на примерах большое количество основных типовых задач, в том числе и встречавшихся на вступительных экзаменах по информатике в вузы. Особое внимание было уделено месту и роли систем счисления в современных компьютерах. Вы узнали об основных недостатках использования двоичной системы для кодирования информации в компьютерах, о проектах создания компьютеров, построенных на иных способах кодирования, которые позволяют решить эти проблемы (ЭВМ «Сетунь», проект Фибоначчи-компьютера).

В заключение хотелось бы отметить, что научные исследования в области систем счисления продолжаются. Математики, например, изучают нега-позиционные системы счисления, основаниями которых являются целые отрицательные числа, а также системы с основанием, содержащим мнимую единицу и т. д. Не исключено, что кому-то из вас удастся сказать свое слово в этих областях математики и информатики.

Глава 2

Представление информации в компьютере

В своей бинарной арифметике Лейбниц видел прообраз творения. Ему представлялось, что единица представляет божественное начало, а нуль — небытие, и что высшее существо создает все сущее из небытия точно таким же образом, как единица и нуль в его системе выражают все числа.

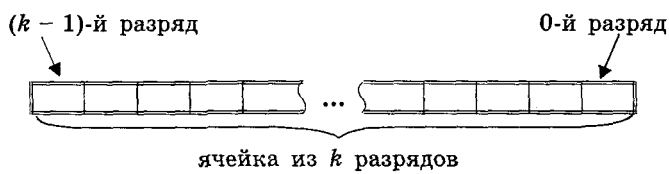
П. С. Лаплас

Очень легко делать удивительные открытия, но трудно усовершенствовать их в такой степени, чтобы они получили практическую ценность.

Т. А. Эдисон

С конца XX века, века компьютеризации, человечество ежедневно пользуется двоичной системой счисления, так как вся информация, обрабатываемая современными компьютерами, представлена в двоичном виде.

Каждый регистр арифметического устройства компьютера, каждая ячейка памяти представляет собой физическую систему, состоящую из некоторого числа однородных элементов, обладающих двумя устойчивыми состояниями, одно из которых соответствует нулю, а другое — единице. Каждый такой элемент служит для записи одного из разрядов двоичного числа. Именно поэтому каждый элемент ячейки называют *разрядом*.



Вы знаете, что вычислительная техника возникла как средство автоматизации вычислений, именно поэтому первые компьютеры назывались ЭВМ — электронно-вычислительными машинами. Сегодня компьютеры обрабатывают различные виды информации: числовую, текстовую, звуковую, графическую. Однако современный компьютер может хранить и обрабатывать только дискретную информацию. Следовательно, любой вид информации, подлежащий компьютерной обработке, тем или иным способом должен быть закодирован с помощью конечной последовательности целых чисел, которая затем переводится в двоичный вид для хранения в компьютере.

В данной главе мы рассмотрим, каким образом решается проблема преобразования исходной информации в компьютерное представление для каждого вида информации. Будет показано, насколько точно компьютерное представление отражает исходную информацию, причем слово «точно» здесь применяется не только к числам (точность представления), но и к другим видам информации. А именно, рассматривается степень реалистичности передачи оттенков цвета на мониторе, степень

приближенности воспроизводимой музыки к естественному звучанию музыкальных инструментов или голосу человека и т. д. Задача перевода информации естественного происхождения в компьютерную называется задачей *дискретизации* или *квантования*. Эту задачу необходимо решать для всех видов информации. Способы дискретизации для разных видов информации различны, но подходы к решению этой задачи построены на одинаковых принципах.

§ 2.1. Представление целых чисел

Любое целое число можно рассматривать как вещественное, но с нулевой дробной частью, т. е. можно было бы ограничиться представлением в компьютере вещественных чисел и реализацией арифметических действий над ними. Однако для эффективного использования памяти, повышения скорости выполнения вычислений и введения операции деления нацело с остатком целые числа представляются специально для них предназначенными способами.

Введение специальных способов представления целых чисел оправдано тем, что достаточно часто в задачах, решаемых с помощью компьютера, многие действия сводятся к операциям над целыми числами. Например, в задачах экономического характера данными служат количества акций, сотрудников, деталей, транспортных средств и т. д., по своему смыслу являющиеся целыми числами. Целые числа используются и для обозначения даты и времени, и для нумерации различных объектов: элементов массивов, записей в базах данных, машинных адресов и т. п.

Для компьютерного представления целых чисел обычно используется несколько различных способов представления, отличающихся друг от друга количеством разрядов и наличием или отсутствием знакового разряда. Беззнаковое представление можно использовать только для неотрицательных целых чисел, отрицательные числа представляются только в знаковом виде.

При беззнаковом представлении все разряды ячейки отводятся под само число. При представлении со знаком самый старший (левый) разряд отводится под знак чис-

ла, остальные разряды — под собственно число. Если число положительное, то в знаковый разряд помещается 0, если число отрицательное — 1. Очевидно, в ячейках одного и того же размера можно представить больший диапазон целых неотрицательных чисел в беззнаковом представлении, чем чисел со знаком. Например, в одном байте (8 разрядов) можно записать положительные числа от 0 до 255, а со знаком — только до 127. Поэтому, если известно заранее, что некоторая числовая величина всегда является неотрицательной, то выгоднее рассматривать ее как беззнаковую.

Говорят, что целые числа в компьютере хранятся в формате с *фиксированной запятой*.

2.1.1. Представление целых положительных чисел

Для получения компьютерного представления беззнакового целого числа в k-разрядной ячейке памяти достаточно перевести его в двоичную систему счисления и дополнить полученный результат слева нулями до k разрядов. Понятно, что существует ограничение на числа, которые мы можем записать в k-разрядную ячейку.

Максимально представимому числу соответствуют единицы во всех разрядах ячейки (двоичное число, состоящее из k единиц). Для k-разрядного представления оно будет равно $2^k - 1$. Минимальное число представляется нулями во всех разрядах ячейки, оно всегда равно нулю. Ниже приведены максимальные числа для беззнакового представления при различных значениях k:

Количество разрядов	Максимальное число
8	255 ($2^8 - 1$)
16	65535 ($2^{16} - 1$)
32	4294967295 ($2^{32} - 1$)
64	18446744073709551615 ($2^{64} - 1$)

При знаковом представлении целых чисел возникают такие понятия, как прямой, обратный и дополнительный коды.

Определение 1. Представление числа в привычной для человека форме «знак–величина», при которой старший разряд ячейки отводится под знак, остальные $k - 1$ разрядов — под цифры числа, называется *прямым кодом.*

Например, прямые коды двоичных чисел 11001_2 и -11001_2 для восьмиразрядной ячейки равны 00011001 и 10011001 соответственно. Положительные целые числа представляются в компьютере с помощью прямого кода. Прямой код отрицательного целого числа отличается от прямого кода соответствующего положительного числа содержимым знакового разряда. Но вместо прямого кода для представления отрицательных целых чисел в компьютере используется дополнительный код (см. п. 2.1.2).

Отметим, что максимальное положительное число, которое можно записать в знаковом представлении в k разрядах, равно $2^{k-1} - 1$, что практически в два раза меньше максимального числа в беззнаковом представлении в тех же k разрядах.

Задание. *Определите максимальное положительное число в восьмиразрядном и шестнадцатиразрядном знаковых способах представления чисел.*

Решение. Максимальное положительное число в 8 битах равно 127 ($2^7 - 1$), в 16 битах — 32767 ($2^{15} - 1$). □

Пример 1. Число $53 = 110101_2$ в восьмиразрядном представлении имеет вид:

0	0	1	1	0	1	0	1

Это же число 53 в 16 разрядах будет записано следующим образом:

0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	1

В обоих случаях неважно, знаковое или беззнаковое представление при этом используется. □

Пример 2. Для числа $200 = 11001000_2$ представление в 8 разрядах со знаком невозможно, так как максимальное допустимое число в таком представлении равно 127, а в беззнаковом восьмиразрядном представлении оно имеет вид:

1	1	0	0	1	0	0	0

□

2.1.2. Представление целых отрицательных чисел

Для представления в компьютере целых отрицательных чисел используют дополнительный код, который позволяет заменить арифметическую операцию вычитания операцией сложения, что существенно увеличивает скорость вычислений. Прежде чем вводить определение дополнительного кода, сделаем следующее важное замечание.

В k–разрядной целочисленной компьютерной арифметике $2^k \equiv 0$.

Объяснить это можно тем, что двоичная запись числа 2^k состоит из одной единицы и k нулей, а в ячейку из k разрядов может уместиться только k цифр, в данном случае только k нулей. В таком случае говорят, что значащая единица вышла за пределы разрядной сетки.

Определение 2. k–разрядный *дополнительный код* отрицательного числа m — это запись в k разрядах положительного числа $2^k - |m|$, где $|m|$ — модуль отрицательного числа m, $|m| \le 2^{k-1}$.

Разберемся, что и до чего дополнительный код дополняет. Дополнительный код отрицательного числа m — это дополнение модуля этого числа до 2^k (или до нуля в k-разрядной арифметике): $(2^k - |m|) + |m| = 2^k \equiv 0$.

Алгоритм получения дополнительного k-разрядного кода отрицательного числа

1. Модуль числа представить прямым кодом в k двоичных разрядах.
2. Значения всех разрядов *инвертировать* (все нули заменить на единицы, а единицы — на нули), получив, таким образом, k–разрядный *обратный код* исходного числа.
3. К полученному обратному коду, трактуемому как k–разрядное неотрицательное двоичное число, прибавить единицу. △

Обратный код является дополнением исходного числа до числа $2^k - 1$, состоящего из k двоичных единиц. Поэтому прибавление единицы к инвертированному коду позволяет получить его искомый дополнительный код.

Пример 3. Получим дополнительный код числа −52 для восьми- и шестнадцатиразрядной ячеек.

Для восьмиразрядной ячейки:

0011 0100 — прямой код числа $|-52| = 52$;
1100 1011 — обратный код числа −52;
1100 1100 — дополнительный код числа −52.

Для шестнадцатиразрядной ячейки:

0000 0000 0011 0100 — прямой код числа $|-52|$;
1111 1111 1100 1011 — обратный код числа −52;
1111 1111 1100 1100 — дополнительный код числа −52.

□

Вопрос. *Какое минимальное отрицательное число можно записать в k разрядах?*

Ответ. Описанный выше алгоритм получения дополнительного кода для отрицательного числа знаковую единицу в левом разряде образует автоматически при $|m| \le 2^{k-1}$. Если же $2^{k-1} < |m| < 2^k$, то попытка реализации данного алгоритма приведет к тому, что в левом разряде будет находиться цифра 0, соответствующая компьютерному представлению положительных чисел, что неверно.

Представим значение $2^k - |m|$ в следующем виде:

$$2^k - |m| = 2^{k-1} + (2^{k-1} - |m|).$$

Здесь первое слагаемое 2^{k-1} соответствует единице в левом знаковом разряде. То есть при представлении отрицательного числа m дополнительным кодом в самом левом (знаковом) разряде записывается знак отрицательного числа (единица), а в остальных разрядах — число $2^{k-1} - |m|$. Следовательно, минимальное отрицательное (максимальное по модулю) число m, которое можно представить в k разрядах, равно -2^{k-1} (это ограничение и было приведено в определении 2). □

Задание. *Постройте дополнительный восьмиразрядный код для чисел −128, −127 и −0.*

Решение. Ответы приведены в таблице ниже:

Число	−128	−127	−0
Прямой код модуля	1000 0000	0111 1111	0000 0000
Обратный код	0111 1111	1000 0000	1111 1111
Дополнительный код	1000 0000	1000 0001	0000 0000

Отметим, что для числа -128 прямой код совпадает с дополнительным, а дополнительный код числа -0 совпадает с обычным нулем. При преобразовании обратного кода для числа -0 в его дополнительный код правила обычной двоичной арифметики нарушаются, а именно:

$$1111\ 1111_2 + 1 = 1\ 0000\ 0000_2 = 2^k \equiv 0. \qquad \square$$

Восстановить модуль исходного десятичного отрицательного числа по его дополнительному коду можно двумя способами.

Способ 1 (обратная цепочка преобразований): вычесть единицу из дополнительного кода, инвертировать полученный код и перевести полученное двоичное представление числа в десятичное.

Способ 2: по приведенному выше алгоритму построить дополнительный код для имеющегося дополнительного кода искомого числа и представить результат в десятичной системе счисления.

Пример 4. Получим десятичное значение числа по его дополнительному коду 10010111_2.

Способ 1:

1) из дополнительного кода вычтем 1:

   $10010111 - 1 = 10010110$ (получили обратный код);
2) инвертируем полученный код: 01101001 (получили модуль отрицательного числа);
3) переведем полученное двоичное значение в десятичную систему счисления:

   $01101001_2 = 2^6 + 2^5 + 2^3 + 1 = 64 + 32 + 8 + 1 = 105.$

Ответ: -105.

Способ 2:

1) инвертируем имеющийся дополнительный код: 01101000;
2) прибавим к результату 1: 01101000 + 1 = 01101001 (получили модуль отрицательного числа);
3) переведем полученное двоичное значение в десятичную систему счисления:

   $01101001_2 = 2^6 + 2^5 + 2^3 + 1 = 64 + 32 + 8 + 1 = 105.$

Ответ: -105. $\square$

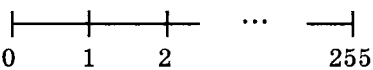
Целые числа со знаком, представимые в k разрядах, принадлежат диапазону $[-2^{k-1}, 2^{k-1} - 1]$, который не является симметричным относительно 0. Это следует учитывать при программировании. Если, например, изменить знак у наибольшего по модулю отрицательного числа, то полученный результат окажется уже не представимым в том же числе разрядов.

Ниже приведены значения границ диапазонов для знаковых представлений в ячейках с различной разрядностью:

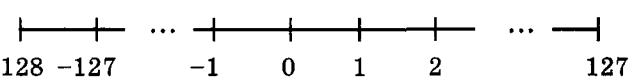
Разрядность	Минимальное число	Максимальное число
8	−128	127
16	−32768	32767
32	−2147483648	2147483647
64	−9223372036854775808	9223372036854775807

2.1.3. Перечисление чисел в целочисленной компьютерной арифметике

Мы выяснили, что в k–разрядном знаковом или беззнаковом представлении целых чисел количество представимых чисел ограничено и зависит от k. Представимые числа можно перечислить по возрастанию. Например, для восьмиразрядного беззнакового представления допустимые числа располагаются на отрезке следующим образом:

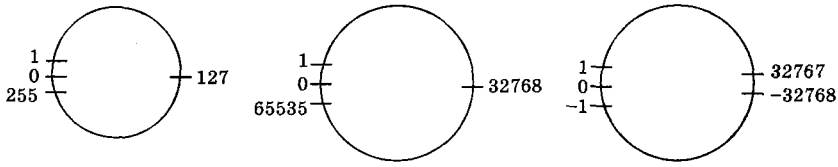


Для восьмиразрядного знакового представления:



Однако в k–разрядной компьютерной арифметике эти отрезки можно замкнуть в кольца и перечислять числа соответствующего представления по кругу.

Все целые числа любого k–разрядного представления можно зафиксировать на кольце по порядку, причем рядом с максимальным числом в том или ином представлении будет находиться минимальное, например:



8-разрядное беззнаковое | 16-разрядное беззнаковое | 16-разрядное знаковое

Действительно, результатом прибавления единицы в арифметике с фиксированным количеством разрядов является следующее по часовой стрелке за исходным значение на кольце, а результатом вычитания единицы — следующее значение по ходу против часовой стрелки. Данный факт очевиден, кроме случаев перехода от максимально допустимого числа к минимальному при прибавлении единицы и от минимального к максимальному при вычитании.

Поясним эти случаи. Рассмотрим вначале беззнаковые представления. Максимально допустимое число в k–разрядном беззнаковом представлении равно $2^k - 1$ и состоит из k единиц в двоичном представлении. При прибавлении к такому числу единицы мы получаем 2^k, что в свою очередь соответствует нулю в k-разрядной арифметике. Вычитание же из нуля единицы является обратной операцией: вычитание единицы можно рассматривать и как прибавление к нулю минус единицы, по правилам же k–разрядной арифметики «минус единица» — это дополнение единицы до 2^k, т. е. число, которое при сложении с единицей даст 2^k. Очевидно, что таковым является число $2^k - 1$, состоящее из k двоичных единиц, значит, в беззнаковом представлении $0 + (-1) \equiv 2^k - 1$, что является максимально представимым числом[1].

В знаковом k–разрядном представлении максимально допустимым является число $2^{k-1} - 1$ ($k - 1$ двоичная единица). При увеличении такого числа на 1 мы получаем 2^{k-1}, или единицу в знаковом бите и нули в остальных, что соответствует отрицательному числу с максимально возможным модулем. Вычитание единицы из такого

[1] Фактически операции сложения, вычитания и умножения в k-разрядном беззнаковом представлении соответствуют математическому понятию «арифметические операции по модулю 2^k» (сравнение по модулю 2^k).

числа можно проводить по правилам обычной двоичной арифметики: $1\underbrace{0\ldots0}_{k-1} - 1 = 0\underbrace{1\ldots1}_{k-1} = 2^k - 1$.

Достоинством рассмотренных форматов представления целых чисел является простота реализации алгоритмов арифметических операций (например, вычитание, благодаря использованию дополнительного кода для представления отрицательных чисел, сводится к сложению).

2.1.4. Особенности реализации арифметических операций в конечном числе разрядов

Целочисленная арифметика в ограниченном числе разрядов несколько отличается от обычной. При выполнении арифметических действий в целочисленной k-разрядной арифметике возможно возникновение следующих ситуаций, незнание которых может привести к неверному результату при выполнении верных алгоритмов:

- старшие (левые) цифры результата, выходящие за отведенное количество разрядов, оказываются утерянными;
- при сложении или умножении двух положительных чисел, имеющих знаковое представление, можно получить отрицательное число, если в результате сложения (умножения) в левом знаковом бите окажется единица.

Пример 5. Выполним сложение 100_{10} и 51_{10} в знаковом восьмиразрядном представлении. В этом представлении числа имеют следующий вид: $100_{10} = 0110\ 0100_2$ и $51_{10} = 0011\ 0011_2$. При сложении этих чисел получим $1001\ 0111_2$. Самая левая единица (знаковый разряд) указывает на то, что в 8 разрядах записано отрицательное число.

Так как все отрицательные числа в машине представляются дополнительным кодом, то для восстановления десятичного значения этого отрицательного числа надо воспользоваться алгоритмом получения исходного числа по его дополнительному коду (см. пример 4). В результате получим число -105.

Таким образом, в восьмиразрядной знаковой арифметике $100 + 51 = -105$, т. е. при сложении двух положительных чисел мы получили отрицательное число! □

Вопросы и задания

1. Обоснуйте целесообразность представления особым образом в компьютере целых чисел.
2. Приведите пример умножения в ограниченном числе разрядов двух положительных чисел, в результате которого получается отрицательное число.
3. Перечислите и объясните все ошибки, которые могут возникать при выполнении арифметических операций над целыми числами в компьютерной арифметике в ограниченном числе разрядов.
4. Покажите, каким образом использование дополнительного кода позволяет заменить операцию вычитания операцией сложения.
5. В восьмиразрядной ячейке запишите дополнительные коды следующих двоичных чисел:
   а) −1010; б) −1001; в) −11; г) −11011.
6. Можно ли по виду дополнительного кода числа сказать, четно оно или нечетно?
7. Найдите десятичные эквиваленты отрицательных чисел, записанных в дополнительном коде:
   а) 11000100; б) 11111001.
8. Какие из чисел 43_{16}, 101010_2, 129_{10} и -135_{10} можно сохранить в одном байте (в 8 разрядах)?
9. Получите 16-разрядное представление следующих чисел:
   а) 25; б) −610.
10. Для чисел $A = 1110_2$, $B = 1101_2$ выполните следующие операции: $A + B$; $A - B$; $B - A$; $-A - A$; $-B - B$; $-A - B$ (в восьмиразрядном знаковом представлении).
11. Вычислите с помощью Инженерного калькулятора (стандартное приложение Windows) следующие выражения:
    а) $111011101_2 - 1101110110_2$;
    б) $1101101001_2 - 11000100100_2$.
    Как вы можете объяснить полученные результаты?

§ 2.2. Представление вещественных чисел

Вещественные числа в компьютере хранятся в *формате с плавающей запятой*, который опирается на нормализованную форму записи чисел.

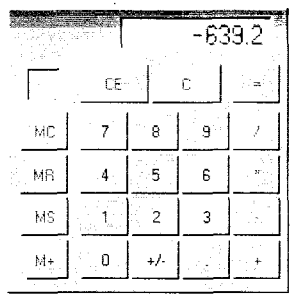
2.2.1. Нормализованная запись числа

Если при представлении целых чисел в компьютере ограничением может служить лишь величина записываемого числа, то при записи вещественного числа речь в первую очередь идет о точности его представления, т. е. о количестве значащих цифр, которые удается сохранить в ограниченном числе разрядов.

Пример 6. Допустим, мы имеем калькулятор, в котором на экране дисплея для вывода чисел есть только 10 знакомест (включая знак числа и запятую между целой и дробной частями десятичного числа). Если нам необходимо работать с числами

$$-6392000000; \quad -639{,}2; \quad -0{,}0000006392,$$

то на дисплее нашего калькулятора отобразить удастся лишь второе из них (первое число занимает 11 знакомест, второе — 6 знакомест, третье — 13 знакомест).



□

Эта задача может быть решена, если числа представить несколько иначе. Забегая вперед, скажем, что для примера 6 искомый способ записи чисел в калькуляторе таков[1]:

$$-6.392\text{E}{+}9 \quad -6.392\text{E}{+}02 \quad -6.392\text{E}{-}07,$$

где знак «E» читается как «умножить на десять в степени». Такая запись отражает *экспоненциальную* форму записи чисел.

Определение 3. Любое число a в *экспоненциальной форме* представляется в виде $a = \pm m \times P^q$, где P — основание системы счисления, m называется *мантиссой* числа, q — *порядком* числа.

---

[1] В калькуляторе, как и во многих компьютерных программах, вместо запятой для отделения дробной части от целой используется точка. Это соответствует западной традиции записи P-ичных чисел.

Пример 7. Длину отрезка, равного 47,8 см, в экспоненциальной форме записи можно представить так:

1) 478×10^{-1} см = 478 мм;
2) $4{,}78 \times 10^{1}$ см = 4,78 дм;
3) $47{,}8 \times 10^{0}$ см = 47,8 см;
4) $0{,}478 \times 10^{2}$ см = 0,478 м. □

Из этого примера видно, что длину одного и того же отрезка можно записать с использованием различных экспоненциальных форм. Эта неоднозначность записи может приводить в определенных случаях к неудобству. Из курса алгебры известно, что если P фиксировано и $1/P \le m < 1$, то представление числа в экспоненциальной форме единственно. Такая форма экспоненциального представления называется *нормализованной формой* и используется в компьютере для однозначности представления вещественных чисел.

Определение 4. *Нормализованная* запись отличного от нуля вещественного числа — это запись вида $a = \pm m \times P^q$, где q — целое число (положительное, отрицательное, или ноль), а m — правильная P–ичная дробь, у которой первая цифра после запятой не равна нулю, т. е. $1/P \le m < 1$.

Заметим, что число ноль не может быть записано в нормализованной форме так, как она была определена. Поэтому относительно нормализованной записи нуля приходится прибегать к особым соглашениям. Условимся, что *запись нуля является нормализованной, если и мантисса, и порядок равны нулю.*

В нормализованной форме все числа записываются одинаково в том смысле, что запятая у них ставится в одном и том же месте — перед первой (самой левой) значащей цифрой мантиссы. Заметим, что в двоичной системе счисления первая цифра мантиссы нормализованного числа всегда равна 1 (за исключением числа ноль). Величина же числа (т. е. ее порядок) указывается отдельно, с помощью соответствующей степени основания системы счисления, в которой это число было записано изначально. Количество цифр в мантиссе может оказаться меньше, чем число значащих цифр в исходном числе. Часто в нормализованной записи мантисса P-ичного числа записывается в P–ичной системе счисления, а порядок и само число P — в десятичной.

Пример 8. Приведем примеры нормализации чисел:

1) $0 = 0{,}0 \times 10^0$ (возможная нормализация нуля);
2) $3{,}1415926 = 0{,}31415926 \times 10^1$ (количество значащих цифр не изменилось);
3) $1000 = 0{,}1 \times 10^4$ (количество значащих цифр уменьшилось с четырех до одной);
4) $0{,}123456789 = 0{,}123456789 \times 10^0$ (запятую передвигать не нужно);
5) $0{,}0000107_8 = 0{,}107_8 \times 8^{-4}$ (количество значащих цифр уменьшилось с семи до трех);
6) $1000{,}0001_2 = 0{,}10000001_2 \times 2^4$ (количество значащих цифр уменьшить невозможно);
7) $AB{,}CDEF_{16} = 0{,}ABCDEF_{16} \times 16^2$ (количество значащих цифр уменьшить невозможно). □

При записи нормализованного числа в компьютере или калькуляторе для записи мантиссы и порядка отводится заранее фиксированное количество разрядов.

В компьютерном представлении вещественных чисел максимально допустимое количество цифр в мантиссе определяет точность, с которой может быть представлено число.

Поясним это на примере школьного калькулятора, который производит вычисления в десятичной системе счисления. Пусть это будет калькулятор с десятью знакоместами на дисплее, мантисса в нем имеет четыре цифры, порядок — две. В отличие от стандартной нормализации, у калькуляторов первая значащая цифра, с которой и начинается мантисса, изображается перед точкой. Порядок при этом соответственно уменьшается на единицу. Такая форма записи нормализованных чисел позволяет экономить одно знакоместо, так как вместо нуля в целой части мы помещаем значащую цифру.

Пример 9

1) Число 248,53786 в калькуляторе превращается в +2.485E+02. Переводя последнее число в привычное представление с фиксированной запятой, получим +248,5.

2) Число −2485378600,0 в калькуляторе превращается в −2.485E+09. Переводя последнее число в привычное представление с фиксированной запятой, получим −2485000000.
3) Число 0,00024853786 в калькуляторе имеет вид +2.485E−04, т. е. равно числу 0,0002485. □

Следовательно, всякое десятичное число, состоящее не более чем из четырех значащих цифр в нормализованном виде, можно представить в таком калькуляторе точно, а остальные числа — лишь приближенно.

В первом пункте примера 9 погрешность калькуляторного представления исходного числа составила 248,53786 − 248,5 = 0,03786. Во втором пункте погрешность равна 2485378600 − 2485000000 = 378600, а в третьем — 0,00024853786 − 0,0002485 = 0,00000003786. Если бы мантисса калькулятора имела больше цифр, погрешность была бы меньше.

Определение 5. Модуль разности между значением числа x и неким его представлением x^* (компьютерным, калькуляторным) называется *абсолютной погрешностью* представления x.

Несмотря на то, что в абсолютном исчислении погрешность может быть значительно больше 1, относительно величины самого числа ее порядок остается неизменным. Относительная погрешность представления чисел в примере 9 равна:

0,03786/248,53786 = 0,00015233...
378600/2485378600 = 0,00015233...
0,00000003786/0,00024853786 = 0,00015233...

Определение 6. *Относительной погрешностью* представления x называют величину $\left|\dfrac{x - x^*}{x}\right|$.

Вопрос. *Как вы думаете, что нам дает знание величин абсолютной и относительной погрешности при решении реальных задач на компьютере?*

Ответ. Абсолютная погрешность говорит о том, *на сколько* полученный результат (например, результат представления числа в компьютере) отличается от истинного результата (в нашем примере — от самого числа).

При решении реальных задач знание этой величины позволяет оценивать, насколько достоверный результат был получен. Если его точность нас не удовлетворяет, то следует выбрать другой (более точный) способ представления чисел.

Относительная же погрешность показывает, сколько верных старших значащих цифр содержит результат. В примере 9 относительная погрешность представления разных по величине чисел равна 0,00015233... Такая относительная погрешность означает, что мы имеем три безусловно верные значащие цифры результата. Значение относительной погрешности непосредственно связано с количеством разрядов, отводимых для представления мантиссы нормализованного числа.

На практике обычно известна относительная погрешность представления чисел, так как разрядность мантиссы фиксирована. Следовательно, можно предположить, сколько верных цифр содержит результат. Если из каких-то априорных соображений известно значение вычисляемой величины, то можно оценить абсолютную погрешность результата. □

В компьютерной записи вещественных чисел с плавающей запятой количество цифр, отводимых под запись порядка, определяет, насколько большие и насколько маленькие положительные числа могут быть представлены.

Покажем это опять на примере школьного калькулятора. В формате нашего калькулятора порядок имеет две цифры, и наибольшее число, которое может быть в нем представлено, — это +9.999E+99. Если бы нам пришлось записать это число в формате с фиксированной запятой, оно имело бы ровно 100 цифр до запятой (четыре девятки и 96 нулей), т. е. это и в самом деле очень большое число.

А самое маленькое положительное число, которое можно ввести в нашем калькуляторе, — это +1.000E–99. В формате с фиксированной запятой оно имеет 99 десятичных знаков после запятой, а именно 98 нулей и единицу. Это очень маленькое число. Таким образом, выражая порядок лишь двумя десятичными цифрами, можно записывать числа из очень широкого диапазона.

2) Число −2485378600,0 в калькуляторе превращается в −2.485E+09. Переводя последнее число в привычное представление с фиксированной запятой, получим −2485000000.
3) Число 0,00024853786 в калькуляторе имеет вид +2.485E−04, т. е. равно числу 0,0002485. □

Следовательно, всякое десятичное число, состоящее не более чем из четырех значащих цифр в нормализованном виде, можно представить в таком калькуляторе точно, а остальные числа — лишь приближенно.

В первом пункте примера 9 погрешность калькуляторного представления исходного числа составила 248,53786 − 248,5 = 0,03786. Во втором пункте погрешность равна 2485378600 − 2485000000 = 378600, а в третьем — 0,00024853786 − 0,0002485 = 0,00000003786. Если бы мантисса калькулятора имела больше цифр, погрешность была бы меньше.

Определение 5. Модуль разности между значением числа x и неким его представлением x^* (компьютерным, калькуляторным) называется *абсолютной погрешностью* представления x.

Несмотря на то, что в абсолютном исчислении погрешность может быть значительно больше 1, относительно величины самого числа ее порядок остается неизменным. Относительная погрешность представления чисел в примере 9 равна:

0,03786/248,53786 = 0,00015233...
378600/2485378600 = 0,00015233...
0,00000003786/0,00024853786 = 0,00015233...

Определение 6. *Относительной погрешностью* представления x называют величину $\left|\dfrac{x - x^*}{x}\right|$.

Вопрос. *Как вы думаете, что нам дает знание величин абсолютной и относительной погрешности при решении реальных задач на компьютере?*

Ответ. Абсолютная погрешность говорит о том, *на сколько* полученный результат (например, результат представления числа в компьютере) отличается от истинного результата (в нашем примере — от самого числа).

При решении реальных задач знание этой величины позволяет оценивать, насколько достоверный результат был получен. Если его точность нас не удовлетворяет, то следует выбрать другой (более точный) способ представления чисел.

Относительная же погрешность показывает, сколько верных старших значащих цифр содержит результат. В примере 9 относительная погрешность представления разных по величине чисел равна 0,00015233... Такая относительная погрешность означает, что мы имеем три безусловно верные значащие цифры результата. Значение относительной погрешности непосредственно связано с количеством разрядов, отводимых для представления мантиссы нормализованного числа.

На практике обычно известна относительная погрешность представления чисел, так как разрядность мантиссы фиксирована. Следовательно, можно предположить, сколько верных цифр содержит результат. Если из каких-то априорных соображений известно значение вычисляемой величины, то можно оценить абсолютную погрешность результата. □

В компьютерной записи вещественных чисел с плавающей запятой количество цифр, отводимых под запись порядка, определяет, насколько большие и насколько маленькие положительные числа могут быть представлены.

Покажем это опять на примере школьного калькулятора. В формате нашего калькулятора порядок имеет две цифры, и наибольшее число, которое может быть в нем представлено, — это +9.999E+99. Если бы нам пришлось записать это число в формате с фиксированной запятой, оно имело бы ровно 100 цифр до запятой (четыре девятки и 96 нулей), т. е. это и в самом деле очень большое число.

А самое маленькое положительное число, которое можно ввести в нашем калькуляторе, — это +1.000E–99. В формате с фиксированной запятой оно имеет 99 десятичных знаков после запятой, а именно 98 нулей и единицу. Это очень маленькое число. Таким образом, выражая порядок лишь двумя десятичными цифрами, можно записывать числа из очень широкого диапазона.

Широкий диапазон представления чисел с плавающей запятой необходим при решении научных и инженерных задач. Такое представление чисел не только позволяет сохранять в разрядной сетке большое количество значащих цифр и тем самым повышать точность вычислений, но также упрощает действия над порядками и мантиссами.

2.2.2. Представление вещественных чисел в формате с плавающей запятой

Как и для целых чисел, при представлении вещественных чисел в компьютере используется чаще всего двоичная система счисления, следовательно, предварительно десятичное число должно быть переведено в двоичную систему, а уж затем представлено в нормализованной форме с $P = 2$.

При представлении нормализованных чисел часть разрядов ячейки отводится для записи порядка числа, остальные разряды — для записи мантиссы. По одному разряду в каждой группе отводится для изображения знака порядка и знака мантиссы. О таком представлении говорят, что число записано в формате *с плавающей запятой*.

Например, можно представить себе такое распределение разрядов ячейки памяти:

s_q	s_m	b_k	b_{k-1}	...	b_2	b_1	a_1	a_2	...	a_{n-1}	a_n

Первые два разряда служат для представления знаков порядка (s_q) и мантиссы (s_m) соответственно. Следующие k разрядов используются для представления абсолютной величины порядка числа (q), остальные n разрядов — для представления абсолютной величины мантиссы. В каждом разряде ячейки может храниться одно из двух значений: 0 или 1.

Тогда изображенному на схеме состоянию ячейки соответствует число

$$(-1)^{s_m} \cdot 2^q \cdot \left(\frac{a_1}{2} + \frac{a_2}{2^2} + \ldots + \frac{a_n}{2^n} \right),$$

где $q = (-1)^{s_q} (b_k \cdot 2^{k-1} + b_{k-1} \cdot 2^{k-2} + \ldots + b_2 \cdot 2 + b_1)$.

При такой системе записи наибольшее по абсолютной величине число, которое может быть представлено в машине, равно $2^{2^k-1} \cdot (1 - 2^{-n})$, а наименьшее по абсолютной величине число, отличное от нуля, равно $2^{-(2^k-1)} \cdot 2^{-n} = 2^{-2^k-n+1}$.

Вещественных чисел, точно представимых в компьютере, конечное число. Остальные числа либо приближаются представимыми, либо оказываются непредставимыми. Последнее относится к слишком большим и к слишком маленьким вещественным числам.

2.2.3. Выполнение арифметических операций над вещественными числами

Использование в компьютере представления чисел в формате с плавающей запятой усложняет выполнение арифметических операций.

При *сложении* и *вычитании* чисел сначала производится подготовительная операция, называемая *выравниванием порядков*. Она состоит в том, что мантисса числа с меньшим порядком сдвигается в своей ячейке вправо на количество разрядов, равное разности порядков данных чисел. После этой операции одноименные разряды мантисс оказываются расположенными в одноименных разрядах обеих ячеек, и теперь уже сложение или вычитание мантисс выполняется достаточно просто, так же как над числами с фиксированной запятой.

Пусть $a = \pm 0, m_a \times 2^{q_a}$, $b = \pm 0, m_b \times 2^{q_b}$ — два нормализованных двоичных числа, и $q_a \geq q_b$. Результатом их сложения или вычитания будет являться следующее выражение: $c = (0, m_a \pm 0, m_b \times 2^{q_b - q_a}) \times 2^{q_a}$.

После операций над порядками и мантиссами мы получаем порядок и мантиссу результата, но последняя, вообще говоря, может не удовлетворять ограничениям, накладываемым на мантиссы нормализованных чисел. Так как от результата арифметических операций в компьютере требуется, чтобы он также был нормализованным числом, необходимо дополнительное преобразование результата — *нормализация*. В зависимости от величины получившейся мантиссы результата, она сдвигается вправо или влево так, чтобы ее первая значащая

цифра попала в первый разряд после запятой. Одновременно порядок результата увеличивается или уменьшается на число, равное величине сдвига.

Заметим, что над мантиссами в арифметическом устройстве могут выполняться все четыре арифметических действия, а также операции сдвига, тогда как над порядками производятся только действия сложения и вычитания. Отрицательные порядки можно записывать в дополнительном коде для того, чтобы операцию вычитания свести к операции сложения.

В ряде случаев, даже если некоторые два числа были представлены в формате с плавающей запятой абсолютно точно, результат выполнения над ними арифметических операций часто может содержать погрешность, а иногда может быть заведомо неверным.

Поясним более подробно особенности выполнения арифметических операций над вещественными числами на примерах. При этом будем считать, что в записи вещественного числа с плавающей запятой один разряд отводится под десятичный порядок и пять разрядов — под десятичную мантиссу.

Пример 10. Предположим, что требуется сложить следующие числа: $0{,}23619 \times 10^2$ и $0{,}71824 \times 10^{-1}$.

Так как порядки у чисел различны, то перед сложением производится выравнивание порядков. Число с меньшим порядком преобразуется в число с порядком, равным порядку другого слагаемого (меньший порядок «приводится» к большему). В данном случае второе слагаемое будет преобразовано к виду $0{,}00071824 \times 10^2$, после чего выполняется сложение.

$$\begin{array}{rl} & 10^2 \times 0{,}23619 \\ + & 10^2 \times 0{,}00071824 \\ \hline & 10^2 \times 0{,}23690824 \end{array}$$

Результат получили с бóльшим числом разрядов, чем вмещает ячейка, поэтому он округляется и записывается в памяти в виде $0{,}23691 \times 10^2$. □

Пример 11. Выполним сложение двух вещественных чисел:

$$0{,}23619 \times 10^8 \text{ и } 0{,}91824 \times 10^8.$$

Так как порядки у этих чисел одинаковы, то производить операцию выравнивания порядков не требуется. Операция сложения сводится к сложению мантисс.

$$\begin{array}{r} 10^8 \times 0{,}23619 \\ + \quad 10^8 \times 0{,}91824 \\ \hline 10^8 \times 1{,}15443 \end{array}$$

Результат — ненормализованное число $1{,}15443 \times 10^8$. Требуется выполнить нормализацию путем сдвига мантиссы вправо на один разряд, а затем округление результата, так как в мантиссе будет уже шесть цифр, а в ячейке памяти под мантиссу отведено их только пять.

Ответ: $0{,}11544 \times 10^9$. □

Пример 12. Выполним сложение двух вещественных чисел, одно из которых достаточно большое по сравнению со вторым: $0{,}23619 \times 10^3$ и $0{,}91824 \times 10^{-3}$.

Так как порядки у этих чисел различны, то требуется произвести предварительную операцию их выравнивания. После выполнения выравнивания порядков складываться будут следующие числа:

$$\begin{array}{r} 10^3 \times 0{,}23619 \\ + \quad 10^3 \times 0{,}00000091824 \\ \hline 10^3 \times 0{,}23619091824 \end{array}$$

После сложения в мантиссе оказалось более пяти значащих цифр и при записи в ячейку памяти произойдет округление результата до $0{,}23619 \times 10^3$. Получившееся число равно первому слагаемому, т. е. при выравнивании порядков все значащие цифры мантиссы второго слагаемого потеряны. Таким образом, мы получили результат, невозможный с точки зрения обычной математики: $a + b = a$ при $b > 0$. Но в компьютерной арифметике с ограниченным числом разрядов такой результат возможен, и об этом необходимо помнить при составлении алгоритмов решения задач. □

В вещественной компьютерной арифметике с ограниченным числом разрядов $1 + \varepsilon = 1$ при $0 < \varepsilon < 2^{-n}$, где n — количество разрядов, отводимых для представления мантиссы вещественного числа.

При *умножении* двух целых чисел с плавающей запятой их порядки необходимо просто сложить, а мантиссы — перемножить (предварительное выравнивание не производится). При *делении* из порядка делимого надо вычесть порядок делителя, а мантиссу делимого разделить на мантиссу делителя.

Результатами выполнения операций умножения и деления нормализованных чисел a и b в арифметике с ограниченным числом разрядов будут:

$$d = a \cdot b = (0, m_a \cdot 0, m_b) \times 2^{q_a + q_b},$$

$$f = a : b = (0, m_a : 0, m_b) \times 2^{q_a - q_b}.$$

Пример 13. Выполним умножение двух вещественных чисел: $0{,}23000 \times 10^3$ и $0{,}95000 \times 10^7$.

При умножении двух вещественных чисел в представлении с плавающей запятой порядки складываются, а мантиссы перемножаются. В результате получим: $0{,}21850 \times 10^{10}$. Это число не умещается в отведенный формат — в нашем формате под порядок отводится один разряд, а в получившемся числе порядок содержит две цифры. Выполнение операции умножения над этими числами приведет к прекращению выполнения программы в связи с ошибкой «переполнение порядка». □

Пример 14. Выполним деление двух вещественных чисел: $0{,}92000 \times 10^4$ и $0{,}30000 \times 10^7$.

При делении вещественных чисел в представлении с плавающей запятой порядки вычитаются, а мантиссы делятся одна на другую. В нашем примере при делении мантисс мы имеем бесконечную периодическую дробь 0,92:0,3 = 3,0(6). Следовательно, при записи мантиссы результата произойдет ее округление. После нормализации результат будет иметь вид: $0{,}30667 \times 10^{-2}$. □

2.2.4. Особенности реализации вещественной компьютерной арифметики

Когда говорят о точности представления вещественных чисел в компьютере, надо помнить следующее: десятичное число, имеющее даже всего одну значащую цифру после запятой, вообще говоря, невозможно представить точно в формате с плавающей запятой. Объясняется это тем, что конечные десятичные дроби часто оказываются бесконечными периодическими двоичными дробями. Так $0{,}1_{10} = 0{,}0(0011)_2$, а значит, и в нормализованном виде такое двоичное число будет иметь бесконечную мантиссу и не может быть представлено точно. При записи подобной мантиссы в ячейку компьютера число не

усекается, а *округляется*. Если под мантиссу отведено n разрядов и $(n+1)$-я значащая цифра двоичной нормализованной мантиссы равна 0, то цифры, начиная с $(n+1)$-й, просто отбрасываются, если же $(n+1)$-я цифра равна единице, то к целому числу, составленному из первых n значащих цифр мантиссы, прибавляется единица.

Пример 15. Рассмотрим, как будет выглядеть запись мантиссы (m) числа $a = 0{,}1_{10}$ при двоичной нормализации для различного количества бит (n), отведенных под мантиссу.

$a = 0{,}1_{10} = 0{,}0(0011)_2 = 0{,}11(0011)_2 \times 2^{-3}$, т. е. мантисса в нормализованном числе есть $0{,}11(0011)_2$.
При $n = 10$ $m = 1100110011$ (остальные цифры мантиссы отброшены в результате округления).
При $n = 12$ $m = 110011001101$ (последняя цифра изменилась с 0 на 1 при округлении).
При $n = 13$ $m = 1100110011010$ (две последние цифры изменились при прибавлении 1). □

Опишем ситуации, приводящие к неточности вычислений, которые могут возникнуть при операциях сложения и вычитания в вещественной компьютерной арифметике.

1. Потеря значащих цифр мантиссы у меньшего из чисел при выравнивании порядков

При сложении и вычитании вещественных чисел в худшем случае утерянными оказываются все значащие цифры меньшего числа, и $a \pm b \equiv a$, что является абсурдным с точки зрения математики, но возможным в компьютерной арифметике с ограниченным числом разрядов (см. пример 12).

2. Потеря крайней справа значащей цифры результата при сложении или вычитании

При сложении и вычитании двух чисел количество значащих цифр может увеличиться лишь на одну, это влияет на точность, но не на правильность результата.

Пример 16. При сложении пятнадцатиразрядных мантисс

$$0{,}111110000011111_2 + 0{,}100100100100100_2 =$$
$$= 1{,}100010101000011_2$$

количество значащих цифр стало равным 16, и после округления и нормализации результат будет выглядеть так: $0{,}110001010100010 \times 2^1$. □

Гораздо хуже обстоит дело при вычитании близких по модулю и имеющих одинаковый знак чисел (или сложении чисел разного знака). В этом случае достоверной может остаться всего одна значащая цифра, а остальные цифры нормализованной мантиссы (скорее всего нули) окажутся недостоверными.

Пример 17. Пусть требуется вычислить разность чисел $0{,}(0011)_2$ и $0{,}001100110011_2$.

Произведем вначале вычитание «на бумаге». Представим первую дробь (она периодическая) в виде $0{,}001100110011(0011)_2$. Перед периодом выписали 12 цифр, так как вторая дробь содержит 12 значащих цифр. Произведем вычитание и получим: $0{,}000000000000(0011)_2 = 0{,}(1100)_2 \times 2^{-14}$.

Переведем периодическую дробь в десятичную систему счисления, для этого воспользуемся формулой суммы бесконечно убывающей геометрической прогрессии с $q = \frac{1}{2}$ и $b = \frac{1100_2}{2^4}$. Получим: $0{,}(0011)_2 - 0{,}001100110011_2 = 0{,}(1100)_2 \times 2^{-14} = 0{,}8 \times 2^{-14}$.

При выполнении этой же операции вычитания в компьютере при условии, что под мантиссу отведено 12 разрядов, получим: $0{,}110011001101_2 \times 2^{-2} - 0{,}1100110011_2 \times 2^{-2} = 0{,}1_2 \times 2^{-13} = 1 \times 2^{-14}$,

т. е. уже старшая значащая цифра результата оказалась неверной. □

3. Выход за границу допустимого диапазона значений при нормализации результата

Данная ситуация возникает в случае, когда порядок результата оказывается либо больше максимально возможного значения, либо меньше минимально возможного. Такую ситуацию различные компиляторы и операционные системы обрабатывают по-разному, но чаще всего выполнение программы прерывается с сообщением об ошибке «арифметическое переполнение».

Пример 18. Выполним сложение

$$0{,}1_2 \times 2^{127} + 0{,}1_2 \times 2^{127} = 0{,}1_2 \times 2^{128},$$

и если максимальный представимый порядок равен 127, результат оказывается не представимым. □

Опишем ситуации, приводящие к неточности вычислений при выполнении умножения/деления вещественных чисел.

4. Получение «не представимого»

Данная ситуация соответствует описанному выше «арифметическому переполнению»; однако в 80-разрядном представлении вещественных чисел (а именно он является основным в современных персональных компьютерах) диапазон допустимых порядков достаточно велик, чтобы производить практические любые вычислительные работы, и возникновение подобной ошибки скорее всего означает, что программа составлена неверно.

5. Потеря младших значащих цифр результата

Во-первых, при перемножении двух n-значных мантисс может получиться число, состоящее из $2n$ значащих цифр, только половина из которых будет сохранена в результате. При операции деления количество цифр в частном может оказаться бесконечным и лишь первые n из них будут сохранены.

Во-вторых, при операции умножения возможна потеря n-й младшей значащей цифры результата при сдвиге мантиссы на один разряд влево. Самый правый разряд мантиссы при этом заполняется нулем, а не очередной значащей цифрой результата перемножения мантисс, с одновременным уменьшением порядка результата на единицу. Для операции же деления может понадобиться сдвиг вправо вместе с увеличением порядка результата на единицу.

Таким образом, у вещественной арифметики есть несколько потенциально опасных особенностей. Все они имеют общее происхождение, а именно, тот факт, что мантисса и порядок в представлении с плавающей запятой занимают фиксированное число разрядов.

Подведем итог всему сказанному о компьютерной вещественной арифметике:

а) уже на стадии записи чисел в компьютер возникают ошибки округления, которые при выполнении арифметических действий нарастают;

б) наличие погрешностей округления приводит к следующему правилу программирования: неразумно сравнивать в программе два вещественных числа на точное равенство (вместо сравнения на равенство правильнее требовать, чтобы модуль разности сравниваемых чисел не превосходил некоторого числа ε, соответствующего абсолютной погрешности представления);

в) в результате вычитания возникают недостоверные значащие цифры, которые могут привести к серьезной потере точности или получению неправильного результата;

г) прибавление или вычитание малого числа может никак не сказаться на результате;

д) получение очень больших чисел может вызвать переполнение порядка, а очень малых — отрицательное переполнение, или исчезновение числа (превращение в нуль), это может привести к аварийному завершению программы.

Вопросы и задания

1. Запишите следующие десятичные числа в нормализованном виде:

   а) 217,934; в) 10,0101; б) 75321; г) 0,00200450.

2. Приведите к нормализованному виду следующие числа, используя в качестве P основания их систем счисления:

   а) $-0{,}000001011101_2$;

   б) 9876543210_{10};

   в) $123456789{,}\text{ABCD}_{16}$.

3. Сравните следующие числа:

   а) $318{,}4785 \times 10^9$ и $3{,}184785 \times 10^{11}$;

   б) $218{,}4785 \times 10^{-3}$ и $21847{,}85 \times 10^{-4}$;

   в) $0{,}1101_2 \times 2^2$ и $101_2 \times 2^{-2}$.

4. Сравните диапазон представления чисел с плавающей запятой в 32-разрядном формате (24 разряда для мантиссы и 6 разрядов для модуля порядка) с диапазоном представления чисел с фиксированной запятой в том же формате.

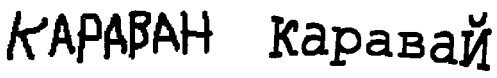
5. Каковы преимущества компьютерного представления чисел с плавающей запятой по сравнению с их представлением с фиксированной запятой, которое мы чаще всего используем в повседневной жизни?

6. Произведите следующие арифметические действия над десятичными нормализованными числами согласно правилам вещественной компьютерной арифметики (в мантиссе должно быть сохранено 6 значащих цифр)[1]:
   а) $0{,}397621 \times 10^3 + 0{,}237900 \times 10^1$;
   б) $0{,}982563 \times 10^2 - 0{,}745623 \times 10^2$;
   в) $0{,}235001 \times 10^2 \cdot 0{,}850000 \times 10^3$;
   г) $0{,}117800 \times 10^2 : 0{,}235600 \times 10^3$.
7. Выполните действие над машинными кодами чисел с плавающей запятой в 32-разрядном формате (см. задание 4): $X = A + B$, где $A = 125{,}75$ и $B = -50$.
8. Перечислите и объясните все ошибки, которые могут возникать при арифметических операциях с нормализованными числами в ограниченном числе разрядов.
9. Измените порядок приведенных ниже действий так, чтобы не происходило переполнения порядка в десятичном калькуляторе с двумя разрядами под порядок:
   3.0E+60 · 4.0E+50 · 1.0E−30.
10. Подберите такие значения вещественных чисел a, b и c, чтобы при вычислениях на описанном выше школьном калькуляторе значение результата зависело от порядка суммирования, т. е. $a + b + c \neq c + b + a$.

§ 2.3. Представление текстовой информации

Всякий текст состоит из *символов* — букв, цифр, знаков препинания и т. д., которые человек различает по начертанию. Однако для компьютерного представления текстовой информации такой метод неудобен, а для компьютерной обработки текстов и вовсе неприемлем.

Пример 19. Пусть в компьютере сохранены изображения двух похожих слов («караван» и «каравай»):



[1] При выполнении этого задания следует нормализовать мантиссу результата соответствующего арифметического действия, а затем округлить ее.

Чтобы расположить эти слова в алфавитном порядке, компьютер должен проанализировать, из каких букв состоят слова и в какой последовательности расположены буквы в словах. Но одинаковые буквы изображены на двух картинках по-разному. Эта задача, как и задача нахождения общей части этих двух слов, неразрешима в таком представлении. Действительно, общая часть этих слов должна быть изображением текста «карава». Но поскольку исходные слова записаны по-разному, то соответствующие части изображений тоже выглядят по-разному. А это значит, что если мы выберем изображение текста «карава» из первой картинки, то оно не будет совпадать с соответствующей частью второго изображения. □

Поскольку текст изначально дискретен — он состоит из отдельных символов, — для компьютерного представления текстовой информации используется другой способ: все символы *кодируются* числами, и текст представляется в виде набора чисел — *кодов символов*, его составляющих. При выводе текста на экран монитора или принтер необходимо восстановить изображения всех символов, составляющих данный текст. Для этого используются так называемые *кодовые таблицы символов*, в которых каждому коду символа ставится в соответствие изображение символа.

Все кодовые таблицы, используемые в любых компьютерах и любых операционных системах, подчиняются международным стандартам кодирования символов. На заре компьютерной эпохи, когда США были абсолютным лидером в этой области, стандарты разрабатывались Американским национальным институтом стандартизации (ANSI); впоследствии для разработки и принятия компьютерных стандартов была создана Международная организация стандартизации (ISO).

В программировании наиболее часто используются однобайтовые кодировки: в них код каждого символа занимает ровно 1 байт, или 8 бит. При этом общее количество различаемых символов составляет $2^8 = 256$, а коды символов имеют значения от 0 до 255.

Определение 7. *Информационным объемом* блока информации называется количество бит, байт или производных

единиц (килобайт, мегабайт и т. д.), необходимых для записи этого блока путем заранее оговоренного способа двоичного кодирования.

Задание. *Оцените в байтах объем текстовой информации в «Современном словаре иностранных слов» из 740 страниц, если на одной странице размещается в среднем 60 строк по 80 символов (включая пробелы).*

Решение. Будем считать, что при записи используется кодировка «один символ — один байт». Количество символов во всем словаре равно $80 \cdot 60 \cdot 740 = 3\,552\,000$. Следовательно, объем в байтах равен 3 552 000 байт = 3 468,75 Кбайт ≈ ≈ 3,39 Мбайт. □

Основой для компьютерных стандартов кодирования символов послужил ASCII (*American Standard Code for Information Interchange*) — американский стандартный код для обмена информацией, разработанный в 1960-х годах и применяемый в США для любых видов передачи информации, в том числе и некомпьютерных (телеграф, факсимильная связь и т. д.). В нем используется 7-битовое кодирование: общее количество символов составляет $2^7 = 128$, из них первые 32 символа — управляющие, а остальные — «изображаемые», т. е. имеющие графическое изображение. Управляющие символы должны восприниматься устройством вывода текста как команды, например:

Код	Действие	Английское название
7	Подача стандартного звукового сигнала	Beep
8	Удаление предыдущего символа	Back Space (BS)
13	Перевод строки	Line Feed (LF)
26	Признак «Конец текстового файла»	End Of File (EOF)
27	Отмена предыдущего ввода	Escape (Esc)

К изображаемым символам в ASCII относятся буквы английского алфавита (прописные и строчные), цифры, знаки препинания и арифметических операций, скобки и некоторые специальные символы. Фрагмент кодировки ASCII приведен в табл. 1.

Таблица 1

Фрагмент кодировки ASCII

Символ	Десятичный код	Двоичный код	Символ	Десятичный код	Двоичный код
Пробел	32	00100000	0	48	00110000
!	33	00100001	1	49	00110001
#	35	00100011	2	50	00110010
$	36	00100100	3	51	00110011
*	42	00101010	4	52	00110100
+	43	00101011	5	53	00110101
,	44	00101100	6	54	00110110
–	45	00101101	7	55	00110111
.	46	00101110	8	56	00111000
/	47	00101111	9	57	00111001
A	65	01000001	N	78	01001110
B	66	01000010	O	79	01001111
C	67	01000011	P	80	01010000
D	68	01000100	Q	81	01010001
E	69	01000101	R	82	01010010
F	70	01000110	S	83	01010011
G	71	01000111	T	84	01010100
H	72	01001000	U	85	01010101
I	73	01001001	V	86	01010110
J	74	01001010	W	87	01010111
K	75	01001011	X	88	01011000
L	76	01001100	Y	89	01011001
M	77	01001101	Z	90	01011010

Хотя в ASCII символы кодируются 7 битами, в памяти компьютера под каждый символ отводится ровно 1 байт, при этом код символа помещается в младшие биты, а старший бит не используется.

Главный недостаток стандарта ASCII заключается в том, что он рассчитан на передачу только английского текста. Со временем возникла необходимость кодирования и неанглийских букв. Во многих странах для этого стали разрабатывать *расширения ASCII-кодировки*, в которых применялись однобайтовые коды символов; при этом первые 128 символов кодовой таблицы совпадали с кодировкой ASCII, а остальные (со 128-го по

255-й) использовались для кодирования букв национального алфавита, символов национальной валюты и т. п. Из-за несогласованности этих разработок для многих языков было создано по нескольку вариантов кодовых таблиц (например, для русского языка их около десятка!).

Впоследствии использование кодовых таблиц было несколько упорядочено: каждой кодовой таблице было присвоено особое название и номер. Указав кодовую таблицу, автоматически выбирают и язык, которым можно пользоваться в дополнение к английскому; точнее, выбирается то, как будут интерпретироваться символы с кодами более 127.

Для русского языка наиболее распространенными являются однобайтовые кодовые таблицы CP-866, Windows-1251[1] и КОИ-8. В них первые 128 символов совпадают с ASCII-кодировкой, а русские буквы размещены во второй части таблицы, однако коды русских букв в этих кодировках различны! Сравните, например, кодировки КОИ-8 (Код Обмена Информацией 8-битовый, международное название koi-8r) и Windows-1251, вторые половины которых приведены в табл. 2 и 3 соответственно.

Таблица 2

Кодировка КОИ-8

─ 128	│ 129	┌ 130	┐ 131	└ 132	┘ 133	├ 134	┤ 135	┬ 136	┴ 137	┼ 138	▀ 139	▄ 140	▌ 141	▐ 142	▒ 143
▒ 144	▒ 145	▒ 146	⌠ 147	▪ 148	∙ 149	√ 150	≈ 151	≤ 152	≥ 153	nbsp 154	⌡ 155	° 156	² 157	· 158	÷ 159
═ 160	║ 161	╒ 162	ё 163	╓ 164	╔ 165	╕ 166	╖ 167	╗ 168	╘ 169	╙ 170	╚ 171	╛ 172	╜ 173	╝ 174	╞ 175
╟ 176	╠ 177	╡ 178	Ё 179	╢ 180	╣ 181	╤ 182	╥ 183	╦ 184	╧ 185	╨ 186	╩ 187	╪ 188	╫ 189	╬ 190	© 191
ю 192	а 193	б 194	ц 195	д 196	е 197	ф 198	г 199	х 200	и 201	й 202	к 203	л 204	м 205	н 206	о 207
п 208	я 209	р 210	с 211	т 212	у 213	ж 214	в 215	ь 216	ы 217	з 218	ш 219	э 220	щ 221	ч 222	ъ 223
Ю 224	А 225	Б 226	Ц 227	Д 228	Е 229	Ф 230	Г 231	Х 232	И 233	Й 234	К 235	Л 236	М 237	Н 238	О 239
П 240	Я 241	Р 242	С 243	Т 244	У 245	Ж 246	В 247	Ь 248	Ы 249	З 250	Ш 251	Э 252	Щ 253	Ч 254	Ъ 255

[1] Иначе эта кодировка обозначается сокращением CP-1251 (Code Page — кодовая страница).

Таблица 3

Кодировка Windows-1251

Ѓ 128	ѓ 129	‚ 130	є 131	„ 132	… 133	† 134	‡ 135	€ 136	‰ 137	Ѕ 138	‹ 139	њ 140	Ќ 141	ћ 142	Ў 143
ў 144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	ѓ 152	™ 153	ѕ 154	› 155	ј 156	ґ 157	ґ 158	ї 159
nbsp 160	Ў 161	ў 162	Ј 163	¤ 164	Ґ 165	¦ 166	§ 167	Ё 168	© 169	Є 170	« 171	¬ 172	shy 173	® 174	Ї 175
° 176	± 177	І 178	і 179	ґ 180	µ 181	¶ 182	· 183	ё 184	№ 185	є 186	» 187	ј 188	Ѕ 189	ѕ 190	ї 191
А 192	Б 193	В 194	Г 195	Д 196	Е 197	Ж 198	З 199	И 200	Й 201	К 202	Л 203	М 204	Н 205	О 206	П 207
Р 208	С 209	Т 210	У 211	Ф 212	Х 213	Ц 214	Ч 215	Ш 216	Щ 217	Ъ 218	Ы 219	Ь 220	Э 221	Ю 222	Я 223
а 224	б 225	в 226	г 227	д 228	е 229	ж 230	з 231	и 232	й 233	к 234	л 235	м 236	н 237	о 238	п 239
р 240	с 241	т 242	у 243	ф 244	х 245	ц 246	ч 247	ш 248	щ 249	ъ 250	ы 251	ь 252	э 253	ю 254	я 255

Несовпадение кодовых таблиц приводит к ряду неприятных эффектов, например, так как один и тот же текст (неанглийский) имеет различное компьютерное представление в разных кодировках, то текст, набранный в одной кодировке, будет нечитабельным в другой!

Пример 20. Вот так будет выглядеть десятичный код слова «Диск» в разных кодировках:

КОИ-8	228 201 211 203
Windows–1251	196 232 241 234
CP-866	132 168 225 170

□

Однобайтовые кодировки обладают одним серьезным ограничением: количество различных кодов символов в этих кодировках недостаточно велико, чтобы можно было пользоваться одновременно несколькими языками. Для устранения этого ограничения в 1993 году был разработан новый стандарт кодирования символов, получивший название *Unicode*, который, по замыслу его разработчиков, позволил бы использовать в текстах любые символы любых языков мира.

В Unicode на кодирование символов отводится 31 бит. Первые 128 символов (коды 0–127) совпадают с таблицей ASCII; далее размещены основные алфавиты современных языков: они полностью умещаются в первой части таблицы, их коды не превосходят 65 536 ($65\,536 = 2^{16}$). А в целом стандарт Unicode описывает алфавиты всех

известных, в том числе и «мертвых», языков; для языков, имеющих несколько алфавитов или вариантов написания (например, японский и индийский), закодированы все варианты; в кодировку Unicode внесены все математические и иные научные символьные обозначения и даже некоторые придуманные языки (например, письменности эльфов и Мордора из эпических произведений Дж. Р. Р. Толкиена). Потенциальная информационная емкость 31-битового Unicode столь велика, что сейчас используется менее одной тысячной части возможных кодов символов!

В современных компьютерах и операционных системах используется укороченная, 16-битовая версия Unicode, в которую входят все современные алфавиты; эта часть Unicode называется базовой многоязыковой страницей (***Base Multilingual Plane***, BMP). В UNIX-подобных операционных системах, где работа с Unicode-текстами невозможна из-за особенностей архитектуры, используются особые формы этого стандарта, которые называются UTF (***Unicode Transformation Form***), в них символы кодируются переменным количеством байтов. Например, в UTF-8 коды символов занимают от 1 до 6 байтов.

Вопросы и задания

1. На чем основывается возможность двоичного кодирования текстовой информации?
2. На основании чего можно утверждать, что для латинских букв применяется семибитовое кодирование?
3. Зная, что в кодировке ASCII десятичный код каждой строчной латинской буквы на 32 больше кода соответствующей прописной буквы, представьте фрагмент этой кодировочной таблицы в формате, основанном на шестнадцатеричной системе счисления:

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
						...										
2		!														
3	0															
4												K				
5												[	\	]	^	_
6	`											k				
7												{	|	}	~	

4. Используя результат выполнения предыдущего задания, декодируйте следующее сообщение, записанное в восьмибитовой кодировке:

   01010101 01110000 00100000 00100110 00100000 01000100 01101111 01110111 01101110.

5. Определите вид кодировки и декодируйте следующие сообщения:

   а) 235 207 212 197 204 216 206 201 203 207 215;
   б) 213 224 244 244 236 224 237.

6. Петя и Коля пишут друг другу электронные письма. Однажды Петя отправил Коле письмо в кодировке Windows-1251. Коля письмо получил, но прочитал его в кодировке КОИ-8. Получился бессмысленный текст, одно из предложений которого имело вид:

   кЧАЮЪ ХМТНПЛЮЖХЪ ЛНФЕР АШРЭ ОПЕДЯРЮБКЕМЮ Я ОНЛНЫЭЧ ВХЯЕК.

   Какое предложение было в исходном сообщении?

7. Во сколько раз уменьшится информационный объем страницы текста при его преобразовании из кодировки Unicode (таблица кодировки содержит 65 536 символов) в кодировку Windows-1251 (таблица кодировки содержит 256 символов)?

8. Будут ли упорядочены по алфавиту фамилии, записанные русскими буквами, если их сортировку осуществить согласно кодам символов из кодировки Windows-1251?

9. Почему в кодировке ASCII сдвиг, с помощью которого по коду прописной английской буквы можно получить код соответствующей строчной, равен 32, а не, например, 26 (в кодировках КОИ-8 и Windows-1251 это же свойство сохраняется и для русских букв)?

§ 2.4. Представление графической информации

Подавляющую часть информации об окружающем мире человек получает с помощью зрения — по оценкам ученых, доля зрительной информации составляет не менее 80% общего потока информации, воспринимаемого всеми органами чувств. Важность зрения обусловлена историческим развитием человека как биологического вида, поэтому зрительные органы человека, и особенно зрительные центры мозга, прекрасно приспособлены к обработке информации с большой скоростью и в больших

объемах. С появлением компьютеров, способных быстро обрабатывать информацию, и их совершенствованием ученые стали разрабатывать компьютерные методы хранения и обработки изображений.

2.4.1. Общие подходы к представлению в компьютере информации естественного происхождения

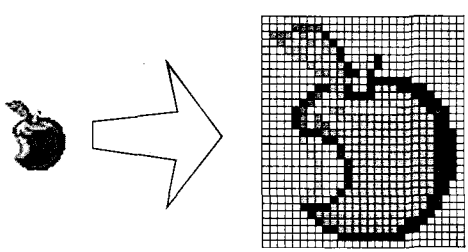
Для хранения и обработки графической и звуковой информации в компьютере требуются значительные вычислительные ресурсы (память и процессорное время), и, кроме того, обрабатываемая информации естественного происхождения должна быть представлена в специальном *компьютерном виде*. Главная проблема разработки такого представления заключается в том, что компьютер может обрабатывать и хранить только ограниченный объем информации, в то время как любые естественные сигналы — носители информации, — непрерывны (недискретны) и неограничены в пространстве и времени.

Для преобразования «естественной» информации в дискретную форму ее подвергают *дискретизации* и *квантованию*.

Определение 8. *Дискретизацией* (англ. *discretisation*) называют процедуру устранения временно́й и/или пространственной непрерывности естественных сигналов, являющихся носителями информации.

При *пространственной дискретизации* изображения его разбивают на небольшие области, в пределах которых характеристики изображения считают неизменными.

Пример 21. Пространственная дискретизация изображения.

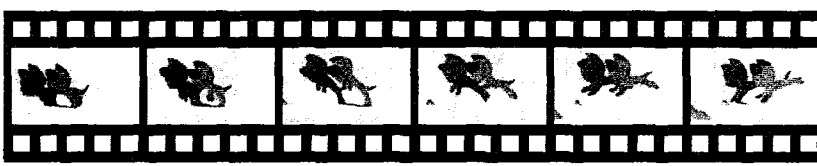


□

При *временно́й дискретизации* время разбивают на небольшие интервалы, в пределах которых характеристики природных сигналов, как и в пространственном случае, считают неизменными.

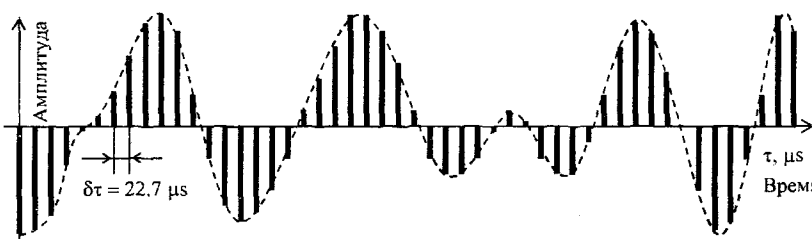
Наглядным примером временно́й дискретизации могут служить кино и телевидение. В них иллюзия подвижного изображения создается путем быстрой смены кадров. При этом сами кадры являются статическими изображениями. Компьютерное кодирование видеоинформации также основано на эффекте смены кадров, на которых изображены последовательные фазы движения.

Пример 22. Временна́я дискретизация движущегося изображения.



□

Пример 23. Временна́я дискретизация звукового сигнала.



□

Вопрос. *Зачем нужна дискретизация изображения?*

Ответ. Пространство *непрерывно*, а это означает, что в любой его области содержится бесконечное количество точек. Если мы хотим *точно сохранить изображение*, то должны запоминать информацию о каждой точке пространства (если мы какие-то точки не учитываем, то не сможем различить два изображения, которые отличаются друг от друга только в этих точках).

Поскольку точек бесконечно много, то и компьютерное представление должно было бы содержать бесконечно много информации, и для его сохранения потребовалось бы бесконечное количество памяти. А это значит, компьютеры в принципе не могли бы ни обрабатывать, ни хранить подобные изображения! Чтобы компьютер мог работать с изображениями, необходимо ограничиться запоминанием конечного количества объектов пространства (точек или областей). *Дискретизация* и есть способ выделения конечного числа пространственных элементов, информация о которых будет сохранена в компьютере. Информация обо всех остальных элементах пространства при дискретизации *утрачивается*! □

С информационной точки зрения графическое изображение является совокупностью световых сигналов на плоскости: отдельные световые сигналы различаются местоположением, цветовым оттенком и яркостью.

Цвет и яркость — характеристики точек изображения, их можно *измерять*, т. е. выражать в числах. Как цвет, так и яркость могут изменяться непрерывно, поэтому их следовало бы выражать вещественными числами. Но в этом случае их невозможно абсолютно точно представить в компьютере. Поэтому все измеряемые непрерывные характеристики (как, например, яркость точек изображения или мгновенная громкость звука) подвергают *квантованию*.

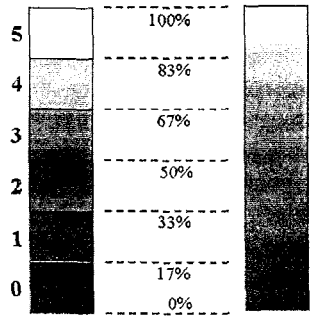
Определение 9. *Квантованием* (англ. *quantisation* или *quantization*) называют процедуру преобразования непрерывного диапазона всех возможных входных значений измеряемой величины в дискретный набор выходных значений.

Обычно при квантовании диапазон возможных значений измеряемой величины разбивается на несколько поддиапазонов. При измерении определяется поддиапазон, в который попадает значение, и в компьютере сохраняется только номер поддиапазона.

Пример 24. Квантование шкалы оттенков серого цвета.

В правой части рисунка изображена шкала *градаций серого* цвета, от черного до белого. Для *численного описа-*

ния яркости цвета надо каждому оттенку поставить в соответствие некое число. Как правило, черному цвету приписывают нулевой уровень яркости, белому — единичный, а промежуточным серым тонам — дробные числа в интервале от 0 до 1, выражающие яркость оттенка как долю от максимальной яркости. Эту величину мы будем выражать в процентах от максимальной яркости, т. е. черный имеет 0% яркости, белый — 100%.



Такое описание цвета непригодно для непосредственного компьютерного представления, так как вещественные числа недискретны, для точной записи их значений надо хранить бесконечное количество цифр! Следовательно, надо провести квантование, для чего диапазон значений яркости разбивают на поддиапазоны, или *уровни*. В нашем примере диапазон яркости серого цвета разбиваем на 6 уровней равной ширины: 0-й уровень — 0–17% максимальной яркости, 1-й уровень — 17–33% максимальной яркости и т. д. Тогда интервалы значений яркости и номера уровней связаны друг с другом простым соотношением: k-й уровень соответствует интервалу

$$\left[k \cdot \frac{100}{6};\ (k+1) \cdot \frac{100}{6}\right].$$

По величине яркости легко вычислить номер соответствующего уровня: $k = [x \cdot 6/100]$ (здесь квадратные скобки означают целую часть числа), где x — величина яркости, k — номер уровня.

Пусть яркость серого оттенка составляет 70%. При квантовании $\left(\left[\frac{70}{100} \cdot 6\right] = \left[\frac{420}{100}\right] = 4\right)$ это значение попадает в 4-й поддиапазон (67–83%), поэтому в компьютере этот оттенок серого будет закодирован целым числом 4. □

Дискретизация и квантование всегда приводят к потере некоторой доли информации. Так, компьютерное изображение живописного полотна всегда отличается от оригинала. А цифровая запись музыкального произведения или концерта (например, на компакт-диске) всегда отличается от живого звучания, даже если различие неощутимо на слух. Степень различия оригинала и цифровой копии определяет *субъективное качество* компьютерного представления.

Из-за больших размеров звуковых, графических и видеофайлов они очень редко хранятся в компьютере в неупакованном виде. Для уменьшения их размеров используют *сжатие информации*. Универсальные, или так называемые *обратимые алгоритмы сжатия*, никак не используют знания о характере обрабатываемой информации и поэтому упаковывают ее достаточно слабо.

Для эффективного сжатия звуковой и графической информации относительно недавно были разработаны специальные алгоритмы, учитывающие специфику человеческого восприятия звука и изображений. Характерной особенностью этих алгоритмов является возможность регулируемого удаления маловажной (с точки зрения человеческого восприятия) информации, поэтому такие алгоритмы сжатия обобщенно называют *алгоритмами с регулируемой потерей информации*. За счет удаления части информации удается добиться очень большой степени сжатия данных при субъективно незначительной потере качества.

Алгоритмы с регулируемой потерей информации *неуниверсальны*, они не могут использоваться для сжатия любых данных, поскольку полное восстановление исходной информации *невозможно*.

При пространственной дискретизации изображений пользуются *растровым* и *векторным* представлением графической информации. Растровое представление можно охарактеризовать как *поточечное представление*, а векторное — как *структурное представление* изображения.

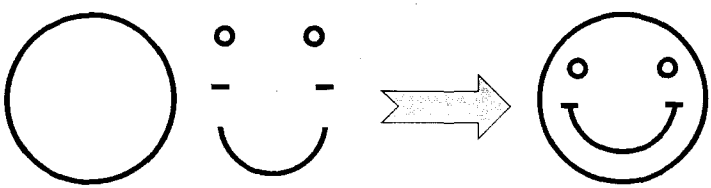
2.4.2. Векторное и растровое представление графической информации

Помните, как вы рисовали в детстве?

Точка, точка, запятая —
Вышла рожица кривая.
Палка, палка, огуречик —
Получился человечек!

Слова этой детской песенки очень точно характеризуют суть векторного представления графической информации.

Векторное представление описывает, как построить исходное изображение при помощи стандартных геометрических фигур из заранее определенного набора, например из отрезков и дуг.



Геометрические фигуры из стандартного набора называют *элементарными фигурами*, или *примитивами* (англ. *primitives*). Построение векторного представления называется *векторизацией* изображения. При выполнении векторизации изображение анализируют, разбивают на примитивы, а затем сохраняют их параметры: положение, размеры и цвет.

Многие виды изображений по своей природе хорошо структурированы и поэтому очень удобны для векторизации: это графики, диаграммы, чертежи, схемы, планы, карты, символы, гербы и флаги, логотипы, всевозможные стилизованные изображения.

Пример 25. Любой чертеж содержит отрезки, окружности, дуги. Положение каждого отрезка на чертеже можно задать координатами двух точек, определяющими его начало и конец; окружность — координатами центра и длиной радиуса; дугу — радиусом, а также координатами начала и конца дуги. Кроме того, для каждой линии можно указать ее тип: тонкая, штрихпунктирная и т. д.

Такая информация о чертеже вводится в компьютер как обычная буквенно-цифровая и обрабатывается в дальнейшем специальными программами. □

В отличие от хорошо структурированных изображений существуют изображения, которые вообще не имеют четкой структуры. К ним относятся фотографии, живописные полотна, рукописные тексты и т. д. Такие изображения крайне неудобны для векторизации.

Для хранения подобных изображений используют *растровое представление*: все изображение разбивается на множество очень маленьких элементов, причем, в отличие от векторного представления, размеры и положение элементов задаются заранее (*a priori*) и совершенно не зависят от самого изображения. В пределах каждого такого элемента изображение считается однородным, т. е. имеющим один и тот же цвет.

Порядок разбиения изображения на элементы называется *растром*, а сами элементы — *пикселями* (*pixel* — аббревиатура от англ. *picture element* — элемент изображения). Пиксели — это «атомы» растрового изображения, на меньшие части его не делят. Как правило, пиксели очень малы, так что их отождествляют с точками. Процедура разбиения изображения на пиксели называется *растеризацией* или *оцифровкой* изображения.

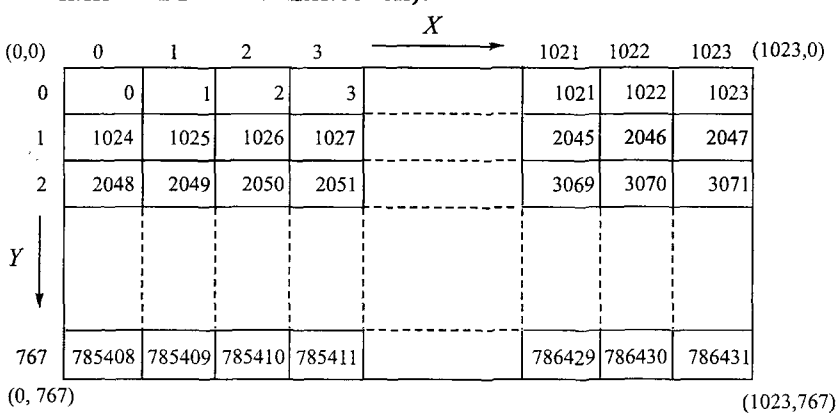
Заметим, что хранение рисунка в векторной форме обычно на несколько порядков сокращает необходимый объем памяти по сравнению с растровой формой представления.

Определение 10. *Растр* — специальным образом организованная *совокупность пикселей*, представляющая изображение. Координаты, форма и размер пикселей задаются при определении растра. Изменяемым атрибутом пикселей является цвет.

В технике и компьютерной графике чаще всего используется *прямоугольный растр*, в котором пиксели составляют прямоугольную матрицу, ее основными параметрами являются размеры растровой матрицы, т. е. количество столбцов и строк, составленных из пикселей.

Главное преимущество прямоугольных растров заключается в том, что положение каждого пикселя на экране (или на изображении) можно *вычислить*, зная только размеры растровой матрицы и линейные размеры пикселей либо *плотность размещения* пикселей, которую обычно измеряют в количестве точек на дюйм (dpi, ***Dots Per Inch***). Для этого достаточно ввести правила перечисления пикселей. Например, в мониторах персональных компьютеров пиксели перечисляются слева направо и сверху вниз: сперва нумеруются все пиксели в верхней строке слева направо, затем нумерация продолжается на нижележащей строке и т. д.

Так, например, если известно, что фотография сохранена в формате JPEG с размерами 768 × 576, то это значит, что матрица пикселей состоит из 768 столбцов и 576 строк. Операционные системы с графическим пользовательским интерфейсом (такие как Windows, MacOS, графическая подсистема X Window в UNIX и т. д.) представляют экран дисплея как растровое прямоугольное изображение некоторого размера (например, 800 × 600 или 1024 × 768 пикселей).



2.4.3. Квантование цвета

Как было сказано выше, графическую информацию естественного происхождения при вводе в компьютер необходимо подвергать операциям пространственной дискретизации и квантования цвета.

Квантование (кодирование) цвета базируется на *математическом описании цвета*, которое, в свою очередь, опирается на тот факт, что цвета можно *измерять* и *сравнивать*. Научная дисциплина, изучающая вопросы измерения цветовых характеристик, называется *метрологией цвета*, или *колориметрией*. Человек обладает очень сложным цветовосприятием, достаточно заметить, что зрительные центры мозга у новорожденных детей в течение нескольких месяцев (!) только тренируются видеть. Поэтому и математическое описание цвета тоже весьма нетривиально.

Ученым долгое время не удавалось объяснить процесс цветовосприятия. До середины XVII века господствовала умозрительная теория Аристотеля, согласно которой все цвета образуются при подмешивании черного цвета к белому.

Первые серьезные результаты в этой области получил Исаак Ньютон, который описал составную природу белого света и установил, что *спектральные цвета* являются *неразложимыми* и что путем смешения спектральных цветов можно синтезировать белый цвет и всевозможные оттенки других цветов. Ньютон выделил в спектре белого света семь наиболее заметных спектральных цветов и назвал их *основными* — красный, оранжевый, желтый, зеленый, голубой, синий и фиолетовый.

М. В. Ломоносов (1711–1765)

Примерно полстолетия спустя, в 1756 году, выдающийся русский ученый М. В. Ломоносов сформулировал так называемую *трехкомпонентную теорию цвета*, обобщив огромный эмпирический материал, накопленный им при разработке технологии производства цветного стекла и мозаики. Исследуя вопросы окрашивания стекол, Ломоносов обнаружил, что для придания стеклу любого цветового оттенка достаточно использовать всего три основные краски, смешивая их в определенных пропорциях.

Спустя примерно столетие выдающийся немецкий ученый Герман Грассман (1809–1877) ввел в трехкомпонентную теорию цвета математический аппарат в форме *законов Грассмана* для аддитивного синтеза цвета. Наиболее важными из них являются следующие два закона.

Закон трехмерности: с помощью трех линейно независимых цветов можно однозначно выразить любой цвет. Цвета считаются линейно независимыми, если никакой из них нельзя получить путем смешения остальных.

Закон непрерывности: при непрерывном изменении состава цветовой смеси результирующий цвет также меняется непрерывно. К любому цвету можно подобрать бесконечно близкий цвет.

Трехкомпонентная теория цвета стала основой колориметрии, однако обоснование этой теории появилось только на рубеже XIX–XX веков, после того, как была изучена физиология органов зрения.

Колориметрические законы Грассмана устанавливают общие свойства математических моделей цвета. Фактически законы Грассмана постулируют, что любому цвету можно однозначным образом поставить в соответствие некоторую точку трехмерного пространства. Точки пространства, которые соответствуют цветам, воспринимаемым человеческим глазом, образуют в пространстве некоторое выпуклое тело. Абсолютно черному цвету всегда соответствует точка {0, 0, 0}.

Таким образом, цвета можно рассматривать как точки или векторы в трехмерном цветовом пространстве. Каждая цветовая модель задает в нем некоторую систему координат, в которой основные цвета модели играют роль базисных векторов. А квантование цвета, по сути, является дискретизацией пространства цветов.

В компьютерной технике чаще всего используются следующие цветовые модели:

- RGB (***Red–Green–Blue***, красный–зеленый–синий).
- CMYK (***Cyan–Magenta–Yellow–blacK***, голубой–пурпурный–желтый–черный).
- HSB (***Hue–Saturation–Brightness***, цветовой оттенок–насыщенность–яркость).

Чтобы исключить неоднозначность трактования терминов *«яркость»*, *«насыщенность»*, *«цветовой оттенок»*, поясним их.

Яркость — это характеристика цвета, определение которой в основном совпадает с бытовым понятием яркости и физическим понятием освещенности или светимости. Ярко-красный, красный и темно-красный цвета различаются именно яркостью.

С физической точки зрения, яркость — это количественная мера потока световой энергии, излучаемой или отражаемой предметом в сторону наблюдателя. Так, при ярком солнечном свете и в сумерках один и тот же цветной рисунок выглядит по-разному. При этом цветовые оттенки не меняются, различными оказываются лишь яркости цветов.

Цветовой оттенок и насыщенность — это две другие независимые характеристики цвета.

Пусть у нас есть набор красок разного цвета. Смешением различных красок между собой мы будем получать новые цвета. Например, смесь равного количества желтой и синей красок даст зеленую краску. *Цветовой оттенок*, или *цветовой тон* рассматриваемого объекта связан со спектральным составом излучения. По цветовому тону объекта мы можем судить об окраске объекта — синей, зеленой, красной и т. д. Отдельные участки видимого спектра вызывают ощущение различных цветов.

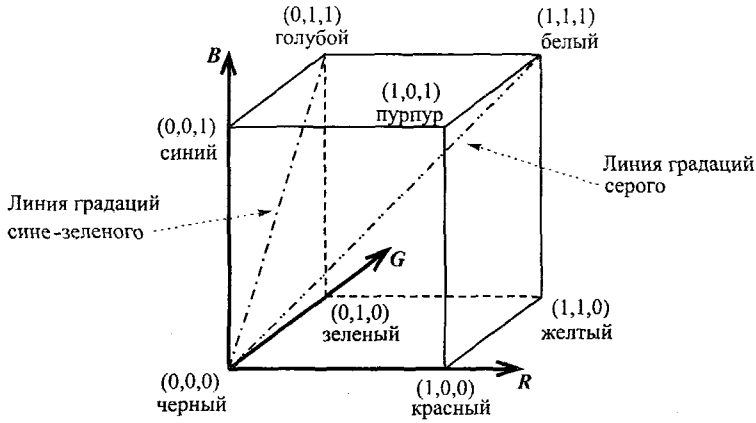
Насыщенность характеризует степень «разбавления» цветового тона белым цветом. Например, если ярко-красную (насыщенную) краску разбавить белой, то ее цветовой оттенок останется прежним, изменится только насыщенность. Ровно так же коричневый цвет, желтый и лимонный имеют один и тот же цветовой оттенок — желтый, их различие заключается в насыщенности цветового оттенка. Наибольшей насыщенностью обладает свет от монохромного источника.

Отметим, что для белого и черного цветов насыщенность составляет 0%, т. е. эти цвета не обладают насыщенностью. Именно поэтому, подмешивая их к цветной краске, мы меняем ее насыщенность, а не оттенок.

2.4.4. Цветовая модель RGB

В модели RGB *основными цветами* являются *красный*, *зеленый* и *синий*. Данная модель используется в основном при отображении графических изображений на эк-

ране монитора, телевизора, сотового телефона и т. д. Смешением трех основных цветов синтезируются все остальные цвета, их условные яркости (интенсивности) задаются вещественными числами от 0 до 1 (значение 1 соответствует максимальной яркости соответствующего цвета, которую может изобразить графическое устройство). Модель RGB определяет пространство цветов в виде единичного куба с осями «яркость красной компоненты», «яркость зеленой компоненты» и «яркость синей компоненты».

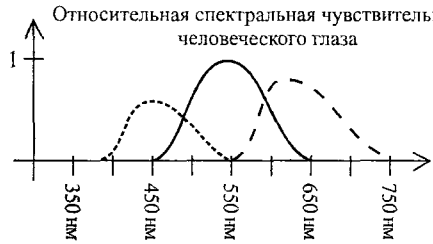


Характерные особенности RGB-модели

- Любая точка куба (r, g, b) определяет некоторый цвет.
- Точка (0, 0, 0) соответствует черному цвету, точка (1, 1, 1) — белому, а линия (0, 0, 0) – (1, 1, 1) описывает все градации серого цвета: от черного до белого.
- При движении по прямой от (0, 0, 0) через точку (r, g, b) получаем все градации яркости цвета $\{r, g, b\}$, от самой темной до самой яркой. Например, ($^1/_4$, $^1/_4$, 0) — темно-коричневый цвет, ($^1/_2$, $^1/_2$, 0) — коричневый, ($^3/_4$, $^3/_4$, 0) — желто-коричневый, (1, 1, 0) — желтый.
- На гранях куба $\{r = 0\}$, $\{g = 0\}$ и $\{b = 0\}$ расположены самые насыщенные цвета.
- Чем ближе точка к главной диагонали (0, 0, 0) – (1, 1, 1), тем менее насыщен соответствующий цвет.

У цветовой модели RGB есть физиологическое обоснование. Человеческий глаз содержит четыре типа зрительных рецепторов: «палочки» (рецепторы интенсивности) и

три типа «колбочек» (рецепторы цветовых оттенков). Колбочки каждого типа чувствительны к свету в своем узком диапазоне длин волн, для колбочек разных типов максимумы чувствительности приходятся на разные длины волн, диапазоны чувствительности частично перекрываются:



Тип колбочек	Диапазон длин волн	Максимум чувствительности
Красные	от 760 до 550 нм	~610 нм
Зеленые	от 650 до 450 нм	~550 нм
Синие	от 550 до 380 нм	~450 нм
Общий спектр видимого света	от 760 до 380 нм	555 нм (дневное зрение) 510 нм (ночное зрение)

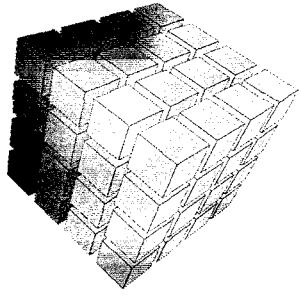
Именно благодаря неравномерной спектральной чувствительности и перекрытию диапазонов чувствительности человеческий глаз способен различать огромное количество цветов (около 10 млн).

Если направить в глаз составной световой сигнал с правильно подобранным соотношением яркостей красного, зеленого и синего цветов, то зрительные центры мозга не смогут отличить подмену и сделают вывод, что наблюдается нужный цвет! Такой механизм синтеза цветовых оттенков используется во всех современных типах цветных мониторов, телевизоров, дисплеев сотовых телефонов.

Чтобы использовать математическую RGB-модель для реального компьютерного представления графической информации, необходимо произвести *квантование цветового пространства*, т. е. найти способ представлять вещественные значения яркостей цветовых компонент в дискретной форме.

Наиболее простой способ добиться этого — перевести вещественные числа из интервала [0; 1) в интервал целых чисел от 0 до $N-1$ путем умножения на целое число N, с последующим округлением. Фактически, интервал [0; 1) разбивается на N равных подинтервалов вида:

$$\left[\frac{i}{N};\frac{i+1}{N}\right),\ i \in \{0,\ \ldots,\ N-1\}.$$



Разбиению на подинтервалы подвергают каждую из цветовых осей. Количество подинтервалов на «красной», «зеленой» и «синей» осях (N_r, N_g, N_b) может быть различным, но чаще принимается, что $N_r = N_g = N_b = N$.

После квантования каждый цвет представляется *триадой* целых неотрицательных чисел (k_r, k_g, k_b), $0 \le k_i < N_i$. Числа N_i обычно выбирают равными степени двойки $N_i = 2^{m_i}$, а величину M, равную сумме $m_r + m_g + m_b$, называют *глубиной цвета* или *глубиной цветности*.

Ниже приведена таблица наиболее распространенных видеорежимов с указанием количества отображаемых цветов (табл. 4).

Таблица 4

Видеорежим	Глубина цвета	Количество отображаемых цветов
256 цветов	8	$2^8 = 256$
High Color	16	$2^{16} = 65\ 536$
True Color	24	$2^{24} = 16\ 777\ 216$

Пример 26. В современных компьютерах в видеорежиме *TrueColor* на хранение информации об одной цветовой компоненте используется 1 байт, для сохранения цвета точки — 3 байта: $m_r = m_g = m_b = 8$, $N_r = N_g = N_b = 2^8 = 256$, $N_r \cdot N_g \cdot N_b = 2^{8+8+8} = 16\ 777\ 216$.

Глубины цветности 24 бита для мониторов вполне достаточно, чтобы создать видимость непрерывности шка-

лы цветовых оттенков. Особенности человеческого зрения таковы, что если на экране монитора изобразить две фигуры, цвета которых при глубине цвета 24 бита отличаются не более чем на 1 в каждой цветовой компоненте, то человек не сможет заметить разницу. □

В табл. 5 для стандарта *TrueColor* приведены двоичные значения уровней интенсивности некоторых цветов.

Таблица 5

Название цвета	Интенсивность основных цветов		
	Красный	Зеленый	Синий
черный	00000000	00000000	00000000
красный	11111111	00000000	00000000
зеленый	00000000	11111111	00000000
синий	00000000	00000000	11111111
голубой	00000000	11111111	11111111
пурпурный	11111111	00000000	11111111
желтый	11111111	11111111	00000000
белый	11111111	11111111	11111111

Пример 27. В видеорежиме *HighColor* цвет каждой точки кодируется 16 битами. На глубину красного и синего цвета отводится 5 бит, на глубину зеленого — 6 бит: $m_r = m_b = 5$, $m_g = 6$. Следовательно, шкала яркостей зеленого цвета содержит в два раза больше уровней, чем шкалы яркостей красного и синего цветов. Для экономии памяти биты цветовых компонент каждой точки записывают в два байта вместо трех. □

Задание. *Подсчитайте объем памяти, требуемый для сохранения изображения всего экрана для видеорежима с размером экрана 1024 × 768 пикселей и с глубиной цвета 24 бита.*

Решение. Экран монитора представляет собой прямоугольный растр, поэтому суммарное количество пикселей равно 1024 × 768 = 786 432 пикселей.

Для видеорежима с глубиной цвета 24 бита требуется 3 байта на каждый пиксель, так что общий объем требуемой памяти составит 1024 · 768 · 3 = 2 359 296 байт = = 2,25 Мбайт. □

Для непосредственной цифровой записи 1 секунды цветного видеоизображения без звука (25 кадров размером 1024 × 768 пикселей) потребуется примерно 60 Мб (25·1024·768·3 = 58 982 400 байт). Для записи двухчасового фильма необходимо около 400 Гб.

2.4.5. Цветовая модель CMYK

Цветовая модель CMYK также базируется на трехкомпонентной теории цвета, но, в отличие от модели RGB, основными цветами в ней являются *голубой*, *пурпурный* и *желтый*. Модель CMYK широко используется в цветной печати. Название модели является аббревиатурой английских названий основных цветов ***Cyan–Magenta–Yellow–blacK***. (О причине добавления черного цвета будет сказано ниже.)

Модель CMYK применяется в цветных принтерах общего назначения и в цветной офсетной печати низкого и среднего качества. Если рассмотреть под микроскопом цветные иллюстрации в какой-нибудь книге или цветной газете, то можно увидеть, что цветные фрагменты напечатаны очень маленькими частично перекрывающимися цветными точками (офсетами). Офсеты хорошо заметны на границах цветных участков и в местах с бледной окраской.

Главной причиной появления модели CMYK является различие в принципах формирования цвета при его воспроизведении на мониторах и при печати. Кто в детстве рисовал акварельными красками или гуашью, тот знает, что при смешении красной и зеленой красок получается не желтая краска (как было бы в модели RGB), а темно-коричневая. Дело в том, что при восприятии цвета с экрана монитора мы видим излучаемый свет, а при при рассматривании картинки, нарисованной на бумаге, — отраженный.

Пиксели монитора *излучают собственный свет*; чтобы создать на экране основной цвет, надо включить субэлемент определенного типа (пиксель монитора состоит из трех субэлементов: красного, зеленого и синего), а для получения составного цвета надо дополнительно включить (т. е. *добавить*) субэлементы другого типа, при этом суммарная яркость пикселя возрастет. Кстати, из-за такого принципа формирования составного цвета RGB-модель называют *аддитивной цветовой моделью* (от англ. *add* — добавлять).

В отличие от монитора, бумага *отражает падающий свет*, который обычно является «белым»: яркости всех его цветовых составляющих равны. Наносимые на бумагу краски являются *поглощающими светофильтрами* — они поглощают лучи определенного цвета, а остальные отражают. Видимый цвет краски определяется теми лучами, которые *не были поглощены*. Таким образом, краски могут только *вычитать*, или *ослаблять цвета* в отражаемом потоке света. По этой причине модель CMYK называют *субтрактивной цветовой моделью* (от англ. *subtract* — вычитать).

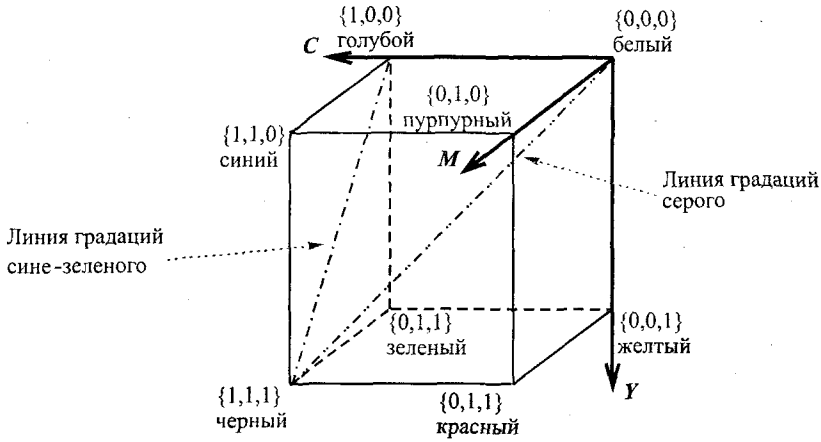
Основные цвета модели CMYK подобраны так, чтобы соответствующие краски поглощали свет в достаточно узкой области спектра: голубая краска сильно поглощает красный свет, пурпурная — зеленый, а желтая — синий.

Ниже показано, как формируются цвета в моделях RGB и CMYK (точнее, в идеальной модели CMY). Считаем, что краски в модели CMYK нанесены на бумагу, от которой, собственно, и происходит отражение непоглощенных лучей.

RGB				
Цвет	Черный	Красный	Зеленый	Синий
CMYK				
RGB				
Цвет	Желтый	Голубой	Пурпурный	Белый
CMYK				

В идеальном случае трех цветов (голубого, пурпурного и желтого) было бы вполне достаточно для формирования на бумаге любого цвета. Однако реально существующие краски не идеальны, они не поглощают цветовые компоненты полностью: если нанести на бумагу все три краски с наибольшей плотностью, то вместо чистого черного цвета получится темно-серый. Для коррекции цветовой гаммы используется четвертая краска — черная.

Пространство цветов модели CMYK также является единичным кубом. Яркости основных красок (или плотность закраски) задаются вещественными числами от 0 до 1.



Характерные особенности CMYK-модели

- Любая точка куба (c, m, y) определяет некоторый цвет.
- Точка (0, 0, 0) соответствует белому цвету, точка (1, 1, 1) — черному, а линия (0, 0, 0) – (1, 1, 1) описывает все градации серого цвета: от белого до черного.
- При движении по прямой от (0, 0, 0) через точку (c, m, y) получаем все градации яркости цвета {c, m, y}, от самой яркой до самой темной.
- Чем ближе точка к главной диагонали (0, 0, 0) – (1, 1, 1), тем менее насыщен соответствующий цвет.
- Если все три координаты точки (c, m, y) ненулевые, то цвет ненасыщенный.
- В модели CMYK оттенки серого цвета могут воспроизводиться путем добавления черной краски к основному набору цветов.

Квантование цвета в модели CMYK выполняется аналогично квантованию в модели RGB.

Как уже упоминалось в п. 2.4.4, цветовая модель RGB соответствует механизму синтеза цветов, используемому в мониторах. А поскольку в современных компь-

ютерах именно мониторы являются главным, наиболее часто используемым устройством вывода информации, то практически все форматы графических файлов хранят изображения в RGB-представлении, и лишь очень немногие графические форматы используют другие цветовые модели (например, в формате JPEG используется YUV-модель). Чтобы вывести на экран изображения, хранящиеся в таких графических файлах, программам приходится «на лету» выполнять преобразование графических данных в RGB-представление.

Модель CMYK соответствует механизму синтеза цветов, используемому в принтерах. Когда графическая информация выводится на принтер, приходится выполнять преобразование изображения из RGB-представления в CMYK-представление. Обычно эта работа выполняется средствами ОС, поскольку формулы пересчета RGB → CMYK просты.

Задание. *Выведите формулы связи между значениями цветовых компонент моделей RGB и CMYK.*

Подсказка. Сперва предположите, что краски являются идеальными, и поэтому не надо отдельно использовать черный цвет (идеализированная модель CMY, а не CMYK), и на основании этого предположения выведите формулы связи RGB ⇔ CMY. Затем предположите, что функция вычисления интенсивности черной краски через яркости RGB-компонент вам уже известна (дело в том, что эту функцию невозможно вывести априори, так как она определяется оптическими свойствами реальных красителей).

2.4.6. Цветовая модель HSB

Цветовая модель HSB (***H**ue–**S**aturation–**B**rightness*) описывает цветовое пространство через такие характеристики цвета, как *цветовой оттенок*, *насыщенность* и *яркость*.

Эти понятия были описаны в п. 2.4.3, дополним их более формальным определением.

Определение 11. *Чистый цветовой тон* — один из цветов спектрального разложения света. *Цветовой оттенок* — смесь чистого цветового тона с серым цветом. *Насыщенность цвета*, или степень чистоты цвета — доля чистого

тона в цветовой смеси (чем больше серого, тем меньше насыщенность). *Яркость* характеризуется общей светлостью смешиваемых цветов (чем больше черного, тем меньше яркость).

В модели HSB цвет описывается тройкой чисел {цветовой оттенок, яркость, насыщенность}. Рассмотрим ряд цветов: красный, темно-красный, красновато-черный, алый, розовый, бледно-розовый. В модели HSB эти цвета — производные от красного цвета и отличаются друг от друга только яркостью и насыщенностью красного оттенка. Такое описание цвета (в отличие от моделей RGB и CMYK) очень точно передает *субъективное восприятие цвета* человеком, а не технические особенности воспроизведения цветов. Подобные описания широко используются во всех областях искусства и производства, где приходится иметь дело с цветом.

Пространство цветов модели HSB имеет форму вложенных концентрических конусов с общей вершиной и общей осью симметрии. Цвета с одинаковой насыщенностью расположены на конической поверхности с определенным углом при основании. Цвета с одинаковой яркостью расположены по кругу — сечению объемного конуса плоскостью, перпендикулярной его оси. При этом вершина конуса соответствует черному цвету. Цвета с одинаковым оттенком расположены в полуплоскости, проходящей через ось симметрии конуса.

Таким образом, пространство HSB организовано следующим образом (рис. 2.1):

- ось конуса — это ось яркости;
- ось цветовых оттенков — окружность в основании конуса;
- насыщенность цвета определяется как угол между осью симметрии конуса и лучом, проходящим через вершину конуса и заданную точку.

Ось цветовых оттенков строится следующим образом: цвета спектра, от красного до фиолетового, и оттенки фиолетово-красного (которых нет в спектре) размещаются на окружности. Точку, соответствующую чистому красному цвету, принимают за ноль на круговой шкале цветовых оттенков (рис. 2.2). Все величины измеряют либо в градусах (0°–360°), либо в условных единицах от 0 до 1. На оси конуса расположены оттен-

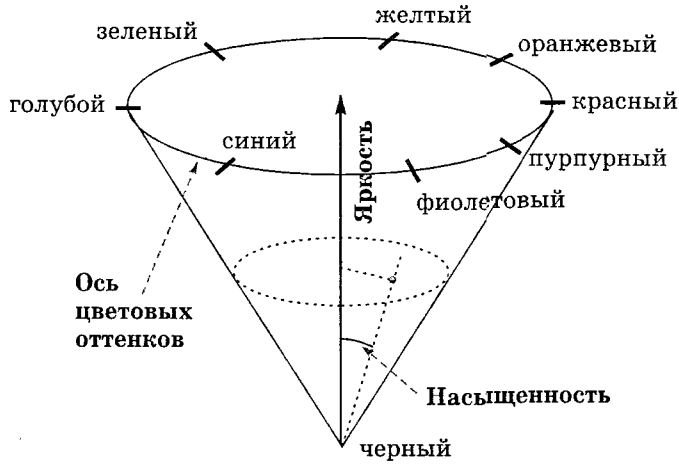


Рис. 2.1. Цветовое пространство модели HSB

ки серого цвета. Чем ближе к краю конуса, тем насыщеннее цвета.

Чтобы использовать математическую модель HSB для компьютерного представления графической информации, надо, как и для моделей RGB и CMYK, провести *квантование цветового пространства*, т. е. непрерывно изменяемые значения компонент цвета представить в дискретной форме. В ОС Windows каждая из HSB-характеристик описывается одним байтом, т. е. шкала значений разделена на 256 уровней.

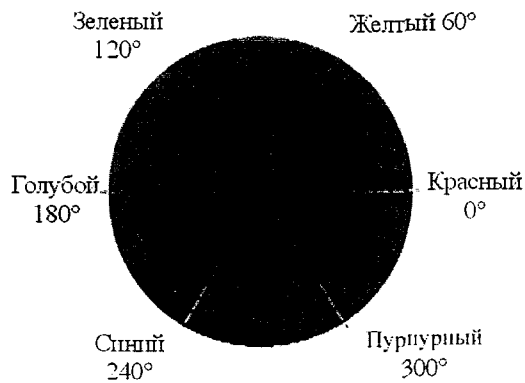


Рис. 2.2. Разрез конуса в плоскости постоянной яркости

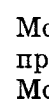
Вопрос. *Для чего к цветовым оттенкам на цветовой шкале были добавлены оттенки фиолетово-красного цвета?*

Ответ. Проходя сквозь призму или дифракционную решетку, световые лучи с различными длинами волн отклоняются на разные углы: призмы и дифракционные решетки осуществляют *пространственное разделение света*. Если за призмой поставить белый экран, то преломленный в призме луч белого света создаст на экране разноцветную полосу — спектр. Каждый цвет спектра является монохромным (как излучение лазера) — он создается лучами определенной длины волны. В спектре присутствуют цветовые оттенки от красного до фиолетового, но нет промежуточных оттенков фиолетово-красного, так что спектральные цвета нельзя «замкнуть» в непрерывную круговую ось. Однако в природе такие цвета есть, например малиновый, пурпурный и т. д. (но они имеют принципиально составной характер).

Разомкнутость спектральной цветовой оси была досадной помехой на пути создания математической модели, поэтому ученые просто добавили в цветовую шкалу мнимые «чистые оттенки» фиолетово-красного и тем самым замкнули ее. Масштаб на цветовой оси был выбран так, чтобы субъективно наиболее контрастные цвета оказались расположены друг напротив друга на цветовом круге. □

Все три рассмотренные цветовые модели (RGB, CMYK, HSB) описывают одно и то же реально существующее цветовое пространство. Их взаимный анализ позволяет отметить следующее:

- в цветовом пространстве модели HSB очень хорошо видна связь между моделями RGB и CMYK: на цветовом круге основные цвета одной модели расположены точно напротив основных цветов другой модели;
- кроме того, если на цветовом круге отметить точками основные цвета RGB-модели, то они образуют равносторонний треугольник, то же самое можно сказать и относительно модели CMYK;
- цвета модели HSB, которые не попадают в этот треугольник, в RGB-модели будут непредставимы. То же самое можно сказать и относительно модели CMYK.



Модель HSB позволяет представить (закодировать) практически все цвета, воспринимаемые человеком. Модели RGB и CMYK описывают возможности компьютерных устройств по воспроизведению цвета. И оказывается, что некоторые цвета в принципе не могут быть воспроизведены на компьютере.

Вопрос. *С какой целью основные цвета RGB-модели расположены на цветовой оси модели HSB так, что они образуют равносторонний треугольник?*

Ответ. Равносторонний треугольник имеет наибольшую площадь среди всех треугольников, вписанных в заданную окружность (докажите это), т. е. при таком выборе основных цветов количество воспроизводимых цветов при переходе от модели HSB к модели RGB будет максимальным. □

Вопросы и задания

1. Будем считать, что каждый пиксель черно-белого изображения кодируется 1, если он окрашен, и 0 — в противном случае. Декодируйте черно-белое изображение, оцифрованное следующим образом (каждая строка изображения закодирована здесь четырехзначным шестнадцатеричным числом):
   а) 0070 00FC 00F7 00FF 8078 C060 C070 FFF8 FFB8 FF38 8E78 E0F0 7FE0;
   б) 0100 0180 01C0 01E0 01F0 01F8 01FC 01FE 0180 0180 7FFE 3FFC 1FF8 0FF0.
2. Рассчитайте объем видеопамяти, необходимой для хранения графического изображения, занимающего весь экран монитора с разрешением 640 × 480 и количеством отображаемых цветов, равным 65 536.
3. Подсчитайте объем информации, передаваемой от видеоадаптера к монитору в видеорежиме 1024 × 768 пикселей с глубиной цвета 24 бита и частотой обновления экрана 85 Гц.
4. Вы хотите работать с разрешением 1600 × 1200 пикселей, используя 16 777 216 цветов. В магазине продаются видеокарты с памятью 512 Кбайт, 2 Мбайта, 4 Мбайта и 64 Мбайта. Какие из них можно купить для вашей работы?

5. Зачем нужны видеокарты с размером видеопамяти 128 Мб и более?
6. При печати цветного изображения на бумаге используется модель CMYK. Голубой цвет является дополнительным к красному (поглощает его на бумаге). Синий цвет поглощается желтым, а зеленый — пурпурным. Черный цвет получается, если нанести на бумагу все три поглощающих цвета. Сказанное выше занесено в три строки таблицы. Заполните остальные строки последнего столбца таблицы.

Голубой (нет красного)	Желтый (нет синего)	Пурпурный (нет зеленого)	Цвет
0	0	0	
0	0	1	Пурпурный
0	1	0	Желтый
1	0	0	Голубой
0	1	1	
1	0	0	
1	1	0	
1	1	1	

§ 2.5. Представление звуковой информации

Звук — это волновые колебания давления в упругой среде (в воздухе, воде, металле и т. д.). Для обозначения звука часто используют термин *«звуковая волна»*.

Основные параметры любых волн, и звуковых в том числе, — частота и амплитуда колебаний. *Частоту звука* измеряют в герцах (Гц, количество колебаний в секунду). Человеческое ухо способно воспринимать звук в широком диапазоне частот, примерно от 16 Гц до 20 кГц. В нетехнических областях (например, в музыке) вместо термина «частота» нередко используют термин «тон».

Амплитуду звуковых колебаний называют *звуковым давлением* или *силой звука*, эта величина характеризует воспринимаемую громкость звука. Абсолютную величину звукового давления измеряют в единицах давления — паскалях (Па). Самые слабые, едва различимые звуки имеют амплитуду около 20 мкПа ($2 \cdot 10^{-5}$ Па, так называемый *порог слышимости*). Самые сильные звуки, еще не выводящие слуховые органы из строя, могут иметь амплитуду до 200 Па (так называемый *болевой порог*). Из-за

столь широкого диапазона значений (максимальное и минимальное значения отличаются на 6–7 порядков!) абсолютными величинами звукового давления пользоваться крайне неудобно, на практике обычно используют *логарифмическую шкалу децибелов.*

Относительную силу звука, или *уровень звука*, определяют как логарифм отношения абсолютной величины звукового давления к величине порога слышимости, умноженный на некоторый постоянный коэффициент. Уровень звука измеряют в особых единицах — *децибелах* (обозначаются *дБ*). Ниже приводится формула расчета уровня звука:

$$L = 20 \cdot \lg(P_{зв}/P_{пс}),$$

где L — уровень звука (в дБ), $P_{пс}$ — порог слышимости ($2 \cdot 10^{-5}$ Па), $P_{зв}$ — давление измеряемого звука (в Па).

Логарифмическая шкала децибелов на практике весьма удобна, хотя поначалу пользоваться ей непривычно:

- весь диапазон слышимых звуков составляет 0–140 дБ: 0 дБ — порог слышимости, 140 дБ — болевой порог;
- человеческое ухо способно уловить различие в громкости, если звуки отличаются по силе не менее, чем на 10%, что соответствует разнице в уровнях примерно на 1 дБ;
- двукратное различие в амплитуде звуков соответствует различию уровней в 6 дб;
- если уровни звуков отличаются на 20 дБ, то амплитуды отличаются в 10 раз, а разница в 40 дБ соответствует 100-кратному различию в амплитудах.

Приведем некоторые значения уровней звука:

Порог слышимости	0 дБ
Шорох листьев, шум слабого ветра	10–20 дБ
Шепот (на задней парте)	20–30 дБ
Разговор средней громкости (в кабинете директора)	50–60 дБ
Автомагистраль с интенсивным движением	80–90 дБ
Авиадвигатели	120–130 дБ
Болевой порог	≈140 дБ

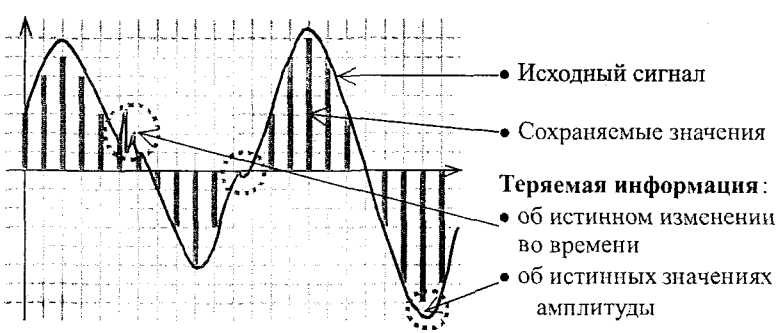
2.5.1. Понятие звукозаписи

Звукозапись — это процесс сохранения информации о параметрах звуковых волн. Способы хранения, или записи, звука разделяются на аналоговые и цифровые. При *аналоговой записи* на носителе размещается непрерывный «слепок» звуковой волны. Так, на грампластинке пропечатывается непрерывная канавка, изгибы которой повторяют амплитуду и частоту звука, а на магнитной ленте параметры звука сохраняются в виде намагниченности рабочей поверхности, степень намагниченности непрерывно изменяется, повторяя параметры звука.

В компьютерах применяется исключительно цифровая форма записи звука. При цифровой записи звук необходимо подвергнуть *временно́й дискретизации* и *квантованию*: параметры звукового сигнала измеряются не непрерывно, а через определенные промежутки времени (временна́я дискретизация); результаты измерений записываются в цифровом виде с ограниченной точностью (квантование).

Вообще говоря, в компьютер приходит не сам звук, а электрический сигнал, снимаемый с какого-либо устройства: например, микрофон преобразует звуковое давление в электрические колебания, которые в дальнейшем и обрабатываются. К компьютеру можно подключить и магнитофон, и радио, и эхолот — словом, любое устройство, вырабатывающее электрические сигналы.

Цифровая запись вносит двойное искажение в сохраняемые параметры сигнала: во-первых, при дискретизации теряется информация об истинном изменении звука между измерениями, а во-вторых, при квантовании со-



храняются не точные параметры, а только близкие к ним дискретные значения.

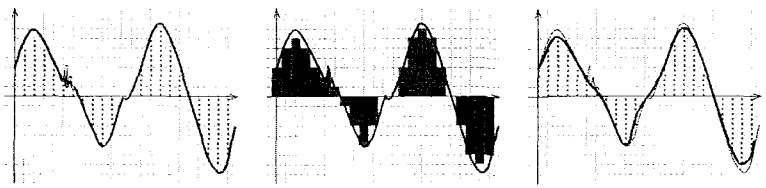
В компьютерах используются так называемые импульсно-кодовое и частотное представления звуковой информации, для обозначения которых чаще всего используют названия технических способов воспроизведения звука: *импульсно-кодовая модуляция* и *частотная модуляция*. Здесь мы рассмотрим первый из них.

2.5.2. Импульсно-кодовая модуляция

Импульсно-кодовая модуляция (англ. ***Pulse Code Modulation***, PCM) заключается в том, что звуковая информация хранится в виде значений амплитуды, взятых в определенные моменты времени (т. е. измерения проводятся «импульсами»).

При записи звука в компьютер амплитуда измеряется через равные интервалы времени с некоторой достаточно большой частотой.

При воспроизведении звука компьютер использует сохраненные значения для того, чтобы восстановить непрерывную форму выходного сигнала.



Исходный сигнал Хранимая информация Воспроизводимый сигнал

Процесс получения цифровой формы звука называют *оцифровкой*. Устройство, выполняющее оцифровку звука, называется *аналого-цифровым преобразователем* (АЦП). Устройство, выполняющее обратное преобразование, из цифровой формы в аналоговую, называется *цифро-аналоговым преобразователем* (ЦАП). В современных компьютерах основная обработка звука выполняется *звуковыми картами*. Помимо АЦП и ЦАП звуковые карты содержат *сигнальный процессор* — специализированный микрокомпьютер для обработки оцифрованного звука, выполняющий значительную часть рутинных расчетов при обработке звуков (смешение звуков, наложение спецэффектов,

расчет формы выходного сигнала и т. п.; центральный процессор не тратит время на выполнение этих работ).

Определение 12. Моменты измерения амплитуды называют *отсчетами*. Частоту, с которой производят измерения сигнала, называют *частотой дискретизации*.

Квантование звука заключается в следующем. Сначала мгновенные значения звукового давления измеряются с ограниченной точностью, затем, как и в случае с квантованием цветов, диапазон значений амплитуды разбивается на подуровни. По измеренному значению определяется подуровень, в который попадает значение, и в компьютере сохраняется только его номер. Количество бит, используемых для записи номеров подуровней, называется *глубиной кодирования звука*.

Если сравнить способы представления графической и звуковой информации, то импульсное кодирование звука соответствует растровому представлению изображений:

- структура звука (в графике — изображения) не анализируется;
- время (в графике — пространство) априори разбивается на небольшие области; в пределах этих областей параметры звука (изображения) считаются постоянными.

При рассмотрении представления графической информации упоминалось, что растровое представление изображения не требует хранения координат отдельных пикселей. Аналогично, при сохранении импульсного представления звука достаточно единожды сохранить параметры оцифровки (глубину кодирования, частоту дискретизации и длительность звукового фрагмента), а затем сохранять только номера подуровней единым потоком.

Увеличивая частоту дискретизации и глубину кодирования, можно более точно сохранить и впоследствии восстановить форму звукового сигнала. При этом улучшается субъективное качество оцифрованного звука, однако увеличивается объем сохраняемых данных. При цифровой записи звука в различных случаях используют разные значения частоты дискретизации и глубины кодирования. Например, в цифровых автоответчиках

используют частоту дискретизации 8–11 кГц и 8 бит для записи амплитуды, а стандарт записи звука на компакт-дисках соответствует частоте дискретизации 44,1 кГц и 16 бит для амплитуды на каждый аудио-канал (стереозвук — 2 канала, моно — один канал).

Пример 28. Оценим объем стереоаудиофайла в формате PCM с глубиной кодирования 16 бит и частотой дискретизации 44,1 кГц, который хранит звуковой фрагмент длительностью звучания 1 секунда.

Объем такого звукового фрагмента равен:
16 бит × 44100 Гц × 2 (канала) = 1 411 200 бит =
= 176 400 байт ≈ 172,3 Кбайт. □

Возникает вполне естественный вопрос: до какой степени можно уменьшать параметры оцифровки (а значит, и объем оцифрованного звукового фрагмента), чтобы при восстановлении звук оставался достаточно близок к исходному?

В 1928 году американский инженер и ученый Гарри Найквист высказал утверждение, что частота дискретизации должна быть в два или более раза выше максимальной частоты измеряемого сигнала. В 1933 году советский ученый В. А. Котельников и независимо от него американский ученый Клод Шеннон сформулировали и доказали теорему, более сильную, чем утверждение Найквиста, о том, при каких условиях и как по дискретным значениям можно восстановить форму непрерывного сигнала. Эта теорема в России называется теоремой Котельникова, на Западе — теоремой Найквиста—Шеннона; есть у нее и «нейтральное» название — *теорема об отсчетах*.

В. А. Котельников

Котельников Владимир Александрович (1908 г. р.), академик АН СССР, область научных интересов — радиотехника. Основные труды посвящены проблемам совершенствования методов радиоприема, изучению помех радиоприему и разработке методов борьбы с ними.

Теорема Найквиста—Котельникова—Шеннона утверждает, что если имеется сигнал $U(t)$, спектр которого ограничен сверху частотой f, то после его дискретизации

с частотой $F > 2 \cdot f$ форму исходного сигнала можно точно восстановить по дискретным значениям (отсчетам), по следующей формуле:

$$U(t) = \sum_{k=-\infty}^{+\infty} U(k\Delta t) \frac{\sin(2\pi F(t - k\Delta t))}{2\pi F(t - k\Delta t)},$$

где $\Delta t = 1/F$ — время между отсчетами,

$k\Delta t$ — время k-го отсчета,

$U(k\Delta t)$ — значение k-го отсчета.

Покажем, почему частота дискретизации должна быть как минимум вдвое выше частоты сигнала. Допустим, на вход АЦП был подан синусоидальный сигнал, и АЦП выдал его цифровое представление. Вопрос: можно ли однозначно определить форму входного сигнала?

Ответ на поставленный выше вопрос получается удручающий — решение не единственно, даже синусоидальных сигналов, имеющих такое цифровое представление, несколько. Точнее, подходящие синусоиды образуют два бесконечных семейства. На рис. 2.3 по горизонтальной оси откладывается время, по вертикальной — амплитуда сигнала; толстые вертикальные линии обозначают измеренные значения; сплошной линией отмечен синусоидальный сигнал с низкой частотой; пунктирной линией отмечен синусоидальный сигнал с высокой частотой. Этот пример показывает, что по одним лишь отсчетам мы принципиально не можем определить характеристики входного сигнала (это, кстати, является следствием того, что при дискретизации непрерывных сигналов часть информации утрачивается). Однако если потребовать, чтобы частота искомого сигнала не превосходила половины частоты дискретизации, то решение будет единственным.

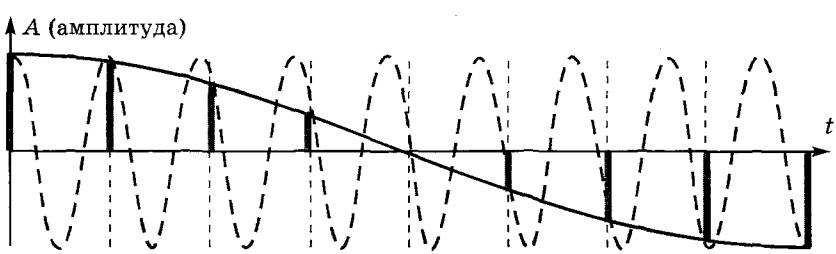


Рис. 2.3

Вообще говоря, точная формулировка теоремы Найквиста—Котельникова применима только к сигналам с неизменными частотными характеристиками и бесконечной длительностью, так что для оцифровки реальных звуковых сигналов частоту дискретизации выбирают с небольшим запасом.

Пример 29. Продемонстрируем использование теоремы Найквиста—Котельникова на практике.

Сотовые телефоны и цифровые автоответчики предназначены для передачи/записи голоса человека, спектр частот которого достаточно узок (не более 3 кГц), поэтому в них используется низкая частота дискретизации (обычно 8–11 кГц). Звуковые компакт-диски (англ. *Audio CD*) предназначены для записи в цифровой форме любого звукового материала с высоким качеством. Для человеческого уха наибольшая слышимая частота звука составляет около 20 кГц. Поэтому в стандарте Audio CD приходится использовать частоту дискретизации 44,1 кГц (что всего на 10% превосходит удвоенную наибольшую слышимую частоту). Приборы с низкой частотой дискретизации в принципе не способны обеспечить качество воспроизведения, какое дает компакт-диск: попробуйте записать музыку с компакт-диска на автоответчик, а затем прослушайте ее — разница в качестве будет легко ощутима! □

2.5.3. Формат MIDI

В 80-х годах прошлого века появились электронные музыкальные инструменты — синтезаторы, способные воспроизводить не только звуки многих существующих музыкальных инструментов, но и абсолютно новые звуки. Было разработано соглашение о системе команд универсального синтезатора, получившее название *стандарта MIDI* (англ. ***Musical Instrument Digital Interface***). Запись музыкального произведения в формате MIDI — последовательность закодированных сообщений синтезатору. Сообщение может быть командой (нажать или отпустить определенную клавишу, изменить высоту или тембр звучания), описанием параметров воспроизведения (например, силы давления на клавиатуру) или управляющим сообщением (включение полифонического режима, синхронизирующее сообщение).

MIDI–команды делают запись музыкальной информации более компактной, чем импульсное кодирование. Если сравнить способы представления графической и звуковой информации, то запись звука в виде MIDI-команд соответствует векторному представлению изображений.

Записанные звуковые файлы можно редактировать, т. е. вырезать, копировать и вставлять фрагменты из других файлов. Кроме того, можно увеличивать или уменьшать громкость, применять различные звуковые эффекты (эхо, уменьшение или увеличение скорости воспроизведения, воспроизведение в обратном направлении и др.), а также накладывать файлы друг на друга (*микшировать*).

2.5.4. Принципы компьютерного воспроизведения звука

При воспроизведении звука на компьютере цифровое представление сигнала преобразуют обратно в аналоговую непрерывную форму. Как уже упоминалось, расчет параметров выходного сигнала выполняет сигнальный процессор, а генерацию аналогового электрического сигнала выполняет ЦАП (цифро-аналоговый преобразователь).

В современной цифровой звукотехнике (например, в компьютерных звуковых картах) используют несколько методов реконструкции формы аналогового сигнала. Эти методы сильно отличаются друг от друга даже идеями, взятыми за их основу.

Общим в методах реконструкции формы сигнала является то, что цифровой сигнал сперва подвергается *передискретизации* – увеличению частоты дискретизации и глубины квантования в несколько раз (например, поток звуковых данных с Audio CD с частотой дискретизации 44,1 кГц и глубиной квантования 16 битов, в звуковой карте преобразуется в поток с частотой дискретизации 192 кГц и глубиной квантования 24 бита).

Для вычисления амплитуд сигнала во вставленных новых отсчетах используются различные математические методы, например *полиномиальная интерполяция*. Суть этого метода состоит в том, что по известным значениям функции $f(x)$ в дискретных моментах времен x_1, x_2, ..., x_n мы можем восстановить ее значения в других

точках x, т. е. по нескольким подряд идущим отсчетам можно построить интерполяционный многочлен, проходящий через заданные точки (например, интерполяционный многочлен Лагранжа). По «старым» значениям отсчетов сигнала вычисляются коэффициенты многочлена, и с их помощью вычисляется амплитуда сигнала в отсчетах, которые вставляются при передискретизации (*рис. 2.4*).

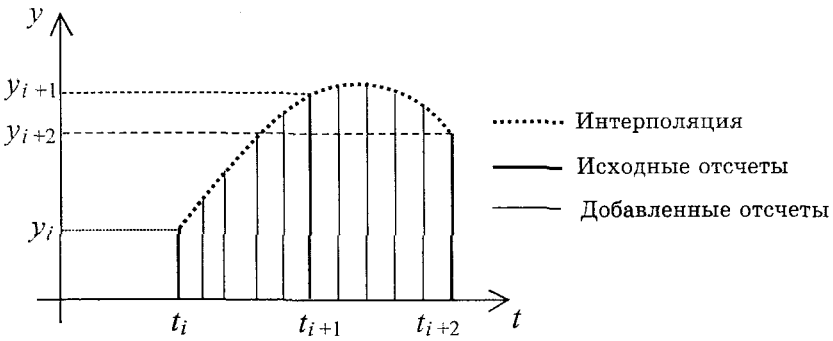


Рис. 2.4

После передискретизации цифровой сигнал с помощью ЦАП преобразуется в ступенчатый аналоговый сигнал. Этот ступенчатый сигнал представляет собой сумму ожидаемого выходного сигнала и так называемого паразитного сигнала дискретизации. Паразитный сигнал имеет малую амплитуду и очень высокую частоту. Выходной сигнал ЦАП пропускается через *пропускающий фильтр низких частот*, который подавляет высокочастотные составляющие сигнала. Такой технический прием позволяет добиться качественной реконструкции формы аналогового сигнала при простой реализации электронных схем (см. рис. в Приложении).

Вопросы и задания

1. Каким образом происходит преобразование непрерывного звукового сигнала в дискретный цифровой код?
2. Перечислите достоинства и недостатки двух способов звукозаписи: импульсной модуляции и MIDI.
3. Можно ли записать с помощью синтезатора вокальные произведения?

4. Оцените информационный объем моноаудиофайла длительностью звучания 1 минута, если глубина квантования и частота дискретизации звукового сигнала равны соответственно 16 бит и 8 кГц.
5. Рассчитайте время звучания моноаудиофайла, если при 16-битном кодировании и частоте дискретизации 32 кГц его объем равен 700 Кбайт.

§ 2.6. Методы сжатия цифровой информации

Характерной особенностью большинства «классических» типов информации, с которыми работают люди, является их *избыточность*.

Пример 30. В русском языке существуют слова, однозначно прочитываемые в случае «потери» некоторых букв. Например, С_НТ_БРЬ, МОС_, Д_Р_ВО. Кроме того, имея текст на русском языке с «потерянными» буквами, человек, достаточно хорошо владеющий русским языком, может однозначно восстановить его. Например, вы без труда прочитаете предложение с пропущенными буквами «Дм_т_ий Ива_ов__ Менд_ле_в — в_л_ки_ рус_кий х_мик». Однако если это предложение будет читать иностранец, едва знающий русский язык и русскую историю, то он, скорее всего, не сможет его понять. Мы, носители русского языка, можем с легкостью восстановить окончания, пропущенные буквы в слогах, подобрать подходящие слова (из тех, что нам известны). А иностранцу просто не с чем сравнивать получаемую информацию. Таким образом, для носителя языка обычный связный текст на его родном языке содержит избыточную информацию — ее можно удалить, но смысл текста для него сохранится. □

Пример 31. Одним из примеров проявления избыточности информации и ее сжатия является использование математических обозначений. Например, сумма чисел 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40 записывается так:

$$2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20 + 22 + \\ + 24 + 26 + 28 + 30 + 32 + 34 + 36 + 38 + 40.$$

Нетрудно заметить, что эта сумма составлена из всех четных чисел от 2 до 40. Используя математическую но-

тацию, эту сумму можно записать намного короче: $\sum_{i=1}^{20} 2 \cdot i$. Итак, введя новые обозначения, мы сумели сократить запись математического выражения, а значит, *сжать информацию*. Однако если первая форма записи понятна любому, кто учил азы математики, то вторая понятна только тем, кто знает, как следует *интерпретировать* эту запись. □

Пример с суммой позволяет продемонстрировать еще одну особенность методов сжатия — *степень сжатия* входных данных принципиально зависит от самих сжимаемых данных. Так, чтобы воспользоваться обозначением суммы, надо найти формулу, выражающую складываемые числа через индекс суммирования. И хотя *какую-нибудь* формулу можно вывести для любой конечной последовательности (например, с помощью интерполяционной формулы Лагранжа), но вот *компактную* формулу подобрать удается далеко не всегда! Пример «плохой» последовательности:

$$102 + 103 + 105 + 107 + 111 + 113 + 117 + 119 + \\ + 123 + 129 + 131 + 137 + 141 + 143 + 147.$$

Каждое число в данном ряду получено прибавлением 100 к простому числу. Увы, общая формула для всех простых чисел до сих пор не найдена, равно как не доказано существование или отсутствие такой формулы (проблема «генератора простых чисел»). Впрочем, для конечного числа простых чисел или для простых чисел с особой структурой формулы-генераторы существуют.

Дадим формальное определение избыточности информации.

Определение 13. Кодирование информации является *избыточным*, если количество бит в полученном коде больше, чем это необходимо для однозначного декодирования исходной информации.

Степень избыточности зависит от типа информации: у видеоинформации она в несколько раз больше, чем у графической информации, а степень избыточности последней в несколько раз больше, чем текстовой информации. Вообще, степень избыточности естественной информации достаточно велика. Клод Шеннон, исследовав

избыточность литературного английского языка, установил, что она составляет около 50%. Это означает, что если в английском тексте наугад стереть около половины букв, то по оставшимся буквам человек, знающий английский язык, почти наверняка сможет восстановить текст. Избыточность языка выполняет очень важную функцию — обеспечивает человеку надежность ее восприятия, особенно в неблагоприятных условиях (просмотр телепередач при наличии помех, чтение текстов в условиях недостаточной освещенности, разговор в вагоне метро и т. п.).

Хранение и передача информации требуют определенной затраты ресурсов. Сжатие данных (перед сохранением или передачей по каналам связи) позволяет уменьшить эти затраты. На практике такие затраты можно даже выразить в денежном эквиваленте: например, на скачивание из Интернета сжатого музыкального файла потребуется меньше времени, а значит, придется меньше заплатить денег за пользование Интернетом.

Первые теоретические разработки в области сжатия информации относятся к концу 1940-х годов, когда была опубликована статья К. Шеннона «Математическая теория коммуникаций».

2.6.1. Алгоритмы обратимых методов

Все методы сжатия можно поделить на два больших класса. Одни алгоритмы только изменяют способ представления входных данных, приводя их к форме, которая более компактно кодируется. Такие алгоритмы принято называть *обратимыми*, поскольку для них существуют *обратные алгоритмы*, которые могут *точно* восстановить исходные данные из сжатого массива. Другие алгоритмы выделяют во входных данных существенную информацию и ту часть, которой можно пренебречь и удалить, после чего оставшиеся «существенные» данные подвергаются дальнейшему сжатию. Такие алгоритмы принято называть алгоритмами с *регулируемой потерей информации*.

Определение 14. Метод сжатия называется *обратимым*, если из данных, полученных при сжатии, можно точно восстановить исходный массив данных.

Обратимые методы можно применять для сжатия любых типов данных. Характерными форматами файлов, хранящих сжатую без потерь информацию, являются:

- GIF, TIF, PCX, PNG — для графических данных;
- AVI — для видеоданных;
- ZIP, ARJ, RAR, LZH, LH, CAB — для любых типов данных.

Существует достаточно много обратимых методов сжатия данных, однако в их основе лежит сравнительно небольшое количество теоретических алгоритмов, которые мы рассмотрим более подробно.

Метод упаковки

Суть *метода упаковки* заключается в уменьшении количества бит, отводимых для кодирования символов, если в сжимаемом массиве данных присутствует только небольшая часть используемого алфавита.

Пример 32. Допустим, входной текст состоит только из десятичной записи целых чисел и знаков «минус», разделенных пробелами (например, «280 – 1296 48 40 365 – 159 13 777»). Множество символов, встречающихся в таком тексте, состоит всего из 12 символов (цифры от «0» до «9», знак «–» (минус) и пробел). Для кодирования такого количества символов достаточно всего четырех бит, целого байта для этого много. Если упаковать коды данных символов в 4 бита (например, так: «0» → «0000», «1» → «0001», ... «9» → «1001», «–» → «1110», пробел → «1111»), то можно будет кодировать по два символа входного текста одним байтом в выходном массиве. В результате получим двукратное сжатие данных. Формат записи чисел, при котором число записывается в десятичной системе, а цифры числа кодируются 4-битовыми кодами, называется BCD-форматом (***Binary Coded Decimal***, или двоично-десятичная запись). BCD-формат нередко используется в программировании для хранения целых чисел, например в базах данных. □

Пример 33. Входной текст «КОЛ_ОКОЛО_КОЛОКОЛА» содержит всего 5 различных символов («К», «О», «Л», «А» и пробел), следовательно, каждый символ может быть закодирован тремя битами. Всего в исходном тек-

сте 18 символов, так что потребуется $18 \cdot 3 = 54$ бита. Округлив это значение с избытком до целого числа байт, получим размер сжатого массива — всего 7 байт. Коэффициент сжатия равен $18/7 = 2{,}(571428) \approx 2{,}6$.

Сжимаемый текст (указано, в какое место выходного массива попадает код каждого символа)

□

Одно из преимуществ метода упаковки заключается в том, что любой фрагмент сжатых данных можно распаковать, совершенно не используя предшествующие данные. Действительно, зная номер требуемого символа N и длину кодов символов M, можно вычислить местоположение кода символа в сжатом массиве данных:

- номер байта, в котором начинается код символа, вычисляется так: $L = [M \cdot N/8]$;
- номер первого бита кода (в пределах этого байта) K равен остатку от деления $M \cdot N$ на 8.

Метод упаковки дает хорошие результаты, только если множество используемых символов невелико. Например, если в тексте используются только прописные русские буквы и знаки препинания, то текст может быть сжат всего на 25%: 33 русские буквы плюс пробел и знаки препинания — итого около 40 символов. Для их кодирования достаточно 6 бит. При упаковке текст уменьшится до $\frac{6}{8} = \frac{3}{4}$ от первоначального объема.

Алгоритм Хаффмана

Слабое место метода упаковки заключается в том, что символы кодируются битовыми последовательностями одинаковой длины. Например, любой текст, состоящий только из двух букв «А» и «В», сжимается методом упаковки в восемь раз. Однако если к такому тексту добавить всего лишь одну букву, например «С», то степень сжатия сразу уменьшится вдвое, причем независимо от длины текста и количества добавленных символов «С»!

Улучшения степени сжатия можно достичь, кодируя часто встречающиеся символы короткими кодами, а редко встречающиеся — более длинными. Именно такова идея метода, опубликованного Д. Хаффманом (Huffman) в 1952 г.

Идея кодирования символов кодами переменной длины была высказана и теоретически проработана американскими учеными К. Шенноном и Р. М. Фано. Ими был предложен алгоритм построения эффективных сжимающих кодов переменной длины (алгоритм Шеннона—Фано), однако он в некоторых случаях строил неоптимальные коды. Алгоритм Хаффмана оказался простым, быстрым и оптимальным: среди алгоритмов, кодирующих каждый символ по отдельности и целым количеством бит, он обеспечивает наилучшее сжатие.

Д. Хаффман (1925–1999)

Дэвид Хаффман руководил кафедрой компьютерных наук Массачусетского технологического института (Massachusetts Institute of Technology — MIT). Хотя Хаффман больше известен как разработчик метода построения минимально-избыточных кодов, он внес важный вклад во множество областей, связанных с информатикой, в частности в электронику.

Алгоритм Хаффмана сжимает данные за два прохода: на первом проходе читаются все входные данные и подсчитываются *частоты встречаемости* всех символов. Затем по этим данным строится *дерево кодирования Хаффмана*, а по нему — коды символов. После этого, на втором проходе, входные данные читаются еще раз и при этом генерируется выходной массив данных.

Вычисление частот встречаемости — тривиальная задача. Разберем построение дерева кодирования Хаффмана.

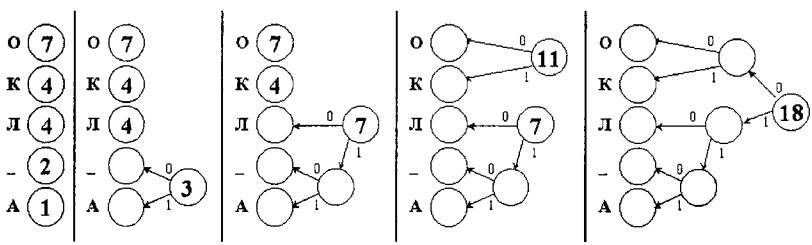
Алгоритм построения дерева Хаффмана

1. Символы входного алфавита образуют список свободных узлов. Каждый узел имеет вес, равный количеству вхождений символа в исходное сообщение.
2. В списке выбираются два свободных узла с наименьшими весами.

3. Создается их узел-«родитель» с весом, равным сумме их весов, он соединяется с «детьми» дугами.
4. Одной дуге, выходящей из «родителя», ставится в соответствие бит 1, другой — бит 0.
5. «Родитель» добавляется в список свободных узлов, а двое его «детей» удаляются из этого списка.
6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева. △

Пусть у нас имеется список частот встречаемости всех символов исходного текста. Выпишем их вертикально в ряд в виде вершин будущего графа (см. пример 34). Выберем две вершины с наименьшими весами (они соответствуют символам с наименьшим количеством повторений). Объединим эти вершины — создадим новую вершину, от которой проведем ребра к выбранным вершинам с наименьшими весами, а вес новой вершины зададим равным сумме их весов. Расставим на ребрах графа числа 0 и 1, например на каждом верхнем ребре 0, а на каждом нижнем — 1. Чтобы выбранные вершины более не просматривались, мы сотрем их веса (это аналог удаления вершин из списка). Продолжим выполнять объединение вершин, каждый раз выбирая пару вершин с наименьшими весами, до тех пор пока не останется одна вершина. Очевидно, что вес этой вершины будет равен длине сжимаемого массива данных.

Пример 34. Построение дерева Хаффмана и префиксных кодов для текста «КОЛ_ОКОЛО_КОЛОКОЛА»:



Теперь для определения кода каждой конкретной буквы необходимо просто пройти от вершины дерева до этой буквы, выписывая нули и единицы по маршруту следования. В нашем примере символы получат следующие коды:

О	00
К	01
Л	10
пробел	110
А	111

После того как коды символов построены, остается сгенерировать сжатый массив данных, для чего надо снова прочесть входные данные и каждый символ заменить на соответствующий код.

В нашем случае непосредственно код текста будет занимать 39 бит, или 5 байт. Коэффициент сжатия равен 18/5 = 3,6. □

Для восстановления сжатых данных необходимо снова воспользоваться деревом Хаффмана, так как код каждого символа представляет собой путь в дереве Хаффмана от вершины до конечного узла дерева, соответствующего данному символу. Общая схема процесса восстановления такова: специальный маркер устанавливается в вершину дерева Хаффмана, и сжатый массив данных читается побитово. Если читаемый бит равен 0, то маркер перемещается из вершины по верхнему ребру, если 1, то по нижнему. Затем читается следующий бит, и маркер снова перемещается, и т. д., пока маркер не попадет в один из конечных узлов дерева. В восстанавливаемый массив записывается символ, которому соответствует этот конечный узел, маркер снова помещается в вершину дерева, и процесс повторяется.

Код Хаффмана является *префиксным*. Это означает, что код каждого символа не является началом кода какого-либо другого символа. Код Хаффмана однозначно восстановим, даже если не сообщается длина кода каждого переданного символа. Подробнее об этом вы узнаете в главе 5. Получателю пересылают только дерево Хаффмана в компактном виде, а затем входная последовательность кодов символов декодируется им самостоятельно без какой-либо дополнительной информации.

Алгоритм RLE

В основу алгоритмов RLE (англ. ***Run-Length Encoding*** — кодирование путем учета числа повторений) положен принцип выявления повторяющихся последовательностей данных и замены их простой структурой: повторя-

Общая схема алгоритма LZ77 такова (это не точное описание алгоритма):

- входные данные читаются последовательно, текущая позиция условно разбивает массив входных данных на прочитанную и непрочитанную части;
- для цепочки первых байтов непрочитанной части ищется наиболее длинное совпадение в прочитанной части. Если совпадение найдено, то составляется комбинация {смещение, длина}, где смещение указывает, на сколько байтов надо сместиться назад от текущей позиции, чтобы найти совпадение, а длина — это длина совпадения;
- если запись комбинации {смещение, длина} короче совпадения, то она записывается в выходной массив, а текущая позиция перемещается вперед (на длину совпадающей части);
- если совпадение не обнаружено или оно короче записи комбинации {смещение, длина}, то в выходной массив копируется текущий байт, текущая позиция перемещается вперед на 1, и анализ повторяется.

Пример 35. Фраза КОЛОКОЛ_ОКОЛО_КОЛОКОЛЬНИ закодируется алгоритмом LZ77 как КОЛО(−4,3)_(−5,4)О_(−14,7)ЬНИ. □

Общая схема алгоритма LZ78 такова (это не точное описание алгоритма):

- алгоритм во время сжатия текста создает специальный *словарь повторяющихся цепочек*, в словаре каждой цепочке соответствует короткий код;
- для цепочки первых байтов непрочитанной части ищется наиболее длинное совпадение в словаре. Код совпадения записывается в выходной массив, туда же заносится первый несовпавший символ, и текущая позиция перемещается вперед на длину совпадения + 1;
- в словарь добавляется новое слово: «совпадение» + + «несовпавший символ», и процесс повторяется до тех пор, пока не будет сжат весь входной массив.

Алгоритмы Лемпеля—Зива тем лучше сжимают текст, чем больше размер входного массива. Характерной особенностью обратных алгоритмов LZ77 и LZ78 является то, что, кроме самих сжатых данных, никакой

дополнительной информации им не требуется! Начав работать, эти алгоритмы по уже распакованной части восстанавливают информацию, необходимую для распаковки следующих частей сжатых данных. Для сравнения: в алгоритме Хаффмана вместе со сжатыми данными требуется сохранять дерево Хаффмана, иначе распаковка будет невозможна.

Поучительна история развития алгоритмов Лемпеля—Зива. Зив и Лемпель придумали плодотворные идеи сжатия, построили алгоритмы и провели теоретическое исследование их эффективности. Но поскольку опубликованные алгоритмы были очень неэффективно реализованы (т. е. запрограммированы), долгое время они не использовались на практике. Только спустя 6 лет, в 1984 году Терри Велч (Terry Welch) сумел существенно улучшить алгоритм LZ78. Эта модификация алгоритма получила название LZW, она широко используется в программах сжатия данных. Алгоритм LZ77 ждал своего часа целых десять лет — только в 1987 году появилась его высокоэффективная версия, которая работала в сотни (!) раз быстрее оригинального алгоритма. В настоящее время существует около полусотни модификаций обоих алгоритмов. Обобщенно все они называются методами сжатия со словарем. Эти алгоритмы оказались настолько быстры и эффективны, что сейчас занимают лидирующее место среди используемых на практике алгоритмов сжатия.

2.6.2. Методы сжатия с регулируемой потерей информации

Описанные выше алгоритмы являются обратимыми: любой массив входных данных при распаковке восстанавливается абсолютно точно. Поэтому обратимые алгоритмы можно применять для сжатия информации любого рода. Но, как оказалось, для аудио- и видеоинформации абсолютно точное восстановление вовсе необязательно. Проведенные в конце XX века исследования психофизиологических характеристик зрения и слуха обнаружили ряд особенностей человеческого восприятия, использование которых позволяет существенно увеличивать степень сжатия звуковой, графической и видеоинформации.

Например, было установлено, что глаз человека наиболее чувствителен к зеленому цвету, чувствительность к красному ниже примерно в 4 раза, а к синему — почти в 10 раз! А это означает, что на хранение информации о красной и синей составляющих цвета можно было бы отводить меньше бит. Но в большинстве форматов графических файлов это не так — цветовые компоненты кодируются одинаковым количеством бит. Этот пример показывает, что традиционные способы представления видеоинформации обладают очень большой степенью избыточности, при условии, что речь идет о воспроизведении видеоинформации для человека.

Для разработки и стандартизации эффективных методов сжатия аудио- и видеоинформации на рубеже 1980–1990-х годов были созданы Группа экспертов по фотографическим изображениям (***Joint Photographic Experts Group***, сокр. JPEG) и Группа экспертов по видеоизображениям (***Motion Picture Experts Group***, сокр. MPEG).

К середине 1990-х годов были разработаны специальные высокоэффективные методы сжатия аудио- и видеоинформации, учитывающие особенности человеческого слуха и зрения. Характерной чертой этих методов является возможность регулируемого удаления маловажной (для человеческого восприятия) информации. Поэтому такие алгоритмы сжатия обобщенно называют *алгоритмами с регулируемой потерей информации*. За счет удаления части информации удается добиться очень большой степени сжатия данных при субъективно незначительной потере качества аудио- и видеоданных. Алгоритмы с регулируемой потерей информации неуниверсальны, они не могут использоваться для сжатия любых данных, поскольку полное восстановление исходной информации *невозможно*.

Наиболее известными методами сжатия с регулируемой потерей информации являются:

- JPEG — метод сжатия графических данных;
- MPEG — группа методов сжатия видеоданных;
- MP3 — метод сжатия звуковых данных.

Эти методы непросты в реализации, в них используется достаточно сложный математический аппарат, выходящий за рамки школьной программы, поэтому далее будет приведено лишь обзорное описание методов.

Алгоритм JPEG

Алгоритм JPEG используется для сжатия статических изображений. Помимо сжимаемого изображения, алгоритму передается также желаемый коэффициент сжатия — этот параметр регулирует долю информации, которая будет удалена при сжатии.

Собственно сжатие JPEG осуществляется в несколько этапов: сперва цвета пикселей переводятся из RGB-представления в YUV-представление (в соответствующей ему цветовой модели YCbCr цвет представляется компонентами «яркость» Y, «цветоразность зеленый–красный» Cr и «цветоразность зеленый–синий» Cb). Затем в каждой второй строке и каждом втором столбце матрицы пикселей информация о цветовых компонентах Cb и Cr просто удаляется (!), что мгновенно уменьшает объем данных вдвое. Оставшиеся данные подвергаются специальной процедуре «сглаживания», при которой объем данных не изменяется, но потенциальная степень их сжимаемости резко увеличивается. На этом этапе учитывается желаемый коэффициент сжатия. Затем данные сжимаются алгоритмом Хаффмана.

Алгоритм JPEG способен упаковывать графические изображения в несколько десятков раз, при этом потери качества становятся заметными только при очень высоких коэффициентах сжатия.

Алгоритм MP3

Алгоритм MP3 (точное название MPEG-1 Layer 3) является частью стандарта MPEG и описывает сжатие аудиоинформации. Помимо сжимаемого звукового фрагмента алгоритму передается также желаемый *битрейт* (англ. *bitrate*) — количество бит, используемых для кодирования одной секунды звука. Этот параметр регулирует долю информации, которая будет удалена при сжатии.

Сжатие MP3 также осуществляется в несколько этапов: звуковой фрагмент разбивается на небольшие участки — *фреймы* (англ. *frames*), а в каждом фрейме звук разлагается на составляющие звуковые колебания, которые в физике называют гармониками. С точки зрения математики, звук разлагается на группу синусоидальных колебаний с разными частотами и амплитудами. Затем начинается психоакустическая обработка —

удаление маловажной для человеческого восприятия звуковой информации, при этом учитываются различные особенности слуха. Желаемый битрейт определяет, какие эффекты будут учитываться при сжатии, а также количество удаляемой информации. На последнем этапе оставшиеся данные сжимаются алгоритмом Хаффмана.

Алгоритм MP3 позволяет сжимать звуковые файлы в несколько раз. При этом даже самый большой битрейт 320 Кбит/с стандарта MP3 обеспечивает четырехкратное сжатие аудиоинформации по сравнению с форматом Audio CD, при таком же субъективном качестве звука. Формат MP3 стал стандартом де-факто для распространения музыкальных файлов через Интернет.

Алгоритмы MPEG

Как уже упоминалось, MPEG — это целое семейство методов сжатия видеоданных. В них используется очень большое количество приемов сжатия, даже краткое перечисление которых заняло бы несколько страниц. Они опираются на несколько базовых идей, а различаются конкретной реализацией алгоритмов.

Одна из основных идей сжатия видео — метод «опорного кадра» — заключается в том, чтобы сохранять не целиком кадры, а только изменения кадров. Например, в фильме есть сцена беседы героев в комнате. При этом от кадра к кадру меняются только выражения лиц, а большая часть изображения неподвижна. Закодировав первый кадр сцены и отличия остальных ее кадров от первого, можно получить очень большую степень сжатия.

Еще один способ уменьшения кодируемой информации заключается в том, чтобы быстро сменяемые участки изображения кодировать с качеством, которое намного ниже качества статичных участков, — человеческий глаз не успевает рассмотреть их детально.

Кроме того, формат MPEG позволяет сохранять в одном файле несколько так называемых потоков данных. Так, в основном потоке можно сохранить фильм, в другом — логотип, в третьем — субтитры, и т. д. Потоки данных накладываются друг на друга только при воспроизведении. Такой способ позволяет, например, хранить субтитры в виде текста вместо изображений букв, логотип сохранить всего один раз, а не в каждом кадре, и т. п.

Разновидности формата MPEG отличаются друг от друга по возможностям, качеству воспроизводимого изображения и максимальной степени сжатия:

- MPEG-1 — использовался в первых Video CD (VCD-I);
- MPEG-2 — используется в DVD и Super Video CD (SVCD, VCD-II);
- MJPEG — формат сжатия видео, в котором каждый кадр сжимается по методу JPEG;
- MPEG-4 — популярный эффективный формат сжатия видео;
- DivX, XviD — улучшенные модификации формата MPEG-4.

Вопросы и задания

1. Для метода упаковки подсчитайте коэффициент сжатия текста, содержащего только прописные английские буквы, пробелы и знаки препинания (точка, запятая, дефис).
2. Для метода упаковки подсчитайте коэффициент сжатия текста, содержащего прописные и строчные русские буквы, пробелы, цифры и знаки препинания (точка, запятая, дефис).
3. Приведите примеры алгоритмов сжатия с потерей и без потери информации.
4. Что произойдет, если в упакованном методом RLE сообщении пропустить один байт?
5. Какова длина последовательности, после кодирования которой методом RLE получилось следующее:
   11111111 11111111 11000000 00000001 00000010 11111111 00000000 ?
6. Как надо поступить при RLE-кодировании, если количество идущих подряд одинаковых байтов больше 127 и не помещается в 7 разрядов?
7. Постройте дерево Хаффмана и выпишите коды символов для сообщения
   AHFBHCEHEHCEAHDCEEHHHCHHHDEGHGGEHCHH.

Заключение

В этой главе мы подробно рассмотрели способы компьютерного представления числовой информации, которая была первым и долгое время оставалась единственным

видом информации, обрабатываемой на компьютере. Вы узнали о способах представления целых и вещественных чисел, познакомились с понятием дополнительного кода и, надеемся, сможете объяснить, почему целые отрицательные числа представляются в такой «искусственной» форме.

На страницах этой главы мы познакомили вас с достоинствами и недостатками форматов с плавающей и с фиксированной запятой, а также с особенностями реализации целочисленной и вещественной арифметик в ограниченном числе разрядов.

Вы узнали о способах кодирования текстовой информации, познакомились с основными принципами дискретизации графической и звуковой информации, получили представление об алгоритмах сжатия различных видов информации.

Мы постарались показать вам, насколько информатика связана с другими науками: физикой, биологией, колориметрией и, конечно, математикой.

Компьютер называют универсальной машиной для обработки информации, при этом подчеркивают универсальность двоичного кодирования информации. Но давайте задумаемся, всякую ли информацию человек научился представлять в двоичном коде и всякую ли информацию может обрабатывать компьютер? Что вы можете сказать, например, об осязательной информации? Как сохранить в двоичном коде всевозможные запахи? Этот пример говорит о том, что мы находимся еще в самом начале эры цифровых технологий.

Глава 3

Введение в алгебру логики

Я, по крайней мере, думал, что противоречить друг другу могут только высказывания, поскольку они через умозаключения ведут к новым высказываниям, и мне кажется, что мнение, будто сами факты и события могут оказаться в противоречии друг с другом, является классическим примером бессмыслицы.

Д. Гильберт

Алгебра логики является частью, разделом бурно развивающейся сегодня науки — *дискретной математики*. Дискретная математика занимается изучением свойств структур конечного характера, которые возникают как внутри математики, так и в ее приложениях. Заметим, что классическая математика, в основном, занимается изучением свойств объектов непрерывного характера, хотя само деление математики на классическую и дискретную в значительной мере условно, поскольку между ними происходит активная циркуляция идей и методов, часто возникает необходимость исследовать модели, обладающие как дискретными, так и непрерывными свойствами. К числу структур, изучаемых дискретной математикой, могут быть отнесены конечные группы, конечные графы, математические модели преобразователей информации типа конечных автоматов или машин Тьюринга и др.

Математический аппарат алгебры логики широко используется в информатике, в частности, в таких ее разделах, как проектирование ЭВМ, теория автоматов, теория алгоритмов, теория информации, целочисленное программирование и т. д.

§ 3.1. Алгебра логики. Понятие высказывания

Алгебра логики изучает свойства функций, у которых и аргументы, и значения принадлежат заданному двухэлементному множеству (например, {0, 1}). Иногда вместо термина «алгебра логики» употребляют термин «двузначная логика».

Дж. Буль (1815–1864)

Отцом алгебры логики по праву считается английский математик XIX столетия Джордж Буль. Именно он построил один из разделов формальной логики в виде некоторой «алгебры», аналогичной алгебре чисел, но не сводящейся к ней. *Алгебра* в широком смысле этого слова — наука об общих операциях, аналогичных сложению и умножению, которые могут выполняться не только над числами, но и над другими математическими объектами. Существуют алгебры натуральных чисел, многочленов, векторов, матриц, множеств и т. д.

Большой вклад в становление и развитие алгебры логики внесли Августус де Морган (1806–1871), Уильям Стенли Джевонс (1835–1882), Платон Сергеевич Порецкий (1846–1907), Чарлз Сандерс Пирс (1839–1914), Андрей Андреевич Марков (1903–1979), Андрей Николаевич Колмогоров (1903–1987) и др.

Долгое время алгебра логики была известна достаточно узкому классу специалистов. Прошло почти 100 лет со времени создания алгебры логики Дж. Булем, прежде чем в 1938 году выдающийся американский математик и инженер Клод Шеннон (1916–2001) показал, что алгебра логики применима для описания самых разнообразных процессов, в том числе функционирования релейно-контактных и электронно-ламповых схем.

Исследования в алгебре логики тесно связаны с изучением высказываний (хотя высказывание — предмет изучения *формальной логики*). С помощью высказываний мы устанавливаем свойства, взаимосвязи между объектами. Высказывание истинно, если оно адекватно отображает эту связь, в противном случае оно ложно.

Примерами высказываний на естественном языке являются предложения «Сегодня светит солнце» или «Трава растет». Каждое из этих высказываний характеризует свойства или состояние конкретного объекта (в нашем примере погоды и окружающего мира). Каждое из этих высказываний несет значение «истина» или «ложь».

Однако определение истинности высказывания далеко не простой вопрос. Например, высказывание «Число $1 + 2^{2^5} = 4\,294\,967\,297$ — простое», принадлежащее Ферма (1601–1665), долгое время считалось истинным, пока в 1732 году Эйлер (1707–1783) не доказал, что оно ложно. В целом, обоснование истинности или ложности простых высказываний решается вне алгебры логики. Например, истинность или ложность высказывания «Сумма углов треугольника равна 180°» устанавливается геометрией, причем в геометрии Евклида это высказывание является истинным, а в геометрии Лобачевского — ложным.

Что же является высказыванием в формальной логике?

Определение 1. *Высказывание* — это языковое образование, в отношении которого имеет смысл говорить о его истинности или ложности (*Аристотель*).

Это словесное определение, не являющееся математически точным, только на первый взгляд кажется удовлетворительным. Оно отсылает проблему определения высказывания к проблеме определения истинности или ложности данного языкового образования. Если рассматривать в качестве высказываний любые утвердительные предложения, то это быстро приводит к парадоксам и противоречиям. Например, предложению «*Это предложение является ложным*» невозможно приписать никакого значения истинности без того, чтобы не получить противоречие. Действительно, если принять, что предложение истинно, то это противоречит его смыслу. Если же принять, что предложение ложно, то отсюда следует, что предложение на самом деле истинно. Как видно, этому предложению осмысленно нельзя приписать какое-либо значение истинности, следовательно, оно не является высказыванием.

Причина этого парадокса лежит в *структуре* построения указанного предложения: *оно ссылается на свое собственное значение*. С помощью определенных ограничений на допустимые формы высказываний могут быть устранены такие ссылки на себя и, следовательно, устранены возникающие отсюда парадоксы.

Определение 2. *Высказывание* называется *простым* (*элементарным*), если никакая его часть не является высказыванием.

Высказывания могут выражаться с помощью математических, физических, химических и прочих знаков. Из двух числовых выражений можно составить высказывание, соединив их знаком равенства или неравенства. Сами числовые выражения высказываниями не являются. Не являются высказываниями и равенства или неравенства, содержащие переменные. Например, предложение «$x < 12$» становится высказыванием при замене переменной каким-либо конкретным значением. Предложения типа «$x < 12$» называют *предикатами*.

Алгебра логики отвлекается от смысловой содержательности высказываний. Мы можем договориться, что абсурдное по смыслу высказывание «Крокодилы летают» является истинным, и с этим значением высказывания будем работать. Вопрос о том, летают крокодилы

или нет, может волновать зоологов, но никак не математиков: им этот потрясающий факт безразличен. Введение таких ограничений дает возможность изучать высказывания алгебраическими методами, позволяет ввести операции над элементарными высказываниями и с их помощью строить и изучать составные высказывания. В информатике для точного определения понятия высказывания строятся ограниченные системы форм высказываний (формальный язык), которые используются при описании алгоритмических языков, в информационных системах, для строгого формального описания алгоритмов и т. д.

Алгебра логики изучает строение (форму, структуру) сложных логических высказываний и способы установления их истинности с помощью алгебраических методов.

Вопросы и задания

1. Из данных предложений выберите те, которые являются высказываниями, и обоснуйте свой выбор.
   а) Коля спросил: «Который час?»
   б) Как пройти в библиотеку?
   в) Картины Пикассо слишком абстрактны.
   г) Решение задачи — информационный процесс.
   д) Число 2 является делителем числа 7 в некоторой системе счисления.
2. Объясните, почему формулировка любой теоремы является высказыванием.
3. Приведите по два примера истинных и ложных высказываний из биологии, географии, информатики, истории, математики, литературы.
4. Из данных высказываний выберите истинные.
   а) Город Джакарта — столица Индонезии.
   б) Решение задачи — информационный процесс.
   в) Число 2 является делителем числа 7 в некоторой системе счисления.
   г) Меню в программе — это список возможных вариантов.
   д) Для всех x из области определения выражения $\sqrt{x+1}$ верно, что $x + 2 > 0$.

е) Сканер — это устройство, которое может напечатать на бумаге то, что изображено на экране компьютера.
ж) II + VI > VIII.
з) Мышь — устройство ввода информации.

5. В приведенных предложениях вместо многоточий поставьте подходящие по смыслу слова «необходимо», «достаточно», «необходимо и достаточно». Помните, что получившиеся высказывания должны быть истинными.
1) Для того чтобы число делилось на 4, ... чтобы оно было четным.
2) Чтобы число делилось на 3, ... чтобы оно делилось на 9.
3) Для того чтобы число делилось на 10, ... чтобы оно оканчивалось нулем.
4) Чтобы произведение двух чисел равнялось нулю, ... чтобы каждое из них равнялось нулю.
5) Чтобы произведение двух чисел равнялось нулю, ... чтобы хоть одно из них равнялось нулю.
6) Чтобы умножить сумму нескольких чисел на какое-нибудь число, ... каждое слагаемое умножить на это число и произведения сложить.
7) Чтобы произведение нескольких чисел разделить на какое-нибудь число, ... разделить на это число только один из сомножителей и полученное частное умножить на остальные сомножители.
8) Для того чтобы сумма двух чисел была четным числом, ... чтобы каждое из слагаемых было четным числом.
9) Для того чтобы число делилось на 10, ... чтобы оно делилось на 5.
10) Для того чтобы число делилось на 6, ... чтобы оно делилось на 2 и на 3.
11) Для того чтобы число делилось на 12, ... чтобы оно делилось на 2 и на 3.
12) Чтобы четырехугольник был квадратом, ... чтобы все его стороны были равны.

§ 3.2. Логические операции. Таблицы истинности

Употребляемые в обычной речи связки «и», «или», «не», «если ..., то ...», «тогда и только тогда, когда ...» и т. п. позволяют из уже заданных высказываний строить новые сложные высказывания. Истинность или

ложность получаемых таким образом высказываний зависит от истинности и ложности исходных высказываний и соответствующей *трактовки связок как логических операций над высказываниями*. Для обозначения истинности, как правило, используются символы «И» и «1», а для обозначения ложности — символы «Л» и «0».

Логическая операция полностью может быть описана таблицей истинности, указывающей, какие значения принимает сложное высказывание при всех возможных значениях простых высказываний.

В алгебре логики логические связки и соответствующие им логические операции имеют специальные названия и обозначаются следующим образом:

Логическая связка	Названия логической операции	Обозначения
не	Отрицание, инверсия	$\bar{}$, $\rceil$, $\neg$
и, а, но, хотя	Конъюнкция, логическое умножение	&, $\cdot$, $\wedge$
или	Дизъюнкция, нестрогая дизъюнкция, логическое сложение	$\vee$, +
либо	Разделительная (строгая) дизъюнкция, исключающее ИЛИ, сложение по модулю 2	$\oplus$, Δ
если ..., то	Импликация, следование	$\Rightarrow$, $\rightarrow$
тогда и только тогда, когда	Эквивалентность, эквиваленция, равнозначность	$\Leftrightarrow$, $\sim$, $\equiv$, $\leftrightarrow$

Введем перечисленные логические операции формальным образом[1].

3.2.1. Высказывание, составленное из двух высказываний путем объединения их связкой «и», называется *конъюнкцией* или *логическим умножением*. Высказывая конъюнкцию, мы утверждаем, что выполняются оба события, о которых идет речь в составляющих высказываниях. Например, сообщая:

{*Ивановы привезли на зиму уголь и закупили дрова на растопку камина*},

мы выражаем в одном высказывании свое убеждение в том, что произошли оба этих события.

[1] Это тем более необходимо, потому что связки, употребляемые в речи, неоднозначны.

Определение 3. *Конъюнкция* — логическая операция, ставящая в соответствие каждым двум элементарным высказываниям новое высказывание, являющееся истинным тогда и только тогда, когда оба исходных высказывания истинны[1]. Логическая операция конъюнкция определяется следующей таблицей, которую называют *таблицей истинности*:

p	q	$p \,\&\, q$
0	0	0
0	1	0
1	0	0
1	1	1

Рассмотрим два высказывания p = {*Завтра будет мороз*} и q = {*Завтра будет идти снег*}. Очевидно, новое высказывание $p \,\&\, q$ = {*Завтра будет мороз, и завтра будет идти снег*} истинно только в том случае, когда одновременно истинны высказывания p и q, а именно, когда истинно, что завтра будет и мороз, и снег. Высказывание $p \,\&\, q$ будет ложно во всех остальных случаях: будет идти снег, но будет оттепель (т. е. не будет мороза); мороз будет, а снег не будет идти; не будет мороза, и снег не будет идти.

В русском языке конъюнкции соответствует не только союз «и», но и другие речевые обороты, например связки «а» или «но».

3.2.2. Высказывание, состоящее из двух высказываний, объединенных связкой «или», называется *дизъюнкцией* или *логическим сложением*, *нестрогой дизъюнкцией*. В высказываниях, содержащих связку «или», указывается на существование двух возможных событий, из которых хотя бы одно должно быть осуществлено. Например, сообщая:

{*Петя читает книгу или пьет чай*},

мы имеем в виду, что хотя бы что-либо одно Петя делает. При этом Петя может одновременно читать книгу и пить чай. И в этом случае дизъюнкция будет истинна.

[1] Это определение легко распространяется на случай n высказываний ($n > 2$, n — натуральное число).

Определение 4. *Дизъюнкция* — логическая операция, которая каждым двум элементарным высказываниям ставит в соответствие новое высказывание, являющееся ложным тогда и только тогда, когда оба исходных высказывания ложны[1]. Логическая операция дизъюнкция определяется следующей таблицей истинности:

p	q	$p \vee q$
0	0	0
0	1	1
1	0	1
1	1	1

Дизъюнкция истинна, когда хотя бы одно из двух образующих ее высказываний истинно.

Рассмотрим два высказывания p = {*Колумб был в Индии*} и q = {*Колумб был в Египте*}. Очевидно, новое высказывание $p \vee q$ = {*Колумб был в Индии или Колумб был в Египте*} истинно как в случае, если Колумб был в Индии, но не был в Египте, так и в случае, если он не был в Индии, но был в Египте, а также в случае, если он был и в Индии, и в Египте. Но это высказывание будет ложно, если Колумб не был ни в Индии, ни в Египте.

Союз «или» может применяться в речи и в другом, «исключающем» смысле. Тогда он соответствует другому высказыванию — разделительной, или строгой дизъюнкции.

3.2.3. Высказывание, образованное из двух высказываний, объединенных связкой «либо» (точнее: «либо только ..., либо только ...»), называется *разделительной* (*строгой*) *дизъюнкцией*, *исключающим ИЛИ*, *сложением по модулю 2*.

В отличие от обычной дизъюнкции (связка «или»), в высказывании, являющемся разделительной дизъюнкцией, мы утверждаем, что произойдет только одно событие из двух. Например, сообщая:

{*Петя сидит на трибуне А либо на трибуне Б*},

мы утверждаем, что Петя сидит либо только на трибуне А, либо только на трибуне Б.

[1] Это определение, как и предыдущее, распространяется на случай n высказываний ($n > 2$, n — натуральное число).

Определение 5. *Строгая*, или *разделительная дизъюнкция* — логическая операция, ставящая в соответствие двум элементарным высказываниям новое высказывание, являющееся истинным тогда и только тогда, когда ровно одно из двух высказываний является истинным. Логическая операция разделительная дизъюнкция определяется следующей таблицей истинности:

p	q	$p \oplus q$
0	0	0
0	1	1
1	0	1
1	1	0

Рассмотрим два высказывания p = {*Кошка охотится за мышами*} и q = {*Кошка спит на диване*}. Очевидно, что новое высказывание $p \oplus q$ истинно только в двух случаях: когда кошка охотится за мышами или когда кошка мирно спит. Это высказывание будет ложно, если кошка не делает ни того, ни другого, т. е. когда оба события не происходят. Но это высказывание будет ложным и тогда, когда предполагается, что оба события будут происходить одновременно.

Вопрос. *В сложном высказывании использована связка «или». Какая это дизъюнкция: нестрогая или строгая?*

Ответ. В логике связкам «либо» и «или» придается разное значение, однако в русском языке связку «или» иногда употребляют вместо связки «либо». Чтобы определить значение связки «или», нужно проанализировать содержание высказывания по смыслу. Например, анализ высказывания {*Петя сидит на трибуне А или на трибуне Б*} однозначно укажет на логическую операцию разделительная дизъюнкция, так как человек не может находить в двух разных местах одновременно. □

3.2.4. Предложение, образованное из двух предложений, объединенных связкой «если ..., то ...», в грамматике называется условным предложением, а в логике такое высказывание называется *импликацией*.

Импликацию мы используем тогда, когда хотим показать, что некоторое событие зависит от другого события. Например, пусть человек сказал: *«Если завтра будет хорошая погода, то я пойду гулять»*. Здесь $p = \{$*Завтра будет хорошая погода*$\}$ и $q = \{$*Я пойду гулять*$\}$. Ясно, что человек окажется лжецом лишь в том случае, если погода действительно окажется хорошей, а гулять он не пойдет. Если же погода будет плохой, то независимо от того, пойдет он гулять или нет, во лжи его нельзя обвинить: обещание пойти гулять он давал лишь при условии, что погода будет хорошей.

Определение 6. *Импликация* — логическая операция, ставящая в соответствие двум элементарным высказываниям новое высказывание, являющееся ложным тогда и только тогда, когда *условие* (*посылка*) — истинно, а *следствие* (*заключение*) — ложно. Логическая операция импликация задается следующей таблицей истинности:

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

Мы видим, что импликация заведомо истинна, если условие p ложно. Другими словами, из неверного условия может следовать все, что угодно. Например, высказывание $\{$*Если 2 > 3, то крокодилы летают*$\}$ является истинным.

Подавляющее число зависимостей между событиями можно описать с помощью импликации.

Пример 1. Истинным высказыванием $\{$*Если на каникулах мы поедем в Петербург, то посетим Исаакиевский собор*$\}$ мы утверждаем, что в случае приезда на каникулах в Петербург Исаакиевский собор мы посетим обязательно. □

В соответствии с определением импликации истинны следующие высказывания:

а) $\{$*Если* $2 \times 2 = 4$, *то через Смоленск протекает Днепр*$\}$.

б) {*Если через Смоленск протекает Енисей, то* $2 \times 2 = 4$}.
в) {*Если через Смоленск протекает Енисей, то* $2 \times 2 = 5$}.
г) {*Если все ученики класса напишут контрольную работу по физике на отлично, то слоны в Африке живут*}.
д) {*Если через Смоленск протекает Енисей, то все ученики класса напишут контрольную работу по физике на отлично*}.

Отметим, что высказывания г) и д) являются истинными импликациями и в том случае, если высказывание {*все ученики класса напишут контрольную работу по физике на отлично*} является истинным, и в том случае, если оно является ложным.

Следующие две импликации являются ложными, так как в них посылки истинны, а заключения ложны:

е) {*Если* $2 \times 2 = 4$, *то через Смоленск протекает Енисей*}.
ж) {*Если через Смоленск протекает Днепр, то Луна сделана из теста*}.

Импликация, образованная из высказываний A и B, может быть записана на естественном языке при помощи следующих предложений: «Если A, то B», «Из A следует B», «A влечет B».

Может показаться странным, что высказывание «Если A, то B» всегда истинно, если посылка (высказывание A) ложна. Но для математика это вполне естественно. В самом деле, исходя из ложной посылки, можно путем верных рассуждений получить как истинное, так и ложное утверждение.

Допустим, что $1 = 2$, тогда и $2 = 1$. Складывая эти равенства, получим $3 = 3$, т. е. из ложной посылки путем тождественных преобразований мы получили истинное высказывание.

Большинство математических теорем являются импликациями. Однако те импликации, в которых посылки (условия) и заключения (следствия) являются предложениями без взаимной (по существу) связи, не могут играть в науке важной роли. Они являются бесплодными предложениями, так как не ведут к выводам более глубокого содержания.

В математике ни одна теорема не является импликацией, в которой условие и заключение не были бы связаны по содержанию. Достаточно часто в математических теоремах импликации формулируются в виде *только необходимого* или *только достаточного условия.*

3.2.5. Высказывание, образованное из двух высказываний при помощи связки «тогда и только тогда, когда», в логике называется *эквивалентностью.* Эквивалентность используется в тех случаях, когда необходимо выразить взаимную обусловленность. Например, сообщая:

{*Я получу паспорт тогда и только тогда, когда мне исполнится 14 лет*},

человек утверждает не только то, что после того, как ему исполнится 14 лет, он получит паспорт, но и то, что паспорт он сможет получить только после того, как ему исполнится 14 лет.

Определение 7. *Эквивалентность* — логическая операция, ставящая в соответствие двум элементарным высказываниям новое, являющееся истинным тогда и только тогда, когда оба исходных высказывания одновременно истинны или одновременно ложны. Логическая операция эквивалентность задается следующей таблицей истинности:

p	q	$p \sim q$
0	0	1
0	1	0
1	0	0
1	1	1

Рассмотрим возможные значения сложного высказывания, являющегося эквивалентностью: {*Учитель утверждает, что 5 в четверти ученику он поставит тогда и только тогда, когда ученик получит 5 на зачете*}.

1) Ученик получил 5 на зачете и 5 в четверти, т. е. учитель выполнил свое обещание, следовательно, высказывание является истинным.
2) Ученик не получил на зачете 5, и учитель не поставил ему 5 в четверти, т. е. учитель свое обещание сдержал, высказывание является истинным.
3) Ученик не получил на зачете 5, но учитель поставил ему 5 в четверти, т. е. учитель свое обещание не сдержал, высказывание является ложным.

4) Ученик получил на зачете 5, но учитель не поставил ему 5 в четверти, т. е. учитель свое обещание не сдержал, высказывание является ложным.

В математических теоремах эквивалентность выражается связкой «необходимо и достаточно».

3.2.6. Рассмотренные выше операции были двуместными (бинарными), т. е. выполнялись над двумя операндами (высказываниями). В алгебре логики определена и широко применяется и одноместная (унарная) операция *отрицание*.

Определение 8. *Отрицание* — логическая операция, которая каждому элементарному высказыванию ставит в соответствие новое высказывание, значение которого противоположно исходному. Логическая операция отрицание задается следующей таблицей истинности:

p	$\overline{p}$
0	1
1	0

В русском языке для построения отрицания используется связка «неверно, что». Хотя связка «неверно, что» и не связывает двух каких-либо высказываний в одно, она трактуется логиками как логическая связка, поскольку, поставленная перед произвольным высказыванием, образует из него новое.

Пример 2. Отрицанием высказывания {*У меня дома есть компьютер*} будет высказывание {*Неверно, что у меня дома есть компьютер*} или, что в русском языке то же самое, {*У меня дома нет компьютера*}. □

Пример 3. Отрицанием высказывания {*Я не знаю корейского языка*} будет высказывание {*Неверно, что я не знаю корейского языка*} или {*Я знаю корейский язык*}. □

Пример 4. Отрицанием высказывания {*Все юноши 11-х классов — отличники*} является высказывание {*Неверно, что все юноши 11-х классов — отличники*} или {*Не все юноши 11-х классов — отличники*} или другими словами, {*Некоторые юноши 11-х классов — не отличники*}. □

На первый взгляд кажется, что построить отрицание к заданному высказыванию достаточно просто. Однако это не так.

Пример 5. Высказывание {*Все юноши 11-х классов — не отличники*} не является отрицанием высказывания {*Все юноши 11-х классов — отличники*}. Объясняется это следующим образом. Высказывание {*Все юноши 11-х классов — отличники*} ложно. Отрицанием к ложному высказыванию должно быть высказывание, являющееся истинным. Но высказывание {*Все юноши 11-х классов — не отличники*} не является истинным, так как среди одиннадцатиклассников есть как отличники, так и не отличники. □

Пример 6. Для высказывания {*На стоянке стоят красные «Жигули»*} следующие предложения отрицаниями являться не будут:

1) {*На стоянке стоят не красные «Жигули»*};
2) {*На стоянке стоит белый «Мерседес»*};
3) {*Красные «Жигули» стоят не на стоянке*}.

Попробуйте этот пример разобрать самостоятельно. □

Проанализировав приведенные примеры, можно вывести полезное правило.

Правило построения отрицания к простому высказыванию

При построении отрицания к простому высказыванию либ используется речевой оборот «неверно, что», либо отрицание строится к сказуемому, тогда к сказуемому добавляется частица «не», при этом слово «все» заменяется на «некоторые» и наоборот. △

В заключение приведем сводную таблицу истинности для введенных логических операций.

p	q	$\overline{p}$	$p \,\&\, q$	$p \vee q$	$p \oplus q$	$p \to q$	$p \sim q$
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

Вопросы и задания

1. В следующих высказываниях выделите простые, обозначив каждое их них буквой; запишите с помощью букв и знаков логических операций каждое составное высказывание.
   а) Число 376 четное и трехзначное.
   б) Зимой дети катаются на коньках или на лыжах.
   в) Новый год мы встретим на даче либо на Красной площади.
   г) Неверно, что Солнце движется вокруг Земли.
   д) Если 14 октября будет солнечным, то зима будет теплой.
   е) Земля имеет форму шара, который из космоса кажется голубым.
   ж) На уроке математики старшеклассники отвечали на вопросы учителя, а также писали самостоятельную работу.
   з) Если вчера было воскресенье, то Дима вчера не был в школе и весь день гулял.
   и) Если сумма цифр натурального числа делится на 3, то число делится на 3.
   к) Число делится на 3 тогда и только тогда, когда сумма цифр числа делится на 3.
2. Ниже приведена таблица, левая колонка которой содержит основные логические союзы (связки), с помощью которых в естественном языке строятся сложные высказывания. Заполните правую колонку таблицы названиями наиболее подходящих логических операций.

В естественном языке	В логике
и	
или	
Неверно, что	
хотя	
в том и только в том случае	
но	
а	
Если ..., то	
однако	
тогда и только тогда, когда	
Либо ..., либо	
необходимо и достаточно	

В естественном языке	В логике
Из ... следует ...	
... влечет ...	
... равносильно ...	
... необходимо ...	
... достаточно ...	

3. Постройте отрицания следующих высказываний.

а) Сегодня в театре идет опера «Евгений Онегин».
б) Каждый охотник желает знать, где сидит фазан.
в) Число 1 есть простое число.
г) Число 1 — составное.
д) Натуральные числа, оканчивающиеся цифрой 0, являются простыми числами.
е) Неверно, что число 3 не является делителем числа 198.
ж) Коля решил все задания контрольной работы.
з) Неверно, что любое число, оканчивающееся цифрой 4, делится на 4.
и) Во всякой школе некоторые ученики интересуются спортом.
к) Некоторые млекопитающие не живут на суше.

4. Являются ли отрицаниями друг друга следующие пары предложений?

а) Он — мой друг. Он — мой враг.
б) Большой дом. Небольшой дом.
в) Большой дом. Маленький дом.
г) $X > 2$. $X < 2$.

5. Пусть $p = \{$*Ане нравятся уроки математики*$\}$, а $q = \{$*Ане нравятся уроки химии*$\}$. Выразите следующие формулы на естественном языке.

а) $p \,\&\, q$; г) $p \vee q$; ж) $\overline{p \,\&\, q}$; к) $p \to \overline{q}$;
б) $\overline{p} \,\&\, q$; д) $p \vee \overline{q}$; з) $\overline{p \vee q}$; л) $p \to \overline{q}$;
в) $p \,\&\, \overline{q}$; е) $\overline{p} \vee \overline{q}$; и) $\overline{p \,\&\, \overline{q}}$; м) $\overline{p \to q}$.

6. В математических теоремах импликация выражается не только связкой «если ..., то». Высказывание «для того чтобы выполнялось A, достаточно, чтобы выполнялось B» соответствует импликации $(B \to A)$. Запишите в символике алгебры логики импликацию «для того чтобы выполнялось A, необходимо, чтобы выполнялось B».

§ 3.3. Логические формулы. Законы алгебры логики

Математики под словом «алгебра» подразумевают науку, которая изучает некие объекты и операции над ними. Например, школьная алгебра (алгебра действительных чисел) изучает действительные числа и операции над ними. Предметом же нашего изучения являются высказывания, операции над ними, а также логические функции. В предыдущих параграфах для обозначения высказываний мы использовали буквы. Как и в алгебре действительных чисел, введем следующие определения.

Определение 9. *Логической переменной* называется переменная, значением которой может быть любое высказывание. Логические переменные (далее «переменные») обозначаются латинскими буквами, иногда снабженными индексами, как обычные алгебраические переменные: x, y, x_1, y_1, x_k, y_n и т. п.

Понятие логической формулы является формализацией понятия сложного высказывания. Введем его индуктивно.

Определение 10. *Логической формулой* является:

1) любая логическая переменная, а также каждая из двух логических констант — 0 (ложь) и 1 (истина);
2) если A и B — формулы, то $\overline{B}$ и $A*B$ — тоже формулы, где знак «*» означает любую из логических бинарных операций.

Формулой является, например, следующее выражение: $(x \,\&\, y) \to z$. Каждой формуле при заданных значениях входящих в нее переменных приписывается одно из двух значений — 0 или 1.

Определение 11. Формулы A и B, зависящие от одного и того же набора переменных x_1, x_2, x_3, ..., x_n, называют *равносильными* или *эквивалентными*, если на любом наборе значений переменных x_1, x_2, x_3, ..., x_n они имеют одинаковые значения. Для обозначения равносильности формул используется знак равенства, например $A = B$.

В дальнейшем будет показано, что любую формулу можно преобразовать к равносильной ей, в которой используются только аксиоматически введенные операции &, $\vee$ и отрицание.

Для преобразования формул в равносильные важную роль играют следующие равенства, отражающие свойства логических операций, которые по аналогии с алгеброй вещественных чисел будем называть законами:

1) **законы коммутативности**
$x \,\&\, y = y \,\&\, x$,
$x \vee y = y \vee x$;
2) **законы ассоциативности**
$(x \,\&\, y) \,\&\, z = x \,\&\, (y \,\&\, z)$,
$(x \vee y) \vee z = x \vee (y \vee z)$;
3) **законы поглощения (нуля и единицы)**
$x \vee 0 = x$, $x \,\&\, 1 = x$;
4) **законы дистрибутивности**
$x \,\&\, (y \vee z) = (x \,\&\, y) \vee (x \,\&\, z)$,
$x \vee (y \,\&\, z) = (x \vee y) \,\&\, (x \vee z)$;
5) **закон противоречия**
$x \,\&\, \overline{x} = 0$;
6) **закон исключенного третьего**
$x \vee \overline{x} = 1$;
7) **законы идемпотентности**[1]
$x \,\&\, x = x$, $x \vee x = x$;
8) **закон двойного отрицания**
$\overline{\overline{x}} = x$;
9) **законы де Моргана**
$\overline{x \,\&\, y} = \overline{x} \vee \overline{y}$,
$\overline{x \vee y} = \overline{x} \,\&\, \overline{y}$;
10) **законы поглощения**
$x \vee (x \,\&\, y) = x$,
$x \,\&\, (x \vee y) = x$.

Любой из этих законов может быть легко доказан с помощью таблиц истинности.

Пример 7. Докажем первый закон де Моргана с использованием таблиц истинности. Построим таблицу истинности для левой и правой частей закона.

---

[1] От латинских слов *idem* — тот же самый и *potens* — сильный; дословно — равносильный.

x	y	$x \,\&\, y$	$\overline{x \,\&\, y}$	$\bar{x}$	$\bar{y}$	$\bar{x} \vee \bar{y}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Так как результирующие столбцы совпали, то формулы, стоящие в левой и правой частях закона, равносильны. □

Любой из законов алгебры логики может быть доказан путем логических рассуждений.

Пример 8. Докажем первый закон поглощения $x \vee (x \,\&\, y) = x$ путем логических рассуждений. Для этого достаточно показать, что если правая часть истинна, то и левая часть истинна, и что если левая часть истинна, то и правая часть истинна.

Пусть истинна правая часть, т. е. $x = 1$, тогда в левой части дизъюнкция $x \vee (x \,\&\, y)$ истинна по определению дизъюнкции. Пусть истинна левая часть. Тогда по определению дизъюнкции истинна или формула x, или формула $(x \,\&\, y)$, или обе эти формулы одновременно. Если x ложна, тогда $(x \,\&\, y)$ ложна, следовательно, x может быть только истинной. □

Еще одним способом вывода законов являются *тождественные преобразования*.

Пример 9. Первый закон поглощения можно вывести при помощи законов поглощения единицы и дистрибутивности:

$$x \vee (x \,\&\, y) = (x \,\&\, 1) \vee (x \,\&\, y) = x \,\&\, (1 \vee y) = x.$$ □

Определение 12. Формула A называется *тавтологией* (или тождественно истинной), если она истинна при любых значениях своих переменных.

Пример 10. Тавтологией является формула $x \vee \bar{x}$, выражающая закон исключенного третьего. □

В алгебре логики дизъюнкцию еще называют логическим сложением, а конъюнкцию — логическим умножением. Если продолжать аналогию между логическими и арифметическими операциями, то операция отрицания по некоторым характеристикам аналогична унарному минусу.

Для логических операций установлен следующий порядок вычислений: 1) отрицание; 2) конъюнкция; 3) дизъюнкция (строгая и нестрогая); 4) импликация и эквивалентность.

Поэтому при записи логических формул с использованием этих операций скобки требуется расставлять только для того, чтобы изменить порядок выполнения операций, фиксированный по умолчанию. Например, выражение $x \vee \overline{y}\ \&\ z$ трактуется как $x \vee (\overline{y}\ \&\ z)$.

Вопросы и задания

1. Какие из рассмотренных логических законов аналогичны законам алгебры чисел, а какие нет?
2. Докажите второй закон де Моргана с помощью таблиц истинности.
3. Рассмотрите два сложных высказывания:

   F_1 = {если одно слагаемое делится на 3 и сумма делится на 3, то и другое слагаемое делится на 3};

   F_2 = {если одно слагаемое делится на 3, а другое не делится на 3, то сумма не делится на 3}.

   Формализуйте эти высказывания, постройте таблицы истинности для каждой из полученных формул и убедитесь, что результирующие столбцы совпадают.
4. Формализуйте следующие высказывания и постройте для них таблицы истинности:

   F_1 = {если все стороны четырехугольника равны и один из его углов прямой, то этот четырехугольник является квадратом};

   F_2 = {если все стороны четырехугольника равны, а он не является квадратом, то один из его углов не является прямым}.
5. Для операций импликации, эквивалентности и разделительной дизъюнкции также может быть сформулирован ряд важных свойств. В частности, каждая из этих операций может быть выражена через конъюнкцию, дизъюнкцию и отрицание. Убедитесь в этом, доказав самостоятельно следующие соотношения:

   а) $a \to b = \overline{a} \vee b$;

   б) $a \sim b = a\ \&\ b \vee \overline{a}\ \&\ \overline{b}$;

в) $a \oplus b = \overline{a} \,\&\, b \vee a \,\&\, \overline{b}$.

г) $a \to b = \overline{b} \to \overline{a}$;

д) $a \sim b = (a \to b) \,\&\, (b \to a)$;

е) $a \oplus b = \overline{a \sim b}$.

6. Найдите x, если $\overline{(x \vee a)} \vee \overline{(x \vee \overline{a})} = b$.

7. Какие из следующих формул являются тавтологиями?

а) $\overline{a \,\&\, \overline{a}}$;

б) $a \to (b \to a)$;

в) $(a \,\&\, b) \to a$.

8. Логическая формула называется *тождественно ложной*, если она принимает значение 0 на всех наборах входящих в нее переменных. Упростите формулу $a \,\&\, (a \to b) \,\&\, (a \to \overline{b})$ и покажите, что она тождественно ложна.

§ 3.4. Методы решения логических задач

Исходными данными в логических задачах являются высказывания. Эти высказывания и взаимосвязи между ними бывают так сложны, что разобраться в них без использования специальных методов достаточно трудно.

Многие логические задачи связаны с рассмотрением нескольких конечных множеств и связей между их элементами. Для решения таких задач зачастую прибегают к помощи таблиц или графов, при этом успешность решения во многом зависит от удачно выбранной структуры таблицы или графа. Аппарат же алгебры логики позволяет построить формальный *универсальный* способ решения логических задач.

Формальный способ решения логических задач

1. Выделить из условия задачи элементарные (простые) высказывания и обозначить их буквами.
2. Записать условие задачи на языке алгебры логики, соединив простые высказывания в сложные с помощью логических операций.
3. Составить единое логическое выражение для всех требований задачи.
4. Используя законы алгебры логики, попытаться упростить полученное выражение и вычислить все его значения либо построить таблицу истинности для рассматриваемого выражения.

5. Выбрать решение — набор значений простых высказываний, при котором построенное логическое выражение является истинным.
6. Проверить, удовлетворяет ли полученное решение условию задачи. △

Рассмотрим, как можно использовать данный способ для решения задач.

Пример 11. Задача «Уроки логики».

На вопрос, кто из трех учащихся изучал логику, был получен ответ: «Если изучал первый, то изучал и второй, но неверно, что если изучал третий, то изучал и второй». Кто из учащихся изучал логику?

Решение. Обозначим через P_1, P_2, P_3 высказывания, состоящие в том, что соответственно первый, второй, третий учащийся изучали логику. Из условия задачи следует истинность высказывания $(P_1 \to P_2) \& (\overline{P_3 \to P_2})$.

Воспользуемся соотношением $a \to b = \overline{a} \vee b$ (см. § 3.3, задание 5) и упростим исходное высказывание:

$$(P_1 \to P_2) \& (\overline{P_3 \to P_2}) = (\overline{P}_1 \vee P_2) \& (\overline{\overline{P}_3 \vee P_2}) =$$

$$= (\overline{P}_1 \vee P_2) \& P_3 \& \overline{P}_2 = \overline{P}_1 \& P_3 \& \overline{P}_2 \vee P_2 \& P_3 \& \overline{P}_2.$$

Высказывание $P_2 \& \overline{P}_2$ ложно, а следовательно, ложно и высказывание $P_2 \& P_3 \& \overline{P}_2$. Поэтому должно быть истинным высказывание $\overline{P}_1 \& P_3 \& \overline{P}_2$, а это означает, что логику изучал третий учащийся, а первый и второй не изучали. □

Для решения следующей задачи составим логическое выражение, удовлетворяющее всем условиям, затем заполним для него таблицу истинности. Анализ полученной таблицы истинности позволит получить требуемый результат.

Пример 12. Задача «Кто виноват?»

По обвинению в ограблении перед судом предстали Иванов, Петров, Сидоров. Следствием установлено:

1) если Иванов не виновен или Петров виновен, то Сидоров виновен;
2) если Иванов не виновен, то Сидоров не виновен.

Виновен ли Иванов?

Решение. Рассмотрим простые высказывания:

A = {Иванов виновен},

B = {Петров виновен},

C = {Сидоров виновен}.

Запишем на языке алгебры логики факты, установленные следствием:

$(\overline{A} \vee B) \to C$;

$\overline{A} \to \overline{C}$.

Обозначим $F = ((\overline{A} \vee B) \to C) \,\&\, (\overline{A} \to \overline{C})$ — единое логическое выражение для всех требований задачи. Оно истинно. Составим для него таблицу истинности:

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Решить данную задачу — значит указать, при каких значениях A полученное сложное высказывание F истинно. Для этого необходимо проанализировать все строки таблицы истинности, где $F = 1$. И если хотя бы в одном из таких случаев $A = 0$ (Иванов не виновен), то у следствия недостаточно фактов для того, чтобы обвинить Иванова в преступлении.

Анализ таблицы показывает, что высказывание F истинно только в тех случаях, когда A истинно, т. е. Иванов в ограблении виновен. □

Иногда, для того чтобы решить задачу, нет необходимости составлять единое логическое выражение, удовлетворяющее всем условиям задачи, достаточно построить таблицу истинности, отражающую каждое условие задачи, и проанализировать ее.

Пример 13. Решим текстовую задачу, построив совместную таблицу истинности для условий задачи и пранализировав ее.

Три подразделения A, B, C торговой фирмы стремились получить по итогам года максимальную прибыль. Экономисты высказали следующие предположения:

1) A получит максимальную прибыль только тогда, когда получат максимальную прибыль B и C,
2) Либо A и C получат максимальную прибыль одновременно, либо одновременно не получат,
3) Для того чтобы подразделение C получило максимальную прибыль, необходимо, чтобы и B получило максимальную прибыль.

По завершении года оказалось, что одно из трех предположений ложно, а остальные два истинны. Какие из названных подразделений получили максимальную прибыль?

Решение. Рассмотрим простые высказывания:

A = {A получит максимальную прибыль},

B = {B получит максимальную прибыль},

C = {C получит максимальную прибыль}.

Запишем на языке алгебры логики прогнозы, высказанные экономистами:

1) $F_1 = A \to B \ \& \ C$;
2) $F_2 = A \ \& \ C \vee \overline{A} \ \& \ \overline{C}$;
3) $F_3 = C \to B$.

Составим таблицу истинности для F_1, F_2, F_3.

A	B	C	F_1	F_2	F_3
0	0	0	1	1	1
0	0	1	1	0	0
0	1	0	1	1	1
0	1	1	1	0	1
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	0	0	1
1	1	1	1	1	1

Теперь вспомним, что один из прогнозов F_1, F_2, F_3 оказался ложным, а остальные два — истинными. Эта ситуация соответствует четвертой строке таблицы.

Ответ: B и C получат максимальную прибыль. □

Если число простых высказываний в решаемой задаче больше трех, то таблица истинности насчитывает 16, 32 и более строк, заполнять ее вручную достаточно трудоемко. Умение заполнять таблицу истинности с привлечением компьютера помогает преодолеть это неудобство.

Вопросы и задания

1. Задача «Валютные махинации».

   В нарушении правил обмена валюты подозреваются четыре работника банка — Антипов (A), Борисов (B), Цветков (C) и Дмитриев (D). Известно:
   1) если A нарушил правила обмена валюты, то и B нарушил;
   2) если B нарушил, то и C нарушил или A не нарушил;
   3) если D не нарушил, то A нарушил, а C не нарушил;
   4) если D нарушил, то и A нарушил.

   Кто из подозреваемых нарушил правила обмена валюты?
2. Задача «Пятеро друзей».

   Пятеро друзей решили записаться в кружок любителей логических задач: Андрей (A), Николай (N), Виктор (V), Григорий (G), Дмитрий (D). Но староста кружка поставил им ряд условий: «Вы должны приходить к нам так, чтобы:
   1) если A приходит вместе с D, то N должен присутствовать обязательно;
   2) если D отсутствует, то N должен быть, а V пусть не приходит;
   3) A и V не могут одновременно ни присутствовать, ни отсутствовать;
   4) если придет D, то G пусть не приходит;
   5) если N отсутствует, то D должен присутствовать, но это в том случае, если не присутствует V; если же и V присутствует при отсутствии N, то D приходить не должен, а G должен прийти».

   В каком составе друзья смогут прийти на занятия кружка?
3. Решите логическую задачу, используя только алгебраические преобразования логических формул.

   Брауну, Джонсу и Смиту предъявлено обвинение в соучастии в ограблении банка. Похитители скрылись на

поджидавшем их автомобиле. На следствии Браун показал, что преступники скрылись на синем «Бьюике»; Джонс сказал, что это был черный «Крайслер», а Смит утверждал, что это был «Форд Мустанг» и ни в коем случае не синий. Стало известно, что, желая запутать следствие, каждый из них указал правильно либо только марку машины, либо только ее цвет. Какого цвета и какой марки был автомобиль?

4. Решите логическую задачу, используя только алгебраические преобразования логических формул.

   Для полярной экспедиции из восьми претендентов A, B, C, D, E, F, G и H надо отобрать шестерых специалистов: биолога, гидролога, синоптика, радиста, механика и врача. Обязанности биолога могут выполнять E и G, гидролога — B и F, синоптика — F и G, радиста — C и D, механика — C и H, врача — A и D. Хотя некоторые претенденты владеют двумя специальностями, в экспедиции каждый сможет выполнять только одну обязанность. Кого и кем следует взять в экспедицию, если F не может ехать без B, D — без C и без H, C не может ехать одновременно с G, а A вместе с B?

5. Напишите программу, которая строит таблицу истинности по заданной логической формуле. Используйте для этого известный вам язык программирования или воспользуйтесь электронной таблицей.

§ 3.5. Алгебра переключательных схем

Одним из практических применений алгебры логики является область параллельно-последовательных переключательных схем.

Определение 13. *Переключательная схема* — это изображение некоторого устройства, содержащего только двухпозиционные переключатели, которые могут находиться в одном из двух состояний: замкнутое (ток проходит) или разомкнутое (ток не проходит).

Очевидно, что состояние переключателя можно кодировать числами 1 и 0. Большинство переключательных схем можно разбить на участки из последовательно или параллельно соединенных переключателей. Каждому переключателю поставим в соответствие логическую переменную, принимающую значение «истина» тогда, ког-

да переключатель замкнут, и «ложь», если переключатель разомкнут. На схемах переключатели будем обозначать теми же буквами, что и соответствующие им переменные.

При описании переключательных схем будем придерживаться следующих соглашений:

1. Все переключатели, работающие так, что они всегда либо одновременно замкнуты, либо одновременно разомкнуты, обозначаются одной и той же буквой.
2. Переключателям, соединенным параллельно, поставим в соответствие операцию дизъюнкции: ток в этой цепи (рис. 3.1, *а*) будет протекать или при замкнутом переключателе A, или при замкнутом переключателе B, или при замкнутых переключателях A и B одновременно.

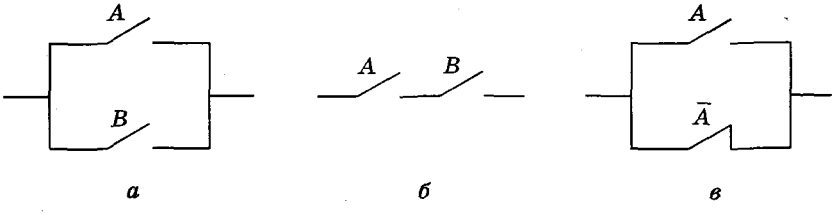


Рис. 3.1

3. Переключателям, соединенным последовательно, поставим в соответствие операцию конъюнкции: ток в цепи (рис. 3.1, *б*) потечет только тогда, когда замкнут переключатель A и замкнут переключатель B.
4. Два переключателя, работающие так, что один из них замкнут, когда другой разомкнут, и наоборот, описываются формулами A и $\bar{A}$ соответственно (рис. 3.1, *в*).

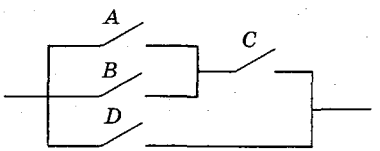
Прочитать переключательную схему — значит определить, протекает по ней ток или нет при указанных состояниях переключателей.

Определение 14. Две схемы, содержащие одни и те же переключатели A, B, ..., будем считать *равными*, если при одном и том же состоянии переключателей обе схемы одновременно пропускают или не пропускают ток. Из двух равных схем более *простой* будем считать ту, которая содержит меньше переключателей.

Каждой переключательной схеме можно поставить в соответствие формулу, истинную тогда и только тогда,

когда схема проводит ток. В алгебре переключательных схем выполняются все законы алгебры логики. В этом достаточно просто убедиться, если построить и прочитать соответствующие этим законам схемы, а затем сравнить столбец состояния каждой схемы с результирующим столбцом таблицы истинности для соответствующей логической формулы.

Пример 14. Составим формулу для схемы, изображенной на рисунке.



Переключатели A и B соединены параллельно, следовательно, этот участок схемы описывается как дизъюнкция переменных: $A \vee B$. Далее следует последовательное соединение с переключателем C: $(A \vee B) \,\&\, C$. Рассмотренный участок цепи параллельно соединяется с переключателем D: $(A \vee B) \,\&\, C \vee D$. □

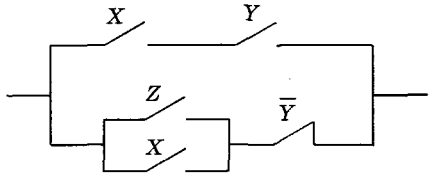
Синтез переключательной схемы — это разработка схемы, условия работы которой заданы таблицей истинности или словесным описанием. Упрощение (минимизация) переключательной схемы сводится к упрощению соответствующей ей формулы на основании законов алгебры логики.

Вопросы и задания

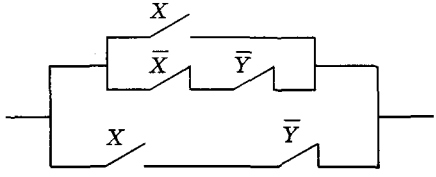
1. Каким образом интерпретируются элементы переключательных схем с помощью объектов и операций алгебры логики? Заполните следующую таблицу:

Переключательная техника	Алгебра логики
Переключатель	
Переключатель замкнут	
Переключатель разомкнут	
Соединение	
Последовательное соединение	
Параллельное соединение	
Состояние тока	
Прохождение тока	
Отключение тока	

2. Комиссия состоит из трех рядовых членов и председателя. Постройте электрическую цепь для тайного голосования, если оно производится следующим образом: каждый член комиссии при голосовании «за» нажимает кнопку. Лампочка зажигается в случаях, если предложение набрало большинство голосов или число голосов «за» и «против» равное, но за предложение «за» подан голос председателя.
3. Можно ли изображенную на рисунке электрическую цепь заменить более простой схемой, соответствующей формуле $X \vee \overline{Y}\ \&\ Z$?



4. Минимизируйте следующую переключательную схему:



§ 3.6. Булевы функции

Как было сказано выше, значение логической формулы определяется заданными значениями входящих в формулу переменных. Тем самым каждая формула может рассматриваться как способ задания *функции алгебры логики*.

Определение 15. *Логической* (булевой) *функцией* называют функцию $f(x_1,x_2,...,x_n)$, аргументы которой x_1, x_2, ..., x_n (независимые переменные) и сама функция (зависимая переменная) принимают значения 0 или 1. Логические функции могут быть заданы табличным способом или аналитически — в виде соответствующих формул.

В общем случае булева функция от n аргументов определяется как отображение $\{0, 1\}^n \to \{0, 1\}$. Обычно совокупность значений n аргументов интерпретиру-

ют как строку нулей и единиц (бинарную строку) длины n. Существует ровно 2^n различных бинарных строк длины n. Так как на каждой такой строке некая функция может принимать значение 0 или 1, то общее количество различных булевых функций от n аргументов равно 2^{2^n}.

Для $n = 2$ существует 16 различных булевых функций.

Рассмотрим их подробно.

x	y	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}	f_{16}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

$f_1(x, y) = 0$ — константа «ложь» ($f(x, y) \equiv 0$);

$f_2(x, y) = x \,\&\, y$ — конъюнкция;

$f_3(x, y) = \overline{x \to y}$ — отрицание импликации;

$f_4(x, y) = x$ — функция равна первому аргументу;

$f_5(x, y) = \overline{y \to x}$ — отрицание обратной импликации;

$f_6(x, y) = y$ — функция равна второму аргументу;

$f_7(x, y) = x \oplus y$ — разделительная (строгая) дизъюнкция (исключающее ИЛИ, сумма по модулю 2);

$f_8(x, y) = x \vee y$ — дизъюнкция;

$f_9(x, y) = x \downarrow y$ — стрелка Пирса[1] (отрицание дизъюнкции, ИЛИ–НЕ);

$f_{10}(x, y) = x \sim y$ — эквивалентность;

$f_{11}(x, y) = \overline{y}$ — отрицание второго аргумента;

$f_{12}(x, y) = y \to x = x \leftarrow y$ — обратная импликация;

$f_{13}(x, y) = \overline{x}$ — отрицание первого аргумента;

$f_{14}(x, y) = x \to y$ — импликация;

$f_{15}(x, y) = x \mid y$ — штрих Шеффера[2] (отрицание конъюнкции, И–НЕ);

$f_{16}(x, y) = 1$ — константа «истина» ($f(x, y) \equiv 1$).

[1] Ч. С. Пирс — американский философ, логик, математик; исследовал свойства функции $x \downarrow y$.

[2] Х. М. Шеффер — американский логик и математик; указал полноту функции $x \mid y$.

Функции, выраженные с использованием одной логической операции, называются по имени этой логической операции. Например, $f_2(x, y) = x \,\&\, y$ — конъюнкция.

С увеличением числа аргументов количество логических функций резко возрастает. Для трех переменных существует 256 различных булевых функций. Для четырех переменных — уже 65 536 функций и т. д. Пугаться резкого увеличения количества функций не следует, как будет показано ниже, при необходимости любая булева функция, может быть выражена только, например, через &, $\vee$ и отрицание.

Вопросы и задания

1. Сколько существует логических функций одной переменной?
2. Составьте сводную таблицу всех логических функций одной переменной. Запишите аналитические выражения этих функций.
3. Убедитесь, что следующие шесть функций тождественны, т. е. принимают одинаковые значения на одинаковых наборах переменных:

$f_1(x_1, x_2, x_3) = x_3 \,\&\, x_1 \vee x_2;$
$f_2(x_1, x_2, x_3) = x_2 \,\&\, 1 \vee x_1 \,\&\, x_3;$
$f_3(x_1, x_2, x_3) = x_2 \vee x_1 \,\&\, x_3 \vee 0;$
$f_4(x_1, x_2, x_3) = x_2 \vee x_1 \,\&\, x_3 \,\&\, (x_1 \vee \overline{x}_1);$
$f_5(x_1, x_2, x_3, x_4) = x_2 \vee x_1 \,\&\, x_3 \vee x_4 \,\&\, \overline{x}_4;$
$f_6(x_1, x_2, x_3) = x_2 \vee x_1 \,\&\, x_3.$

§ 3.7. Канонические формы логических формул. Теорема о СДНФ

Всякая логическая формула определяет некоторую булеву функцию. В то же время ясно, что для всякой булевой функции можно записать бесконечно много формул, ее представляющих (см. задание 3 к § 3.6). Действительно, если имеется хотя бы одна формула, выражающая булеву функцию, то, используя тождественные преобразования, можно изменить эту формулу, построив сколь угодно сложную равносильную формулу.

Одна из основных задач алгебры логики — нахождение *канонических форм* (т. е. формул, построенных по определенному правилу, канону), а также наиболее простых формул, представляющих булевы функции.

Определение 16. Если логическая функция выражена через дизъюнкцию, конъюнкцию и отрицание переменных, то такая форма представления называется *нормальной*.

Среди нормальных форм выделяют такие, в которых функции записываются единственным образом. Их называют *совершенными*.

Особую роль в алгебре логики играют *классы дизъюнктивных и конъюнктивных совершенных нормальных форм*. В их основе лежат понятия элементарной дизъюнкции и элементарной конъюнкции.

Определение 17. Формулу называют *элементарной конъюнкцией*, если она является конъюнкцией одной или нескольких переменных, взятых с отрицанием или без отрицания. Одну переменную или ее отрицание считают *одночленной элементарной конъюнкцией*.

Пример 15. Формулы x_2, $\overline{x}_2$, $x_1 \,\&\, x_3$, $x_1 \,\&\, x_3 \,\&\, x_1 \,\&\, \overline{x}_3$ являются элементарными конъюнкциями; первые две из них — одночленными. □

Определение 18. Формула называется *дизъюнктивной нормальной формой* (ДНФ), если она является дизъюнкцией неповторяющихся элементарных конъюнкций. ДНФ записываются в виде $A_1 \vee A_2 \vee \ldots \vee A_n$, где каждое A_i — элементарная конъюнкция.

Пример 16. Приведем две дизъюнктивные нормальные формы: $x_2 \vee x_1 \,\&\, x_3$, $\overline{x}_2 \vee x_2 \,\&\, x_1 \vee \overline{x}_1$. □

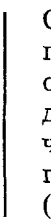
Определение 19. Формула A от k переменных называется *совершенной дизъюнктивной нормальной формой* (СДНФ), если:

1) A является ДНФ, в которой каждая элементарная конъюнкция есть конъюнкция k переменных $x_1, x_2, \ldots, x_k$, причем на i-м месте этой конъюнкции стоит либо переменная x_i, либо ее отрицание;

2) все элементарные конъюнкции в такой ДНФ попарно различны.

Задание. *Даны формулы* $A = x_1 \,\&\, \bar{x}_2 \vee x_1 \,\&\, x_2$; $B = x_1 \vee x_2 \,\&\, x_3$ и $C = x_1 \,\&\, x_2 \vee x_2 \,\&\, \bar{x}_2$. *Определить, являются ли они СДНФ двух переменных.*

Решение. Формула A является СДНФ двух переменных. Формулы B и C не являются СДНФ. Формула B зависит от трех переменных, но количество переменных в элементарных конъюнкциях меньше трех. В формуле C переменная x_2 дважды входит в одну и ту же элементарную конъюнкцию. □



Совершенная дизъюнктивная нормальная форма представляет собой формулу, построенную по строго определенным правилам с точностью до порядка следования элементарных конъюнкций (дизъюнктивных членов) в ней. Она является примером однозначного представления булевой функции в виде формульной (алгебраической) записи.

Определение 20. Формула называется *элементарной дизъюнкцией*, если она является дизъюнкцией (быть может, одночленной) переменных и отрицаний переменных.

Пример 17. Приведем три элементарные дизъюнкции: x_2, $\bar{x}_2$, $x_1 \vee x_3$. □

Определение 21. Формула называется *конъюнктивной нормальной формой* (КНФ), если она является конъюнкцией неповторяющихся элементарных дизъюнкций. КНФ записываются в виде $A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_n$, где каждое A_i — элементарная дизъюнкция.

Пример 18. Формулы

x_2, $\bar{x}_2$, $x_2 \vee x_2$, $(\bar{x}_2 \vee x_1) \,\&\, x_3$, $(x_2 \vee \bar{x}_2) \,\&\, (x_1 \vee x_1)$

являются конъюнктивными нормальными формами. □

Определение 22. Формула A от k переменных называется *совершенной конъюнктивной нормальной формой* (СКНФ), если:

1) A является КНФ, в которой каждая элементарная дизъюнкция есть дизъюнкция k переменных $x_1, x_2, \ldots, x_k$, причем на i-м месте этой дизъюнкции стоит либо переменная x_i, либо ее отрицание;

2) все элементарные дизъюнкции в такой КНФ попарно различны.

Задание. *Даны формулы* $A = (x_1 \vee \overline{x}_2) \,\&\, (x_1 \vee x_2)$ *и* $B = (x_1 \vee \overline{x}_1) \,\&\, (x_2 \vee x_3)$. *Определить, являются ли они СКНФ.*

Решение. Формула A является СКНФ, а формула B не является СКНФ, поскольку переменная x_1 дважды входит в первый конъюнктивный член, кроме того, количество переменных в каждой элементарной дизъюнкции меньше трех, в то время как формула зависит от трех переменных. □

Вопрос. *Всякую ли логическую функцию можно представить в одной из рассмотренных канонических совершенных форм?*

Ответ. Да, любую булеву функцию, не равную тождественно 0 или 1, можно представить в виде СДНФ или СКНФ. Для обоснования этого утверждения ниже будут доказаны соответствующие теоремы. □

Теорема 1. Пусть $f(x_1, x_2, \ldots, x_n)$ — булева функция от n переменных, не равная тождественно нулю. Тогда существует совершенная дизъюнктивная нормальная форма, выражающая функцию f.

Доказательство. Сначала докажем, что для всякой булевой функции f от n переменных выполняется соотношение:

$$f(x_1, x_2, \ldots, x_i, \ldots, x_n) = \overline{x}_i \,\&\, f(x_1, x_2, \ldots, 0, \ldots, x_n) \vee \vee\, x_i \,\&\, f(x_1, x_2, \ldots, 1, \ldots, x_n), \quad (3.1)$$

где x_i — любая из n переменных.

Формулу (3.1) легко получить, последовательно подставляя вместо переменной x_i все ее возможные значения (ноль и единицу):

$$f(x_1, x_2, \ldots, 0, \ldots, x_n) = 1 \,\&\, f(x_1, x_2, \ldots, 0, \ldots, x_n) \vee \vee\, 0 \,\&\, f(x_1, x_2, \ldots, 1, \ldots, x_n) \equiv f(x_1, x_2, \ldots, 0, \ldots, x_n);$$

$$f(x_1, x_2, \ldots, 1, \ldots, x_n) = 0 \,\&\, f(x_1, x_2, \ldots, 0, \ldots, x_n) \vee \vee\, 1 \,\&\, f(x_1, x_2, \ldots, 1, \ldots, x_n) \equiv f(x_1, x_2, \ldots, 1, \ldots, x_n).$$

Соотношение (3.1) позволяет «выносить» переменную x_i за знак функции. Последовательно вынося $x_1, x_2, \ldots, x_n$ за знак функции f, мы получим следующую формулу:

$$
\begin{aligned}
f(x_1, x_2, \ldots, x_n) = {} & \bar{x}_1 \,\&\, \bar{x}_2 \,\&\, \ldots \,\&\, \bar{x}_{n-1} \,\&\, \bar{x}_n \,\&\, f(0, 0, \ldots, 0, 0) \vee \\
& \vee \bar{x}_1 \,\&\, \bar{x}_2 \,\&\, \ldots \,\&\, \bar{x}_{n-1} \,\&\, x_n \,\&\, f(0, 0, \ldots, 0, 1) \vee \\
& \ldots \qquad (3.2) \\
& \vee x_1 \,\&\, x_2 \,\&\, \ldots \,\&\, x_{n-1} \,\&\, \bar{x}_n \,\&\, f(1, 1, \ldots, 1, 0) \vee \\
& \vee x_1 \,\&\, x_2 \,\&\, \ldots \,\&\, x_{n-1} \,\&\, x_n \,\&\, f(1, 1, \ldots, 1, 1).
\end{aligned}
$$

Так как применение преобразования (3.1) к каждой из переменных увеличивает количество дизъюнктивных членов в два раза, то для функции n переменных в формуле (3.2) мы имеем 2^n дизъюнктивных членов. Причем каждый из них соответствует значению функции на одном из 2^n возможных наборов значений n переменных. Если на некотором наборе $f = 0$, то весь соответствующий дизъюнктивный член в (3.2) также равен 0, и в представлении данной функции он фактически не нужен. Если же $f = 1$, то в соответствующем дизъюнктивном члене само значение функции можно опустить. В результате для произвольной булевой функции f мы получили формулу, состоящую из n-членных попарно различных элементарных конъюнкций, объединенных дизъюнкциями, т. е. искомая СДНФ построена.
Теорема доказана.

На основании теоремы 1 можно предложить следующий *алгоритм построения СДНФ по таблице истинности* функции f.

Алгоритм построения СДНФ по таблице истинности

1. В таблице истинности отмечаем наборы переменных, на которых значение функции f равно единице.
2. Записываем для каждого отмеченного набора конъюнкцию всех переменных следующим образом: если значение некоторой переменной в этом наборе равно 1, то в конъюнкцию включаем саму переменную, в противном случае — ее отрицание.
3. Все полученные конъюнкции связываем операциями дизъюнкции. △

Следствие. Для любой формулы можно найти равносильную ей ДНФ.

Доказательство. Если булева функция не равна тождественно нулю, то, согласно доказанной теореме 1, можно построить СДНФ, ее реализующую. Построенная СДНФ является одной из искомых ДНФ. Если же данная формула равна тождественно нулю, то в качестве искомой ДНФ можно взять, например, $\overline{x}_1 \,\&\, x_1$. □

Аналогично теореме 1 доказывается следующая теорема.

Теорема 2. Пусть $f(x_1, x_2, ..., x_n)$ — булева функция n переменных, не равная тождественно единице. Тогда существует совершенная конъюнктивная нормальная форма, выражающая функцию f.

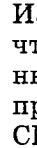
На основании теоремы 2 можно предложить следующий *алгоритм построения СКНФ по таблице истинности* функции f.

Алгоритм построения СКНФ по таблице истинности

1. В таблице истинности отмечаем наборы переменных, на которых значение функции f равно нулю.
2. Записываем для каждого отмеченного набора дизъюнкцию всех переменных следующим образом: если значение некоторой переменной в этом наборе равно 0, то в конъюнкцию включаем саму переменную, в противном случае — ее отрицание.
3. Все полученные дизъюнкции связываем операциями конъюнкции. △

Следствие. Для любой формулы можно найти равносильную ей КНФ.

Доказательство. Если булева функция не равна тождественно единице, то, согласно доказанной теореме 2, можно построить СКНФ, ее реализующую. Построенная СКНФ является одной из искомых КНФ. Если же данная формула равна тождественно единице, то в качестве искомой КНФ можно взять, например, $\overline{x}_1 \vee x_1$. □



Из алгоритмов построения СДНФ и СКНФ следует, что если на большей части наборов значений переменных функция равна 0, то для получения ее формулы проще построить СДНФ, в противном случае — СКНФ.

Пример 19. Требуется построить формулу для функции $f(x_1, x_2, x_3)$, заданной таблицей истинности:

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Используя описанный выше алгоритм, построим для нее СДНФ:

$$f(x_1, x_2, x_3) = \overline{x}_1 \,\&\, x_2 \,\&\, \overline{x}_3 \vee x_1 \,\&\, \overline{x}_2 \,\&\, \overline{x}_3 \vee x_1 \,\&\, \overline{x}_2 \,\&\, x_3.$$

□

Пример 20. Выразим функцию импликация с помощью операций отрицания, дизъюнкции и конъюнкции. Для этого запишем таблицу истинности функции импликация:

x_1	x_2	$f(x_1, x_2) = x_1 \to x_2$
0	0	1
0	1	1
1	0	0
1	1	1

Так как в таблице истинности только один набор переменных, на котором функция принимает значение 0, то проще построить СКНФ.

Ответ: $f(x_1, x_2) = x_1 \to x_2 = \overline{x}_1 \vee x_2$. □

Вопросы и задания

1. Приведите примеры нескольких формул, представляющих собой СДНФ.
2. Приведите примеры нескольких формул, представляющих собой СКНФ.
3. По заданной таблице истинности найдите аналитическое представление логических функций f_1 и f_2:

x_1	x_2	x_3	f_1	f_2
ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ
ЛОЖЬ	ЛОЖЬ	ИСТИНА	ЛОЖЬ	ИСТИНА
ЛОЖЬ	ИСТИНА	ЛОЖЬ	ИСТИНА	ЛОЖЬ
ЛОЖЬ	ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА
ИСТИНА	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ	ИСТИНА
ИСТИНА	ЛОЖЬ	ИСТИНА	ИСТИНА	ИСТИНА
ИСТИНА	ИСТИНА	ЛОЖЬ	ЛОЖЬ	ЛОЖЬ
ИСТИНА	ИСТИНА	ИСТИНА	ИСТИНА	ЛОЖЬ

Проверку произведите с помощью электронной таблицы.

4. С помощью отрицания, дизъюнкции и конъюнкции постройте наиболее простое аналитическое представление для функций эквивалентность и разделительная дизъюнкция.
5. Не используя таблицы истинности, постройте СДНФ и СКНФ, выражающие следующие функции:
   1) $f(x_1, x_2, x_3)$, равную 1 тогда и только тогда, когда большинство переменных равно 1;
   2) $f(x_1, x_2, x_3, x_4)$, равную 1 тогда и только тогда, когда $x_1 + x_2 + x_3 + x_4 \geq 3$. Здесь имеется в виду обычная алгебраическая сумма.
6. Не используя таблицу истинности, преобразуйте в СДНФ следующую функцию:
   $$f(x_1, x_2, x_3) = x_1 \,\&\, x_2 \vee \overline{x}_1 \,\&\, x_3.$$
7. Самостоятельно проведите доказательство теоремы 2.
8. Какие из следующих формул представляют собой СДНФ, а какие СКНФ?
   а) $f(x) = 1$;
   б) $f(x) = x \,\&\, \overline{x}$;
   в) $f(x, y) = \overline{x} \,\&\, \overline{y}$;
   г) $f(x, y) = x \,\&\, \overline{y} \vee \overline{y}$;
   д) $f(x, y, z) = \overline{x} \,\&\, y \,\&\, \overline{z}$;
   е) $f(x, y, z) = x \,\&\, y \vee z$;
   ж) $f(x, y, z) = (x \,\&\, y \,\&\, z) \vee (x \,\&\, \overline{y} \,\&\, \overline{z})$.

§ 3.8. Минимизация булевых функций в классе дизъюнктивных нормальных форм

Результаты, полученные в предыдущем параграфе, имеют большое прикладное значение при проектировании вычислительной техники. Как известно, информация

хранится и обрабатывается компьютерами в двоичном виде, т. е. в виде последовательностей нулей и единиц. Для любой операции (например, сложения чисел) исходными данными и результатами являются последовательности нулей и единиц. Таким образом, фактически происходит вычисление значений булевых функций (возможно, очень сложных, с большим количеством переменных). Теоремы 1 и 2 говорят о том, что любую булеву функцию можно записать в виде формулы с использованием операций $\neg$, & и $\vee$ (в виде СДНФ или СКНФ), причем сделать это можно по единому универсальному алгоритму. Следовательно, если у нас имеются электронные логические элементы (вентили)[1], реализующие операции **НЕ, И, ИЛИ**, то, взяв достаточно много таких элементов и соединив их определенным образом, мы можем построить машину, выполняющую заданный набор действий над двоичными данными.

Конечно, желательно все булевы функции реализовать так, чтобы при этом использовалось как можно меньше логических элементов. Однако разумно уметь находить наиболее короткие формы записи функций в некотором едином классе формул. Большое значение в этом смысле имеют *минимальные ДНФ*.

Определение 23. Дизъюнктивная нормальная форма называется *минимальной*, если она содержит наименьшее общее число вхождений переменных по сравнению со всеми равносильными ей ДНФ. Процесс нахождения минимальной ДНФ называется *минимизацией* в классе ДНФ.

Строить минимальные ДНФ из СДНФ можно разными способами, например используя тождественные преобразования.

Пример 21. Используя закон дистрибутивности и закон поглощения, выполним минимизацию ДНФ $x_1 \,\&\, x_2 \vee x_1 \,\&\, \overline{x}_2$:

$$x_1 \,\&\, x_2 \vee x_1 \,\&\, \overline{x}_2 = x_1 \,\&\, (x_1 \vee \overline{x}_2) = x_1. \qquad \square$$

К сожалению, для нахождения минимальной ДНФ необходимо перебрать все возможные способы применения основных законов алгебры логики к исходной формуле. Для функций от большого числа переменных этот процесс оказывается слишком трудоемким, даже если проводить его с использованием компьютера.

[1] См. § 3.10 «Элементы схемотехники. Логические схемы».

Более эффективным способом нахождения минимальных ДНФ является *метод минимизирующих карт.*

Для булевой функции n переменных составим следующую таблицу (карту):

x_1	x_2	...	x_{n-1}	x_n	x_1x_2	x_1x_3	...	$x_{n-2}x_n$	$x_{n-1}x_n$	...	$x_1x_2...x_{n-1}x_n$
x_1	x_2	...	x_{n-1}	$\bar{x}_n$	x_1x_2	x_1x_3	...	$x_{n-2}\bar{x}_n$	$x_{n-1}\bar{x}_n$	...	$x_1x_2...x_{n-1}\bar{x}_n$
x_1	x_2	...	$\bar{x}_{n-1}$	x_n	x_1x_2	x_1x_3	...	$x_{n-2}x_n$	$\bar{x}_{n-1}x_n$	...	$x_1x_2...\bar{x}_{n-1}x_n$
...	...	...	...	...	...	...	...	...	...	...	...
$\bar{x}_1$	$\bar{x}_2$	...	$\bar{x}_{n-1}$	$\bar{x}_n$	$\bar{x}_1\bar{x}_2$	$\bar{x}_1\bar{x}_3$	...	$\bar{x}_{n-2}\bar{x}_n$	$\bar{x}_{n-1}\bar{x}_n$	...	$\bar{x}_1\bar{x}_2...\bar{x}_{n-1}\bar{x}_n$

В последнем столбце карты перечислены все элементарные конъюнкции, которые могут входить в СДНФ функции n переменных (знаки конъюнкции для краткости опущены). Каждая такая элементарная конъюнкция соответствует одному из возможных наборов n переменных в таблице истинности. В остальных столбцах каждой строки перечислены все возможные конъюнкции меньшего размера, полученные из элементарной конъюнкции последнего столбца путем удаления от одной до $(n-1)$ переменных.

Предположим, что конъюнкция из последнего столбца k-й строки таблицы не входит в СДНФ, выражающую функцию $f(x_1, x_2, ..., x_n)$, тогда любая конъюнкция k-й строки не входит ни в какую ДНФ, выражающую функцию f. Докажем это. Действительно, если конъюнкция не входит в СДНФ, выражающую функцию f, то, согласно алгоритму построения СДНФ, значение функции на этом наборе равно 0. Если какая-то конъюнкция k-й строки вошла в некоторую ДНФ функции f, то на этом наборе функция должна быть равна единице. Мы получили противоречие с первоначальным предположением. Используя доказанное утверждение, можно предложить следующий способ построения минимальной ДНФ по *минимизирующей карте.*

Способ минимизации ДНФ методом минимизирующих карт

1. Вычеркнем из таблицы (минимизирующей карты) все строки, в которых конъюнкция последнего столбца не входит в СДНФ функции f.

2. Конъюнкции «вычеркнутых строк» вычеркнем во всех остальных строках таблицы.
3. Если в строке остались конъюнкции с различным числом сомножителей, то конъюнкции с не минимальным числом сомножителей оставляем только тогда, когда они встречаются в других строках.
4. Отметим конъюнкции, оставшиеся единственными на строке. Вычеркнем строки, в которых присутствуют такие же конъюнкции.
5. Всеми возможными способами выберем из каждой строки по одной конъюнкции (из оставшихся) и составим для каждого случая ДНФ.
6. Из всех построенных ДНФ выберем минимальную. Заметим, что мы должны выполнить перебор различных ДНФ для нахождения минимальной из них. Однако в данном случае число вариантов перебора, как правило, существенно меньше вариантов перебора равносильных ДНФ или способов сокращения СДНФ. △

Покажем, что, действуя в соответствии с п. 1–6, мы получим ДНФ, выражающую функцию $f(x_1, x_2, \ldots, x_n)$. Действительно, пусть F — ДНФ, полученная в п. 6. Тогда если f на каком-то наборе равна 1, то и $F = 1$. В самом деле, если этому набору соответствует j-я строка, то хотя бы одна конъюнкция из этой строки осталась невычеркнутой (в этой или в других строках) и вошла в ДНФ. Любая конъюнкция из этой строки на этом наборе значений принимает значение 1. Следовательно, и формула F, содержащая одну из таких конъюнкций, принимает на этом наборе значение 1. Если же на каком-то наборе $f = 0$, то на этом наборе все невычеркнутые конъюнкции принимают значение 0, так как все конъюнкции, принимающие на этом наборе значение 1, уже вычеркнуты при выполнении п. 2. Следовательно, ДНФ, составленная из оставшихся невычеркнутых конъюнкций, принимает значение 0, т. е. $F = 0$.

Пример 22. Пусть требуется минимизировать функцию

$$f(x, y, z) = x_1x_2\overline{x}_3 \vee x_1\overline{x}_2x_3 \vee x_1\overline{x}_2\overline{x}_3 \vee \overline{x}_1\overline{x}_2x_3.$$

Составим минимизирующую карту для функции трех переменных. Отметим знаком «*» не вошедшие в СДНФ строки.

x_1	x_2	x_3	x_1x_2	x_1x_3	x_2x_3	$x_1x_2x_3$	*
x_1	x_2	$\overline{x}_3$	x_1x_2	$x_1\overline{x}_3$	$x_2\overline{x}_3$	$x_1x_2\overline{x}_3$	
x_1	$\overline{x}_2$	x_3	$x_1\overline{x}_2$	x_1x_3	$\overline{x}_2x_3$	$x_1\overline{x}_2x_3$	
x_1	$\overline{x}_2$	$\overline{x}_3$	$x_1\overline{x}_2$	$x_1\overline{x}_3$	$\overline{x}_2\overline{x}_3$	$x_1\overline{x}_2\overline{x}_3$	
$\overline{x}_1$	x_2	x_3	$\overline{x}_1x_2$	$\overline{x}_1x_3$	x_2x_3	$\overline{x}_1x_2x_3$	*
$\overline{x}_1$	x_2	$\overline{x}_3$	$\overline{x}_1x_2$	$\overline{x}_1\overline{x}_3$	$x_2\overline{x}_3$	$\overline{x}_1x_2\overline{x}_3$	*
$\overline{x}_1$	$\overline{x}_2$	x_3	$\overline{x}_1\overline{x}_2$	$\overline{x}_1x_3$	$\overline{x}_2x_3$	$\overline{x}_1\overline{x}_2x_3$	
$\overline{x}_1$	$\overline{x}_2$	$\overline{x}_3$	$\overline{x}_1\overline{x}_2$	$\overline{x}_1\overline{x}_3$	$\overline{x}_2\overline{x}_3$	$\overline{x}_1\overline{x}_2\overline{x}_3$	*

После действий в соответствии с п. 1–3 в таблице останутся конъюнкции, находящиеся в выделенных ячейках таблицы. Выполнив п. 4–6, получим минимальную ДНФ: $x_1\overline{x}_3 \vee \overline{x}_2x_3$.

Практические задания

1. Постройте отрицание к сложному высказыванию на русском языке. (В § 1.3 было показано, как непросто построить отрицание к простым высказываниям.) Возможный способ выполнения задания:
   1) выделите простые высказывания и обозначьте их буквами;
   2) запишите исходное предложение на языке алгебры логики, т. е. в виде логической формулы;
   3) постройте таблицу истинности полученного логического выражения;
   4) запишите таблицу значений противоположного по смыслу выражения;
   5) восстановите по ней искомое логическое выражение (например, в виде СДНФ или СКНФ);
   6) если вы получили искомую формулу в виде СДНФ, то минимизируйте ее;
   7) запишите на русском языке высказывание, соответствующее полученному выражению.

   Варианты задания

   а) Если центральные углы равны, то и соответствующие им дуги равны, а если соответствующие центральным углам дуги равны, то и центральные углы равны.

   б) Если две плоскости взаимно перпендикулярны и к одной из них проведен перпендикуляр, имеющий общую точку с другой плоскостью, то этот перпендикуляр весь лежит в этой плоскости.

в) Если стороны одного угла соответственно перпендикулярны сторонам другого угла, то такие углы или равны, или в сумме составляют два прямых.

г) Если пирамида пересечена плоскостью, параллельной основанию, то в сечении получается многоугольник, подобный основанию.

д) Для того чтобы оплатить проезд в общественном транспорте, необходимо иметь некоторую сумму денег и достаточно иметь 100 руб.

е) Из того, что некоторая ломаная, вписанная в одну окружность и описанная около другой окружности, замкнулась, следует, что и любая такая ломаная замкнется, а из того, что некоторая подобная ломаная не замкнулась, следует, что и любая такая ломаная не замкнется.

2. По выданной вам таблице истинности булевой функции постройте СДНФ и найдите для нее минимальную ДНФ любым удобным вам способом.

§ 3.9. Полные системы булевых функций

Теоретические результаты, изложенные в этом параграфе, имеют важное значение при разработке элементной базы вычислительных машин.

Определение 24. Система булевых функций $\{f_1, f_2, ..., f_n\}$ называется *полной*, если произвольная булева функция может быть выражена через функции $f_1, f_2, ..., f_n$.

Пример 23. Полной является система функций $\{\neg, \&, \vee\}$. Действительно, согласно теоремам 1 и 2 (см. § 3.7), любая булева функция представима в виде СДНФ либо СКНФ, в выражении каждой из которых используются только упомянутые в примере функции. □

Полноту других систем можно доказать с помощью следующего утверждения.

Утверждение. Пусть система $\{f_1, f_2, ..., f_n\}$ — полна и любая из функций $f_1, f_2, ..., f_n$ может быть выражена через функции $g_1, ..., g_m$. Тогда система $\{g_1, ..., g_m\}$ также полна.

Задание. *Докажите, что система функций* $\{\neg, \vee\}$ *является полной.*

Решение. Для доказательства воспользуемся сформулированным выше утверждением. Действительно, пусть

$f_1(x_1) = \overline{x}_1$, $f_2(x_1, x_2) = x_1 \vee x_2$, $f_3(x_1, x_2) = x_1 \,\&\, x_2$,
$g_1(x_1) = \overline{x}_1$, $g_2(x_1, x_2) = x_1 \vee x_2$.

Выразим функции f_1, f_2, f_3 через g_1, g_2:

$f_1(x_1) = g_1(x_1)$, $f_2(x_1, x_2) = g_2(x_1, x_2)$,

$f_3(x_1, x_2) = x_1 \,\&\, x_2 = \overline{\overline{x}_1 \vee \overline{x}_2} = g_1(g_2(g_1(x_1), g_1(x_2)))$.

Для выражения конъюнкции через дизъюнкцию и отрицание был использован один из законов де Моргана. Используя второй из этих законов, можно доказать полноту системы $\{\neg, \&\}$. □

Таким образом, любая (сколь угодно сложная) булева функция может быть выражена через две функции $\{\neg, \&\}$ или $\{\neg, \vee\}$.

Еще более неожиданным может показаться тот факт, что любую булеву функцию можно выразить всего лишь через одну функцию. Другими словами, существует функционально полная система, состоящая из одной булевой функции.

Определение 25. Логическая функция *штрих Шеффера* (другое название **И-НЕ**) обозначается $x_1 \mid x_2$ и задается следующей таблицей истинности:

x_1	x_2	$x_1 \mid x_2$
0	0	1
0	1	1
1	0	1
1	1	0

Пример 24. Докажем, что функция штрих Шеффера является полной. Построим для нее СКНФ, т. е. выразим ее через конъюнкцию, дизъюнкцию и отрицание.

$x_1 \mid x_2 = \overline{x}_1 \vee \overline{x}_2$ или $x_1 \mid x_2 = \overline{x_1 \,\&\, x_2}$ (последняя формула объясняет другое название данной функции — И-НЕ).

Тогда $\overline{x}_1 = \overline{x_1 \,\&\, x_1} = x_1 \mid x_1$;

$x_1 \vee x_2 = \overline{\overline{x_1 \vee x_2}} = \overline{\overline{x}_1 \,\&\, \overline{x}_2} = \overline{x}_1 \mid \overline{x}_2 = (x_1 \mid x_1) \mid (x_2 \mid x_2)$;

$x_1 \,\&\, x_2 = \overline{\overline{x_1 \,\&\, x_2}} = \overline{x_1 \mid x_2} = (x_1 \mid x_2) \mid (x_1 \mid x_2)$.

Таким образом, мы выразили через функцию штрих Шеффера функции $\{\neg, \&, \vee\}$. Следовательно, система, состоящая только из функции штрих Шеффера, полна. □

Определение 26. Логическая функция *стрелка Пирса* (другое название **ИЛИ-НЕ**) обозначается $x_1 \downarrow x_2$ и задается следующей таблицей истинности:

x_1	x_2	$x_1 \downarrow x_2$
0	0	1
0	1	0
1	0	0
1	1	0

Можно доказать, что и система, состоящая из одной функции стрелка Пирса, является полной.

Из приведенных выше рассуждений следует, что если научиться физически представлять логический элемент (вентиль), реализующий функцию, являющуюся полной, то любая другая функция может быть реализована в виде схемы, состоящей из одинаковых вентилей, соединенных для каждой функции особым образом.

Вопросы и задания

1. Выразите функции $\&$, $\vee$, $\neg$ через *стрелку Пирса*.
2. Докажите, что система функций, состоящая только из функции отрицания, не является полной.
3. Докажите, что система функций, состоящая только из функции конъюнкция, не является полной.
4. Докажите, что система функций $\{\&, \vee\}$ не является полной.
5. Используя второй закон де Моргана, докажите, что система $\{\neg, \&\}$ полна.
6. Выразите функции $\&$ и $\vee$ через $\rightarrow$ и $\neg$, доказав тем самым полноту системы $\{\rightarrow, \neg\}$.
7. Как с помощью функции исключающее ИЛИ и одной из констант 0 или 1 (определите, какой именно) можно выразить логическое отрицание?

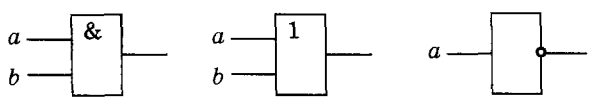
§ 3.10. Элементы схемотехники. Логические схемы

Любое устройство компьютера, выполняющее арифметические или логические операции, может рассматриваться как преобразователь двоичной информации: значения входных переменных для него — последовательность нулей и единиц, а значения выходной функции — новая двоичная последовательность. Необходимые преобразования информации в блоках компьютера производятся логическими устройствами двух типов: комбинационными схемами и цифровыми автоматами с памятью.

В *комбинационной схеме* набор выходных сигналов в любой момент времени полностью определяется набором входных сигналов.

Определение 27. Дискретный преобразователь, который выдает после обработки двоичных сигналов значение одной из логических операций, называется *логическим элементом* (*вентилем*).

Ниже приведены условные обозначения (схемы) логических элементов, реализующих логическое умножение, логическое сложение и отрицание.



а *б* *в*

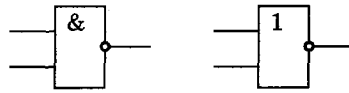
Логический элемент **И** (конъюнктор) реализует операцию логического умножения (рис. 3.2, *а*). Единица на выходе этого элемента появится тогда и только тогда, когда на всех входах будут единицы.

Логический элемент **ИЛИ** (дизъюнктор[1]) реализует операцию логического сложения (рис. 3.2, *б*). Если хотя бы на одном входе будет единица, то на выходе элемента также будет единица, иначе на выходе будет ноль.

Логический элемент **НЕ** (инвертор) реализует операцию отрицания (рис. 3.2, *в*). Если на входе элемента ноль, то на выходе единица и наоборот.

[1] Знак «1» на схеме элемента — дань устаревшему обозначению операции ИЛИ «≥1» — результат операции ИЛИ равен 1, если сумма значений операндов больше или равна 1.

Базовыми в микроэлектронике являются также логические элементы **И-НЕ** и **ИЛИ-НЕ**, реализующие функции штрих Шеффера и стрелка Пирса. Их условные обозначения:



Из отдельных логических элементов можно составить, например, устройства, производящие арифметические операции над двоичными числами.

Определение 28. Электронная логическая схема, выполняющая суммирование двоичных кодов, называется *сумматором*.

Рассмотрим схему сложения двух n-разрядных двоичных чисел.

$$\begin{array}{rc|l} & a_n\ldots & a_i & \ldots a_1a_0 \\ + & b_n\ldots & b_i & \ldots b_1b_0 \\ \hline s_{n+1}s_n\ldots & & s_i & \ldots s_1s_0 \end{array}$$

При сложении цифр i-го разряда складываются a_i и b_i, к ним прибавляется p_i — признак переноса из $(i-1)$-го разряда. Результатом сложения будет s_i и p_{i+1} — признак переноса в следующий разряд.

Таким образом, одноразрядный двоичный сумматор — это устройство с тремя входами и двумя выходами. Его работа описывается следующей таблицей истинности:

Входы			Выходы	
a_i	b_i	p_i	s_i	p_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

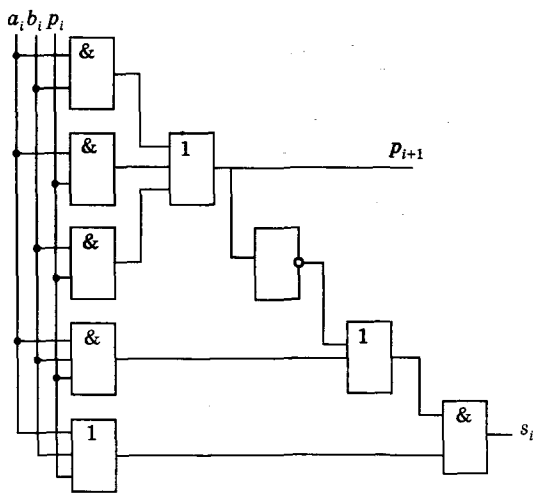
Выходные функции можно восстановить по таблице в виде СДНФ или СКНФ и упростить с помощью тождественных преобразований. В результате преобразований искомые функции приобретают, например, следующий вид:

$$p_{i+1} = a_i \,\&\, b_i \vee a_i \,\&\, p_i \vee b_i \,\&\, p_i;$$
$$s_i = (\overline{p}_{i+1} \vee a_i \,\&\, b_i \,\&\, p_i) \,\&\, (a_i \vee b_i \vee p_i). \qquad (3.3)$$

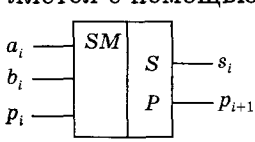
Первое из соотношений (3.3) является решением задания 5, п. 1 к § 3.7. Второе из соотношений (3.3) выведено из СДНФ с учетом уже имеющегося выражения для p_{i+1}.

Заметим, что функции p_{i+1} и s_i можно выразить другими формулами, что, естественно, приведет к другим логическим схемам. Так, наиболее короткой формулой для s_i является следующая: $s_i = a_i \oplus b_i \oplus p_i$.

Одноразрядный двоичный сумматор можно реализовать следующей схемой, что соответствует (3.3):



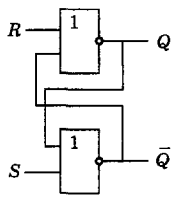
Сложение n-разрядных двоичных чисел осуществляется с помощью комбинации одноразрядных сумматоров (условное обозначение одноразрядного сумматора приведено на рисунке слева.) В зависимости от способа ввода/вывода данных и организации переносов многоразрядные сумматоры бывают последовательного и параллельного принципа действия.



В *цифровых автоматах с памятью* набор выходных сигналов зависит не только от набора входных сигналов, но и от внутреннего состояния данного устройства. Такие устройства всегда имеют память.

Определение 29. Логический элемент, способный хранить один разряд двоичного числа, называют *триггером*.

Триггер был изобретен в 1918 г. М. А. Бонч-Бруевичем (1888–1940). Самый простой триггер — *RS*. Он состоит из двух элементов **ИЛИ-НЕ**, входы и выходы которых соединены кольцом: выход первого соединен со входом второго, выход второго — со входом первого. Схема *RS*-триггера:



Здесь: вход S (set) — установка триггера в 1, вход R (reset) — установка триггера в 0.

Принцип работы *RS*-триггера иллюстрирует следующая таблица истинности:

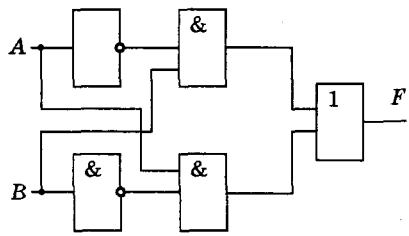
Режимы работы триггера	Входы		Состояние триггера
	R	S	Q
Хранение предыдущего состояния	0	0	Q
Установка триггера в 1	0	1	1
Установка триггера в 0	1	0	0
Запрещенное состояние	1	1	Недопустимо

Обычно на входы поступают сигналы $R = 0$ и $S = 0$, и триггер хранит старое состояние. Если на вход S поступает на короткое время сигнал 1, то триггер переходит в состояние 1, и после того, как сигнал S станет равен 0, триггер будет сохранять состояние 1. При подаче 1 на вход R триггер перейдет в состояние 0. Подача на оба входа логической единицы может привести к неоднозначному результату, поэтому такая комбинация входных сигналов запрещена.

Для хранения 1 байта информации необходимо 8 триггеров, для 1 килобайта — 8×1024 триггера. Таким образом, оперативная память современных компьютеров содержит миллионы триггеров. В целом же компьютер состоит из огромного числа логических элементов, образующих все его узлы и память.

Вопросы и задания

1. Проанализируйте схему, приведенную на следующем рисунке, и выпишите формулу для функции F:



2. Существует 16 логических элементов, имеющих два входа (16 логических функций от двух переменных). Реализуйте их схемы с помощью логических элементов **И**, **ИЛИ**, **НЕ**.
3. Может ли произвольная логическая схема быть построена только из логических элементов одного типа?
4. Постройте схему трехразрядного сумматора из одноразрядных.

Заключение

Итак, подведем итоги. XIX век подарил нам плеяду выдающихся математиков, которым удалось построить стройный аппарат алгебры логики. В первую очередь это Джордж Буль, Огастес де Морган, Джордж Венн.

Стремительный XX век расширил границы практического применения теоретических результатов алгебры логики. В частности, ее математический аппарат широко используется в информатике, при проектировании компьютеров, в теории алгоритмов, в целочисленном программировании и т. д.

Познакомившись с материалом данной главы, вы знаете теперь основные законы алгебры логики, сможете без ошибки построить отрицание к сложному высказыванию, сформулированному на любом национальном языке, знаете, из скольких элементов состоит сумматор, что такое триггер и зачем нам нужны такие необычные логические функции, как стрелка Пирса и штрих Шеффера.

Надеемся, что мы заинтересовали вас алгеброй логики — чрезвычайно красивой областью математики, применение которой будет наверняка расширяться.

Глава 4

Элементы теории алгоритмов

Понятие алгоритма является не только центральным понятием теории алгоритмов, не только одним из главных понятий математики вообще, но одним из главных понятий современной науки. Более того, сегодня, с наступлением эры информатики, алгоритмы становятся одним из важнейших факторов цивилизации.

В. А. Успенский

Благо везде и всюду зависит от соблюдения двух условий: 1) правильного установления цели всякого рода деятельности и 2) отыскания соответствующих средств, ведущих к этой цели.

Аристотель

Данная глава знакомит вас с относительно новой научной дисциплиной — теорией алгоритмов. Заметим сразу, что специалист в области теории алгоритмов — это прежде всего математик, уверенно владеющий аппаратом теории множеств, дискретной математики, высшей алгебры. Зарождение и развитие теории алгоритмов связано с фундаментальными достижениями выдающихся математиков: А. Тьюринга (Англия), А. Чёрча (США), Э. Поста (США), Дж. фон Неймана (США), А. А. Маркова (СССР), П. С. Новикова (СССР), А. Н. Колмогорова (СССР), Г. С. Цейтина (СССР), Ю. В. Матиясевича (СССР) и многих других.

Глава посвящена разъяснению одного из основных понятий математики и информатики — понятия алгоритма; в ней прослеживается взаимовлияние научных открытий математики и информатики в области теории алгоритмов.

§ 4.1. Понятие алгоритма. Свойства алгоритмов

Каждый из нас ежедневно решает задачи различной сложности: как быстрее добраться в школу или на работу в условиях нехватки времени; в каком порядке выполнять дела, намеченные на текущий день, и т. д. Некоторые задачи настолько сложны, что требуют длительных размышлений для нахождения решения (иногда решение так и не удается найти), другие задачи мы решаем автоматически, так как выполняем их ежедневно на протяжении многих лет (выключить звенящий будильник; почистить утром зубы; позвонить другу по телефону). В большинстве случаев решение каждой задачи можно подразделить на простые этапы.

Пример 1. Задача «Звонок другу по телефону» подразделяется на следующие этапы (шаги):

1. Поднять телефонную трубку.
2. Если услышал гудок, то набрать номер друга, иначе конец решения задачи с отрицательным результатом (телефон неисправен).
3. Определить тип гудков: «вызов» или «занято». Если «вызов», перейти к шагу 4, если «занято», перейти к шагу 6.
4. Дождаться восьми вызывающих гудков.

5. Если за это время абонент не поднял трубку, то конец решения задачи с отрицательным результатом (абонент не отвечает), иначе начать разговор. Задача «Позвонить другу» решена успешно.
6. Положить телефонную трубку; конец решения задачи с отрицательным результатом (абонент занят). □

Последовательность шагов, приведенная в примере 1, является *алгоритмом* решения задачи «Звонок другу по телефону». *Исполнитель* этого алгоритма — человек. Объекты алгоритма — телефон и телефонные сигналы.

Для решения любой задачи надо знать, что дано и что следует получить, т. е. у задачи есть исходные данные (некие объекты) и искомые результаты. Для получения результатов необходимо знать способ решения задачи, т. е. располагать алгоритмом. В базовом курсе информатики вы знакомились с основами теории алгоритмов, в частности, вы знаете следующее неформальное определение (понятие) алгоритма.

Определение 1. *Алгоритм* — это точная конечная система предписаний, определяющая содержание и порядок действий исполнителя над некоторыми объектами (исходными и промежуточными данными) для получения (после конечного числа шагов) искомого результата.

Аль-Хорезми (780–850 н. э.) — узбекский математик IX века; из европеизированного произношения имени аль-Хорезми возник термин «алгоритм»

Приведенное определение не является определением в математическом смысле слова, это описание понятия алгоритма, раскрывающее его сущность. Оно не является формальным потому, что в нем используются такие неуточняемые понятия, как «система предписаний», «действия исполнителя», «объект».

Понятие алгоритма, являющееся фундаментальным понятием математики и информатики, возникло задолго до появления вычислительных машин.

Первоначально под словом «алгоритм» понимали способ выполнения арифметических действий над десятичными числами. В дальнейшем это понятие стали использовать для обо-

значения любой последовательности действий, приводящей к решению поставленной задачи.

Приведем примеры двух алгоритмов, которые будут вам полезны на уроках математики.

Пример 2. Рассмотрим способ выписывания всех простых чисел в интервале от 1 до некоторого N. Этот способ носит название *«Решето Эратосфена»*, по имени древнегреческого ученого, впервые предложившего данный алгоритм.

Эратосфен (ок. 275–194 до н. э.) — один из самых разносторонних ученых античности. Особенно прославили Эратосфена труды по астрономии, географии и математике, однако он успешно трудился и в области филологии, поэзии, музыки и философии

Для этого выпишем подряд все натуральные числа от 1 до N. Возьмем первое число, большее 1 (это будет 2), и зачеркнем каждое второе число, начиная отсчет со следующего за двойкой числа. Затем возьмем первое незачеркнутое число, большее 2 (это будет 3), и зачеркнем каждое третье число, начиная отсчет с числа 3 + 1 (ранее зачеркнутые числа участвуют в отсчете). Далее возьмем первое незачеркнутое число, большее 3 (это будет 5), и зачеркнем каждое пятое число, начиная отсчет с числа 5 + 1. Продолжая так действовать, остановимся тогда, когда первое незачеркнутое число окажется больше $\sqrt{N}$. □

В результате применения этого алгоритма незачеркнутыми останутся все простые числа, не превосходящие N, и только они. Докажите это самостоятельно.

Пример 3. Дано вещественное число P. Требуется вычислить его квадратный корень с заданной точностью, например с m знаками после запятой.

Правило вычисления квадратного корня «в столбик» можно записать в следующем виде:

1. Запись числа P разделить на группы P_i по 2 цифры влево и вправо от запятой $\{P_1P_2...P_k,P_{k+1}...P_n\}$. Заметим, что самая левая и самая правая группы (P_1 и P_n) могут состоять из одной цифры. Группу P_n в этом случае дополнить до двух цифр, приписав 0 справа.

2. Подобрать такую цифру x, квадрат которой не превосходит P_1. Записать x в качестве первой цифры результата (искомого числа).
3. Вычислить $r = P_1 - x^2$.
4. Если нерассмотренных групп в числе P больше нет и $r = 0$ (т. е. мы вычислили корень точно), или если $r \neq 0$, но точность вычисления нас устраивает (у искомого числа получено m знаков после запятой), то конец алгоритма.
5. Если нерассмотренные группы есть, или точность вычисления нас не устраивает, то к r дописать справа цифры очередной (P_i) группы; получившееся число обозначить через b. При этом возможны следующие варианты:
   5.1) если очередная группа (P_i) стоит в записи числа первой после запятой, то дописать справа от x десятичную запятую;
   5.2) если все группы в записи числа уже обработаны, то в качестве очередной группы P_i взять «00», при этом если число P было целым, то дописать справа от x десятичную запятую.
6. Подобрать такую максимальную цифру y, что $(2 \cdot x' \cdot 10 + y) \cdot y \leq b$, где x' — целое число, совпадающее по записи с x без учета десятичной запятой. Тогда y будет очередной цифрой искомого числа.
7. Вычислить $r = b - (2 \cdot x' \cdot 10 + y) \cdot y$.
8. Приписать y справа к результату. Получившееся число обозначить через x.
9. Перейти к шагу 4. □

Любой алгоритм существует не сам по себе, а предназначен для определенного *исполнителя*. Алгоритм описывается в *командах исполнителя*, который этот алгоритм будет выполнять. Объекты, над которыми исполнитель может совершать действия, образуют так называемую *среду исполнителя*. *Исходные данные и результаты* любого алгоритма всегда принадлежат среде того исполнителя, для которого предназначен алгоритм.

Значение слова «алгоритм» очень схоже со значением слов «рецепт», «метод», «способ». Однако любой алгоритм, в отличие от рецепта или способа, обязательно обладает следующими свойствами.

1. *Дискретность.* Выполнение алгоритма разбивается на последовательность законченных действий-шагов. Только выполнив одно действие, можно приступать к исполнению следующего. Произвести каждое отдельное действие исполнителю предписывает специальное указание в записи алгоритма, называемое *командой.*
2. *Детерминированность.* Способ решения задачи однозначно определен в виде последовательности шагов, т. е. если алгоритм многократно применяется к одному и тому же набору исходных данных, то каждый раз получаются одни и те же промежуточные результаты и один и тот же выходной результат.
3. *Понятность.* Алгоритм не должен содержать предписаний, смысл которых может восприниматься исполнителем неоднозначно, т. е. запись алгоритма должна быть настолько четкой и полной, чтобы у исполнителя не возникало потребности в принятии каких-либо самостоятельных решений.

Алгоритм всегда рассчитан на выполнение «неразмышляющим» исполнителем.

Рассмотрим известный пример «бытового» алгоритма — алгоритм перехода улицы: «Посмотри налево. Если машин нет, дойди до середины улицы. Если есть, подожди, пока они проедут и т. д.». Представьте себе ситуацию: машина слева есть, но она не едет — у нее меняют колесо. Если вы думаете, что исполнитель алгоритма должен ждать, то вы поняли этот алгоритм. Если же вы решили, что улицу переходить можно, считая алгоритм подправленным в виду непредвиденных (по вашему мнению!) обстоятельств, то вы не усвоили понятие алгоритма.

4. *Результативность.* Содержательная определенность результата каждого шага и алгоритма в целом. При точном исполнении команд алгоритма процесс должен прекратиться за конечное число шагов, и при этом должен быть получен ответ на вопрос задачи. В качестве одного из возможных ответов может быть и установление того факта, что задача решений не имеет.

Свойство результативности содержит в себе свойство *конечности* — завершение работы алгоритма за конечное число шагов.

5. *Массовость.* Алгоритм пригоден для решения любой задачи из некоторого класса задач, т. е. алгоритм правильно работает на некотором множестве исходных данных, которое называется областью применимости алгоритма.

Проиллюстрируем свойства алгоритма на примере алгоритма вычисления квадратного корня «в столбик» (пример 3).

Массовость этого алгоритма заключается в том, что его можно применить к любому положительному рациональному числу. Данный процесс разбит на шаги, т. е. он *дискретен*. *Детерминированность* алгоритма вытекает из того, что каждая команда выполняется исполнителем однозначно и каждая команда снабжена указанием, какую команду выполнять следующей. *Понятность* алгоритма обеспечивается тем, что, во-первых, исполнителю известно, с чего начинать выполнение алгоритма (с команды номер 1), и, во-вторых, четко описано, какие из допустимых действий исполнителя надо выполнять на каждом шаге. *Результативность* алгоритма состоит в том, что он определяет процесс, приводящий к нахождению квадратного корня с заданной точностью вычислений для любого положительного рационального числа.

Вопрос. *Возможна ли ситуация, что способ решения задачи есть, но он не является алгоритмом?*

Ответ. Не каждый способ, приводящий к решению задачи, является алгоритмом. Например, опишем следующий способ (метод) проведения перпендикуляра к прямой MN, проходящего через заданную точку A:

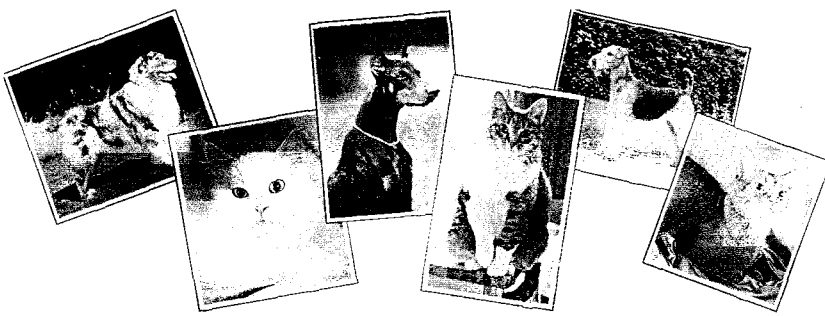
1. Отложить в обе стороны от точки A на прямой MN циркулем отрезки равной длины с концами B и C.
2. Увеличить раствор циркуля до радиуса, в полтора-два раза большего длины отрезков AB и AC.
3. Провести указанным раствором циркуля дуги окружностей с центрами B и C так, чтобы они охватили точку A и образовали две точки пересечения друг с другом (D и E).
4. Взять линейку, приложить ее к точкам D и E и соединить их отрезком.

При правильном построении отрезок пройдет через точку A и будет искомым перпендикуляром.

Указанный способ рассчитан на исполнителя-человека. Применяя его, человек, разумеется, построит искомый перпендикуляр. Но тем не менее этот способ алгоритмом не является. Прежде всего он не обладает свойством детерминированности. Так, в пункте 1 требуется от исполнителя сделать выбор отрезка *произвольной* длины (для построения точек B и C можно провести окружность произвольного радиуса r с центром в точке A). В пункте 2 требуется сделать выбор отрезка, в полтора-два раза большего длины отрезков AB и AC. В пункте 3 надо провести дуги, которые также однозначно не определены. Человек-исполнитель, применяющий данный способ к одним и тем же исходным данным (прямой MN и точке A) повторно, получит несовпадающие промежуточные результаты. Это противоречит требованию детерминированности алгоритма. □

Вопрос. *Существуют ли задачи, которые человек, вообще говоря, умеет решать, не зная при этом алгоритм их решения?*

Ответ. Да, можно привести пример и такой задачи. Например, перед человеком лежат фотографии кошек и собак. Человек должен определить, кошка или собака изображена на конкретной фотографии. Человек решает эту задачу на интуитивном уровне с высоким процентом правильных ответов. Но написать точный алгоритм решения данной задачи сегодня не представляется возможным, так как формализация этой проблемы чрезвычайно сложна и пока практически нереализуема. □



Дадим уточненное понятие алгоритма, которое опять же не является определением в математическом смысле

слова, но более формально описывает понятие алгоритма, раскрывающее его сущность.

Определение 2. *Алгоритм* — это конечная система правил, сформулированная на языке исполнителя, которая определяет последовательность перехода от допустимых исходных данных к конечному результату и которая обладает свойствами дискретности, детерминированности, результативности, конечности и массовости.

И. Кант (1724–1804), немецкий философ

Отметим, что для каждого исполнителя набор допустимых действий всегда ограничен — не может существовать исполнителя, для которого любое действие является допустимым. Перефразированное рассуждение И. Канта обосновывает сформулированное утверждение следующим образом: «Если бы такой исполнитель существовал, то среди его допустимых действий было бы создание такого камня, который он не может поднять. Но это противоречит допустимости действия «поднять любой камень».

Ограничение на набор допустимых действий означает, что для любого исполнителя имеются задачи, которые нельзя решить с его помощью.

При изучении алгоритмов важно разделять два понятия: *запись алгоритма* и *выполнение алгоритма*. В учебно-научной литературе термин «алгоритм» используется как в первом, так и во втором значении. Для более четкого изложения мы будем конкретизировать употребление этого термина.

В заключение данного параграфа сделаем три важных замечания.

1) Существует много разных способов для записи (описания) одного и того же алгоритма:

- текстовая форма записи;
- запись в виде блок-схемы;
- запись алгоритма на каком-либо алгоритмическом языке;
- представление алгоритма в виде машины Тьюринга или машины Поста.

Выбор способа записи алгоритма зависит от нескольких причин. Если для вас наиболее важна наглядность

записи алгоритма, то разумно использовать блок-схему. Если алгоритм небольшой, то его можно записать в текстовой форме. При этом команды могут быть пронумерованы (пример 1) или записаны в виде сплошного текста (пример 2).

2) Вне зависимости от выбранной формы записи элементарные шаги алгоритма (команды) при укрупнении объединяются в алгоритмические конструкции: *последовательные, ветвящиеся, циклические, рекурсивные*. В 1969 году Эдсгер В. Дейкстра в статье «Структуры данных и алгоритмы» доказал, что для записи любого алгоритма достаточно трех основных алгоритмических конструкций: последовательных, ветвящихся, циклических.

3) Если задача имеет алгоритмическое решение, то можно придумать множество различных способов ее решения, т. е. различных алгоритмов. Теория алгоритмов предоставляет аппарат анализа различных алгоритмов решения одной и той же задачи, на основе которого можно выбрать самый эффективный (наилучший) алгоритм.

Вопросы и задания

1. Почему кулинарный рецепт приготовления торта нельзя считать алгоритмом? Какими свойствами алгоритма не обладает кулинарный рецепт?
2. Составьте алгоритм сложения в столбик двух натуральных чисел. Предполагается, что операция сравнения двух натуральных чисел для человека является выполнимой.
3. Переформулируйте способ проведения перпендикуляра к прямой в заданной точке так, чтобы он стал алгоритмом.
4. Есть двое песочных часов: на 3 минуты и на 8 минут. Для приготовления эликсира бессмертия его надо варить ровно 7 минут. Как это сделать?
   Придумайте систему команд исполнителя «Колдун». Запишите последовательность команд этого исполнителя для приготовления эликсира.
5. Приведите примеры алгоритмов, использующих ветвящиеся алгоритмические конструкции.

6. Приведите примеры алгоритмов, использующих циклические алгоритмические конструкции.
7. Приведите примеры рекурсивных алгоритмов.
8. Составьте в виде блок-схемы алгоритм нахождения факториала числа N.
9. Докажите, что при помощи алгоритма «Решето Эратосфена» действительно можно найти все простые числа из указанного диапазона.

§ 4.2. Уточнение понятия алгоритма. Машина Тьюринга

Определения алгоритма, приведенные в § 4.1, не являются строгими, так как в них используются не определяемые точно термины, например «правило». Однако математики достаточно долго пользовались интуитивным понятием алгоритма, записывая алгоритмы примерно так, как в примерах 1 и 2. В рамках подобного определения были сформулированы и успешно применялись на практике алгоритмы для решения таких задач, как нахождение корней квадратных и кубических уравнений, решение систем линейных уравнений (метод Гаусса) и др.

4.2.1. Необходимость уточнения понятия алгоритма

Постепенно математики подходили к постановке и решению все более сложных задач. Так, например, Г. Лейбниц в XVII веке пытался построить общий алгоритм решения любых математических задач. В XX веке эта идея приобрела более конкретную форму: построить алгоритм проверки правильности любой теоремы при любой системе аксиом. Построить такие алгоритмы не удавалось, и математики выдвинули предположение: а вдруг для того или иного класса задач в принципе невозможно построить алгоритм решения? Следовательно, если алгоритма не существует, то они ищут то, чего нет.

На основе этого предположения возникло понятие *алгоритмически неразрешимой задачи* — задачи, для которой невозможно построить процедуру решения. Но для того, чтобы прекратить поиски решения задачи, относительно которой выдвинуто предположение о ее алго-

ритмической неразрешимости, надо было научиться математически строго доказывать факт отсутствия соответствующего алгоритма. А это возможно только в том случае, если существует строгое определение алгоритма. Поэтому возникла проблема: построить формальное определение алгоритма, аналогичное известному интуитивному понятию.

Попытки выработать формальное определение алгоритма привели в 20–30-х годах XX века к возникновению *теории алгоритмов*. В первой половине XX века разные математики (А. Тьюринг, Э. Пост, А. Н. Колмогоров, А. А. Марков и др.) предложили несколько подходов к формальному определению алгоритма: нормальный алгоритм Маркова, машина Тьюринга, машина Поста и т. д. В дальнейшем было показано, что все эти определения эквивалентны.

Мы рассмотрим два формальных определения алгоритма, введенных Э. Постом и А. Тьюрингом. Этими математиками для уточнения понятия алгоритма были предложены *абстрактные вычислительные конструкции*, которые позже были названы «машинами». А. Тьюринг описал свою «машину» в 1936 году. Аналогичную концепцию «машины» ввел позднее, в 1937 году, и независимо от Тьюринга американский математик Э. Пост.

А. Тьюринг (1912–1954)

Тьюринг признан одним из основателей информатики и теории искусственного интеллекта, его считают первым теоретиком современного программирования и, наконец, первым в мире хакером. Между прочим, его «хакерская деятельность» внесла во время второй мировой войны существенный вклад в победу союзных войск над германским флотом, а один из коллег Тьюринга однажды сказал: «Я не берусь утверждать, что мы выиграли войну благодаря Тьюрингу. Однако без него могли бы ее и проиграть».

У Алана Тьюринга целью создания такой абстрактной воображаемой машины было получение возможности доказательства существования или несуществования алгоритмов решения различных задач. Руководствуясь этой целью, Тьюринг искал как можно более простую, «бед-

ную» алгоритмическую схему, лишь бы она была универсальной.

Прежде чем мы начнем знакомиться с машиной Тьюринга, необходимо сделать два общих замечания относительно объектов, с которыми работают алгоритмы.

Замечание 1. Одной из причин расплывчатости интуитивного понятия алгоритма является разнообразие объектов, с которыми работают алгоритмы. В вычислительных алгоритмах объектами являются числа. В алгоритме шахматной игры объектами являются фигуры и их позиции на шахматной доске. В алгоритме форматирования текста — слова некоторого языка и правила переноса слов. Однако во всех этих и других случаях можно считать, что алгоритм имеет дело не с объектами реального мира, а с некоторыми изображениями этих объектов. Например, есть алгоритм сложения двух целых чисел. Результатом сложения числовых объектов 26 и 22 будет числовой результат 48. Но мы можем считать, что объектом этого алгоритма является входная последовательность, состоящая из пяти символов: «26 + 22», а результатом является последовательность, состоящая из двух символов: «48».

При этом мы исходим из того, что имеется набор из 11 различных символов {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +}. Используемые символы будем называть *буквами*, а их набор — *алфавитом*. В общем случае буквами могут служить любые символы, требуется только, чтобы они были различны между собой и чтобы их число было конечным.

Определение 3. Любая конечная последовательность букв из некоторого алфавита называется *словом* в этом алфавите. Количество букв в слове называется *длиной* слова. Слово, в котором нет букв, называется *пустым* словом. Оно часто изображается символом «Λ» или «a_0».

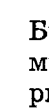
Так, алгоритм сложения двух целых чисел перерабатывает слово, которое состоит из двух слагаемых, разделенных символом «+», в слово, изображающее сумму.

Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова.

Определение 4. Слово, к которому применяется алгоритм, называется *входным* словом; слово, получаемое в результате работы алгоритма, называется *выходным*. Совокупность слов, к которым применим алгоритм, называется *областью применимости алгоритма*.

К сожалению, нельзя доказать, что все возможные объекты можно описать словами, так как само понятие объекта не было формально (то есть строго) определено. Но можно проверить, что для любого наугад взятого алгоритма, работающего не над словами, его объекты можно выразить так, что они становятся словами, а суть алгоритма от этого не меняется.

Замечание 2. Любой алфавит можно заменить другим. Такая замена называется *кодированием*. Например, пусть каждой букве из первого алфавита ставится в соответствие код, представляющий собой слово во втором алфавите. В качестве второго алфавита достаточно иметь алфавит из двух букв, так как любое слово из любого алфавита можно закодировать в двухбуквенном алфавите с гарантией однозначного восстановления исходного слова. Следовательно, любой алгоритм можно свести к алгоритму над словами в алфавите {0, 1}, а перед применением алгоритма входное слово следует закодировать, после применения алгоритма выходное слово надо раскодировать.



Будем считать, что алгоритмы работают со словами, и мы формально описываем объекты-слова, над которыми работают алгоритмы, в некотором алфавите.

4.2.2. Описание машины Тьюринга

Далее для уточнения понятия алгоритма следует формально описать действия над объектами-словами и порядок выполнения этих действий. В качестве такой формальной схемы мы и рассмотрим машину Тьюринга.

Вопрос. *Что же представляет собой машина Тьюринга?*

Ответ. Машина Тьюринга — это строгое математическое построение, математический аппарат (аналогичный, например, аппарату дифференциальных уравнений), созданный для решения определенных задач. Этот матема-

тический аппарат был назван «машиной» по той причине, что по описанию его составляющих частей и функционированию он похож на вычислительную машину. Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том, что ее запоминающее устройство представляет собой бесконечную ленту: у реальных вычислительных машин запоминающее устройство может быть как угодно большим, но обязательно конечным. Машину Тьюринга нельзя реализовать именно из-за бесконечности ее ленты. В этом смысле она мощнее любой вычислительной машины. □

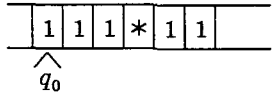
В каждой машине Тьюринга есть две части:

1) *неограниченная* в обе стороны *лента*, разделенная на ячейки;
2) *автомат* (головка для считывания/записи, управляемая программой).

С каждой машиной Тьюринга связаны *два конечных алфавита*: алфавит входных символов $A = \{a_0, a_1, \ldots, a_m\}$ и алфавит состояний $Q = \{q_0, q_1, \ldots, q_p\}$. (С разными машинами Тьюринга могут быть связаны разные алфавиты A и Q.) Состояние q_0 называется *пассивным*. Считается, что если машина попала в это состояние, то она закончила свою работу. Состояние q_1 называется *начальным*. Находясь в этом состоянии, машина начинает свою работу.

Входное слово размещается на ленте по одной букве в расположенных подряд ячейках. Слева и справа от входного слова находятся только пустые ячейки (в алфавит A всегда входит «пустая» буква a_0 — признак того, что ячейка пуста).

Автомат может двигаться вдоль ленты влево или вправо, читать содержимое ячеек и записывать в ячейки буквы. Ниже схематично нарисована машина Тьюринга, автомат которой обозревает первую ячейку с данными:



Автомат каждый раз «видит» только одну ячейку. В зависимости от того, какую букву a_i он видит, а также в зависимости от своего состояния q_j, автомат может выполнять следующие действия:

- записать новую букву в обозреваемую ячейку;
- выполнить сдвиг по ленте на одну ячейку вправо/влево или остаться на месте;
- перейти в новое состояние.

То есть у машины Тьюринга есть три вида команд. Каждый раз для очередной пары (q_j, a_i) машина Тьюринга может выполнить определенную команду в соответствии с программой.

Программа для машины Тьюринга представляет собой таблицу, в каждой клетке которой записана команда.

	a_0	a_1	...	a_i	...	a_n
q_0						
q_1						
...						
q_j				a_k \{ Л / П / Н \} q_m		
...						
q_p						

Клетка (q_j, a_i) определяется двумя параметрами — символом алфавита и состоянием машины. Команда представляет собой указание: какой символ записать в текущую ячейку, куда передвинуть головку чтения/записи, в какое состояние перейти машине. Для обозначения направления движения автомата используем одну из трех букв: «Л» (влево), «П» (вправо) или «Н» (неподвижен).

После выполнения автоматом очередной команды он переходит в состояние q_m (которое может в частном случае совпадать с прежним состоянием q_i). Следующую команду нужно искать в m-й строке таблицы на пересечении со столбцом a_t (букву a_t автомат видит после сдвига).

Если следить только за лентой, не обращая внимания на автомат, то мы увидим, что в результате выполнения каждой команды изменяются слова: какие-то буквы стираются и вместо них остаются пустые ячейки, в каких-то ячейках появляются новые буквы.

Договоримся, что когда лента содержит входное слово, то автомат находится против какой-то ячейки в со-

стоянии q_1. В процессе работы автомат будет перескакивать из одной клетки программы (таблицы) в другую, пока не дойдет до клетки, в которой записано, что автомат должен перейти в состояние q_0. Эта клетка называется *клеткой останова*. Дойдя до любой такой клетки, машина Тьюринга *останавливается*.

Если *клеток останова* в программе нет или машина в процессе работы на них не попадает, то считается, что машина Тьюринга *неприменима* к данному входному слову. Машина Тьюринга *применима* к входному слову только в том случае, если, начав работу над этим входным словом, она рано или поздно дойдет до одной из клеток останова.

Вопрос. *Прослеживая работу машины Тьюринга, мы можем только узнать, что она к данному слову применима. Но если она к данному слову неприменима, то такое прослеживание ничего не даст, так как в любой момент времени мы можем надеяться дойти до клетки останова. Как же выйти из этой ситуации?*

Ответ. Неприменимость машины Тьюринга не может быть выяснена прямым способом; в неприменимости можно убедиться только путем косвенных рассуждений. Например, если в программе нет клетки останова, то данная машина Тьюринга неприменима ни к одному слову. Или, например, в программе некой машины Тьюринга с алфавитом {0, 1} есть клетка останова, но первая строка программы (таблицы) имеет следующий вид:

	a_0	0	1
q_1	$a_0 \, П \, q_1$	$0 \, П \, q_1$	$1 \, П \, q_1$

Тогда, что бы автомат ни увидел на ленте, он ничего не меняет и сдвигается на шаг вправо, оставаясь всегда в состоянии q_1. А поскольку лента бесконечна, он никогда не остановится. □

4.2.3. Примеры машин Тьюринга

Несмотря на свое простое устройство, машина Тьюринга может выполнять все возможные преобразования слов, реализуя тем самым все возможные алгоритмы.

Задание 1. *Требуется построить машину Тьюринга, которая прибавляет единицу к числу на ленте. Входное слово состоит из цифр целого десятичного числа, записанных в последовательные ячейки на ленте. В начальный момент машина находится против самой правой цифры числа.*

Решение. Машина должна прибавить единицу к последней цифре числа. Если последняя цифра равна 9, то ее надо заменить на 0 и прибавить единицу к предыдущей цифре. Программа для данной машины Тьюринга может выглядеть так:

	a_0	0	1	2	3	4	...	7	8	9
q_1	1 Н q_0	1 Н q_0	2 Н q_0	3 Н q_0	4 Н q_0	5 Н q_0	...	8 Н q_0	9 Н q_0	0 Л q_1

В этой машине Тьюринга q_1 — состояние изменения цифры, q_0 — состояние останова. Если в состоянии q_1 автомат видит цифру 0..8, то он заменяет ее на 1..9 соответственно и переходит в состояние q_0, т. е. машина останавливается. Если же он видит цифру 9, то заменяет ее на 0, сдвигается влево, оставаясь в состоянии q_1. Так продолжается до тех пор, пока автомат не встретит цифру меньше 9. Если же все цифры были равны 9, то он заменит их нулями, в том числе запишет 0 на месте старшей цифры, сдвинется влево и в пустой клетке запишет 1. Затем перейдет в состояние q_0, т. е. остановится. □

Для краткости и наглядности записи программы машины Тьюринга можно условиться переход в состояние останова отмечать знаком «!». Тогда программа из задания 1 будет выглядеть так:

	a_0	0	1	2	3	4	...	7	8	9
q_1	1 Н!	1 Н!	2 Н!	3 Н!	4 Н!	5 Н!	...	8 Н!	9 Н!	0 Л q_1

Задание 2. *Построить машину Тьюринга для подсчета штрихов, которые располагаются подряд и образуют входное слово, при этом требуется стереть все штрихи и записать на ленте их количество в десятичной системе.*

Решение. Будем формировать искомое число на ленте слева от штрихов (без пустого символа между числом и штрихами). В начальный момент машина Тьюринга обозревает любой из штрихов и находится в состоянии q_1.

Запишем алгоритм решения задачи в текстовой форме.

1. Найти правый конец слова на ленте.
2. Если слово оканчивается штрихом, то стереть этот штрих, иначе остановить машину.
3. Прибавить к числу единицу и перейти к п. 1.

В соответствии с этим алгоритмом каждый момент времени на ленте находится слово вида

	b_k	b_{k-1}	...	b_1	b_0	/	/	...	/	

Здесь $b_k b_{k-1} \ldots b_1 b_0$ — десятичная запись числа стертых штрихов, сразу после этой записи находятся еще не стертые штрихи.

Выполнение алгоритма продолжается до тех пор, пока не будет стерт самый последний штрих, после чего, согласно условию из п. 2, машина Тьюринга остановится. Заметим, что каждый из трех пунктов алгоритма может быть реализован одним состоянием машины Тьюринга:

- состояние q_1 — автомат ищет правый конец слова;
- состояние q_2 — автомат стирает штрих;
- состояние q_3 — автомат ищет правый конец числа и прибавляет к числу единицу.

Ниже приведена программа машины Тьюринга для решения предлагаемой задачи.

	a_0	0	1	2	3	...	8	9	/
q_1	a_0 Л q_2	0 П q_1	1 П q_1	2 П q_1	3 П q_1	...	8 П q_1	9 П q_1	/ П q_1
q_2	a_0 Н q_0	0 Н q_0	1 Н q_0	2 Н q_0	3 Н q_0	...	8 Н q_0	9 Н q_0	a_0 Л q_3
q_3	1 П q_1	1 П q_1	2 П q_1	3 П q_1	4 П q_1	...	9 П q_1	0 Л q_3	/ Л q_3

□

Проанализировав программы машин Тьюринга из заданий 1 и 2, можно сделать вывод, что программа машины Тьюринга для решения сложной задачи может состоять из композиций (объединений) программ машин Тьюринга для решения элементарных подзадач исходного

алгоритма. Таким же методом выделения в исходной задаче подзадач мы пользуемся при составлении алгоритмов в привычной для нас алгоритмической схеме (запись алгоритма в виде блок-схемы, в текстовом виде и т. д.).

Богатство возможностей машины Тьюринга проявляется в том, что если какие-то алгоритмы A и B реализуются машинами Тьюринга, то можно построить машины Тьюринга, реализующие различные композиции алгоритмов A и B. Например, «Выполнить A, затем выполнить B» или «Выполнить A. Если в результате получилось слово «да», выполнить B. В противном случае не выполнять B» или «Выполнять поочередно A, B, пока B не даст ответ 0».

Очевидно, что такие композиции также являются алгоритмами, поэтому их реализация посредством машины Тьюринга подтверждает, что конструкция Тьюринга является универсальным исполнителем.

4.2.4. Формальное описание алгоритма. Математическое описание машины Тьюринга

Описывая различные алгоритмы для машины Тьюринга и доказывая реализуемость всевозможных композиций алгоритмов, Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции, что позволило ему выступить со следующим тезисом.

Тезис Тьюринга. Всякий алгоритм может быть реализован соответствующей машиной Тьюринга.

В тезисе Тьюринга речь идет, с одной стороны, о понятии алгоритма, которое не является точным математическим понятием, с другой стороны, о точном математическом понятии — машине Тьюринга. Настало время показать, что машина Тьюринга — математическое понятие, т. е. она может получить точное математическое определение.

Определение 5. *Машиной Тьюринга* (*МТ*) называется система вида $\{A, a_0, Q, q_1, q_0, T, \tau\}$,
где A — конечное множество, называемое *алфавитом МТ*,

$a_0 \in A$ — *пустая буква* алфавита,
Q — конечное множество, элементы которого называются *состояниями МТ*,
$q_1 \in Q$ — *начальное состояние МТ*,
$q_0 \in Q$ — *пассивное состояние*, или *состояние останова МТ*,
$T = \{Л, Н, П\}$ — *множество сдвигов МТ*,
τ — *программа МТ*, т. е. $\tau: A \times Q \backslash \{q_0\} \to A \times T \times Q$.

Нетрудно убедиться, что в этом определении фигурируют только математические и логические термины (или символы), например множество, конечное множество, элемент множества, принадлежность множеству, произведение множеств. И никакие другие, нематематические или нелогические понятия в приведенной формулировке не используются.

Тезис Тьюринга является *основной гипотезой теории алгоритмов в форме Тьюринга*. Одновременно этот тезис приводит к формальному определению алгоритма.

Определение 6. *Алгоритм* (по Тьюрингу) — программа для машины Тьюринга, приводящая к решению поставленной задачи.

Основную гипотезу теории алгоритмов невозможно доказать, потому что она оперирует неформальным понятием алгоритма. Однако обоснование гипотезы есть: все алгоритмы, придуманные человечеством в течение столетий, действительно могут быть реализованы машинами Тьюринга.

Чтобы опровергнуть основную гипотезу, необходимо придумать такой алгоритм, который невозможно было бы реализовать на машине Тьюринга. Пока такого алгоритма нет.

Выше мы с вами говорили, что каждый алгоритм предназначен для какого-то конкретного исполнителя, у каждого исполнителя есть своя система команд, есть свой круг задач. Тьюрингом же был построен *универсальный исполнитель*, который может решить любую известную задачу. Этот фундаментальный результат был получен в то время, когда универсальных вычислительных машин еще не существовало. Более того, сам

факт построения воображаемого универсального исполнителя позволил высказать предположение о целесообразности построения универсальной вычислительной машины, которая бы могла решать любые задачи при условии соответствующего кодирования исходных данных и разработки соответствующей программы действий исполнителя.

Тезис Тьюринга всегда будет нацелен в будущее, так как теперь можно доказывать существование или несуществование алгоритмов для решения любых возникающих задач. Если поиск алгоритмического решения наталкивается на препятствие, то математики пытаются использовать это препятствие для доказательства невозможности решения, опираясь на основную гипотезу теории алгоритмов.

Вопросы и задания

1. Докажите, что любой алфавит можно заменить двухбуквенным алфавитом.
2. Покажите, что машина Тьюринга обладает всеми свойствами алгоритма.
3. Запишите программу машины Тьюринга из задания 2 с использованием знака «!» для обозначения перехода в состояние останова.
4. Постройте машину Тьюринга для решения следующей задачи: во входном слове все буквы «а» заменить на буквы «б».

§ 4.3. Машина Поста как уточнение понятия алгоритма

Почти одновременно с Тьюрингом американский математик Эмиль Пост в 1937 году предложил иную абстрактную машину, характеризующуюся еще большей простотой, чем машина Тьюринга. Это строгое математическое построение было также предложено в качестве уточнения понятия алгоритма.

Определение 7. *Алгоритм* (по Посту) — программа для машины Поста, приводящая к решению поставленной задачи.

Э. Пост
(1897–1954)

Американский математик и логик. Им получен ряд фундаментальных результатов в математической логике; он одним из первых ввел формальное определение алгоритма в терминах «абстрактной вычислительной машины»; кроме того, первым (одновременно с А. А. Марковым) привел доказательства алгоритмической неразрешимости ряда проблем математической логики и алгебры и сформулировал основной тезис теории алгоритмов о возможности описать любой конкретный алгоритм посредством этого определения.

Тезис Поста. Всякий алгоритм представим в форме машины Поста.

Этот тезис одновременно является формальным определением алгоритма.

Тезис Поста является гипотезой. Его невозможно строго доказать (так же, как и тезис Тьюринга), потому что в нем фигурирует, с одной стороны, интуитивное понятие «всякий алгоритм», а с другой стороны — точное понятие «машина Поста». В теории алгоритмов доказано, что машина Поста и машина Тьюринга эквивалентны по своим возможностям.

Вопрос. *Что представляет собой машина Поста?*

Ответ. В машине Поста в ячейках бесконечной ленты можно записывать всего два знака: 0 и 1 (ставить метку в ячейку или стирать метку). Это ограничение не влияет на ее универсальность, так как любой алфавит может быть закодирован двумя знаками.

Кроме ленты в машине Поста имеется каретка (головка чтения/записи), которая:

- умеет двигаться вперед, назад и стоять на месте;
- умеет читать содержимое, стирать и записывать 0 или 1;
- управляется *программой*.

Как и машина Тьюринга, машина Поста может находиться в различных состояниях, но каждому состоянию соответствует не строка состояния с клетками, а некоторая команда одного из следующих шести типов (в синтаксисе команд указывается номер строки, поэтому все строки в программе пронумерованы):

1. Записать 1 (отметку), перейти к i-й строке программы.
2. Записать 0 (стереть отметку), перейти к i-й строке программы.
3. Выполнить сдвиг влево, перейти к i-й строке программы.
4. Выполнить сдвиг вправо, перейти к i-й строке программы.
5. Останов.
6. Если 0, то перейти к i-й строке программы, иначе перейти к j-й строке программы.

Состояние машины — это состояние ленты и положение каретки. Приведем список недопустимых действий, ведущих к аварийной остановке машины:

- попытка записать 1 (отметку) в заполненную ячейку;
- попытка стереть отметку в пустой ячейке;
- уход каретки в бесконечность (вообще говоря, это трудно назвать остановкой, но бессмысленное повторение одних и тех же действий — зацикливание — ничуть не лучше вышеперечисленного).

«Машиной» эта математическая конструкция называется потому, что в ней используются некоторые понятия реальных машин — *память*, *команда* и пр. □

Машина Поста, несмотря на внешнюю простоту, может производить различные вычисления, для чего надо задать начальное состояние каретки и программу, которая эти вычисления сделает. Условимся каждую строку программы обозначать номером. В каждой строке программы записывается ровно одна команда. Команды машины будем обозначать следующим образом:

$\rightarrow a$	— шаг вправо, перейти к строке с номером a;
$\leftarrow a$	— шаг влево, перейти к строке с номером a;
V a	— записать отметку, перейти к строке с номером a;
X a	— стереть отметку, перейти к строке с номером a;
? a; b	— просмотреть ячейку; если в ячейке находится 0, то перейти к строке с номером a, иначе — к строке с номером b;
!	— останов.

Вопрос. *Чем отличаются состояния в машине Тьюринга от состояний в машине Поста?*

Ответ. В машине Тьюринга состояние определяет, что следует записать в обозреваемую ячейку для каждого определенного символа, задает характер движения головки и, наконец, указывает новое состояние машины.

В машине Поста состояние описывает местонахождение каретки и состояние ленты. □

Пример 4. На ленте проставлена отметка в одной-единственной ячейке. Каретка стоит на некотором расстоянии слева от этой ячейки. Необходимо подвести каретку к ячейке, стереть отметку и остановить каретку слева от нее.

Сначала попробуем описать алгоритм обычным языком. Поскольку нам известно, что каретка стоит напротив пустой ячейки, но неизвестно, сколько шагов нужно совершить до непустой ячейки, мы можем сразу сделать шаг вправо, проверить, заполнена ли ячейка, после чего повторять эти действия до тех пор, пока не наткнемся на заполненную ячейку. Как только мы ее найдем, мы выполним операцию стирания, после чего нужно будет лишь сместить каретку влево и остановить выполнение программы.

Программа для машины Поста:

1. → 2
2. ? 1; 3
3. X 4
4. ← 5
5. ! □

Вопросы и задания

1. Начальное состояние: лента машины Поста пуста. Что будет находиться на ленте в результате работы следующей программы?
   1. V 2
   2. → 3
   3. ← 1
2. Известно, что на ленте машины Поста находится метка. Напишите программу, которая находит ее.
3. На ленте имеется массив из n отмеченных ячеек. Каретка обозревает крайнюю левую отметку. Справа от данного массива на расстоянии в m ячеек находится еще одна от-

метка. Составьте для машины Поста программу, придвигающую данный массив к данной ячейке.

4. На ленте расположены два массива разной длины. Каретка обозревает крайний элемент одного из них. Составьте программу для машины Поста, сравнивающую длины массивов и стирающую больший из них. Отдельно продумайте случай, когда длины массивов равны.

§ 4.4. Алгоритмически неразрешимые задачи и вычислимые функции

В предыдущих параграфах было введено понятие алгоритмически неразрешимой задачи, да и наше воображение допускает, что, наверное, существуют задачи, которые невозможно решить. А что же представляют собой такие задачи? Приведем несколько примеров алгоритмически неразрешимых задач.

Пример 5. В начале XX века известный немецкий математик Давид Гильберт в 1900 г. сформулировал 23 математические проблемы. Сегодня решение (даже частичное) какой-либо проблемы Гильберта расценивается во всем мире как высшее математическое достижение. Десятая проблема Гильберта о диофантовых уравнениях (в упрощенной формулировке) звучит так:

Дано произвольное алгебраическое уравнение $P(x_1, x_2, \ldots, x_n) = 0$, где P — многочлен с целыми коэффициентами (например, $ax_1^2 + bx_2^2 + cx_3^3 = 0$). Требуется выяснить, существует ли у данного уравнения решение в целых числах.

Иная формулировка: требуется выработать алгоритм, позволяющий для любого диофантова уравнения выяснить, имеет ли оно целочисленное решение.

Д. Гильберт (1862–1943)

Его называют последним всесторонним математиком и самым замечательным учителем математиков XX века. Вундеркиндом он не был, а был типичным «классиком», т. е. Гильберт поочередно старался понять каждую область математики на всю ее глубину и решить в ней те задачи, которые его интересовали.

В 1970 г. советский математик Ю. В. Матиясевич доказал невозможность построения алгоритма для решения этой задачи.

Замечание. Если известно, что решение в целых числах есть, то алгоритм отыскания этого решения *существует*. □

Матиясевич Юрий Владимирович (р. 1947), воспитанник школы-интерната им. А. Н. Колмогорова. Член-корреспондент РАН с 1997 г., специалист в области математической логики, теории алгоритмов, дискретной математики.

Ю. В. Матиясевич

Пример 6. По описанию алгоритма A и аргументу x необходимо выяснить, остановится ли алгоритм A, если x является входным данным. Эта задача (ее называют *проблемой останова*) является алгоритмически неразрешимой. Докажем это.

Для доказательства воспользуемся методом «от противного». Пусть универсальный алгоритм решения подобной задачи существует. Рассмотрим класс алгоритмов, обрабатывающих некоторые тексты, в том числе и тексты алгоритмов. В силу существования универсального алгоритма существует и алгоритм, который для алгоритма из упомянутого класса определяет, остановится ли он на своем собственном тексте или нет (аргумент x в этом случае — текст самого алгоритма и его можно не указывать). Назовем такой алгоритм B. Построим следующий алгоритм C, входными данными для которого будет являться текст алгоритма A, обрабатывающего свой текст:

1. Выполнить алгоритм B над A.
2. Если алгоритм B определил, что A на своем тексте остановится, то перейти к шагу 1, в противном случае — к шагу 3.
3. Конец алгоритма C.

Применим алгоритм C к самому себе, т. е. входными данными для него будет текст алгоритма C. Пусть алгоритм B определил, что алгоритм C остановится при анализе своего текста. Но тогда после выполнения п. 2 мы снова перейдем к п. 1, т. е. на самом деле C в этом случае «зацикливается». Пусть, наоборот, алгоритм B опре-

делил бесконечность алгоритма C. Но тогда после выполнения п. 2 мы перейдем к п. 3 и алгоритм C завершится. Таким образом, так называемая «проблема *самоприменимости* алгоритма» неразрешима. Значит, наше предположение о существовании алгоритма B, решающего нашу задачу, неверно. □

Важным практическим следствием доказанного факта является невозможность создания универсального (пригодного для любой программы) алгоритма отладки программы.

Однако для конкретных алгоритмов и некоторых классов алгоритмов проблему останова и/или отладки соответствующей программы решить можно. Так, для программы, состоящей только из линейных конструкций, легко показать, что она всегда закончит свою работу.

Пример 7. Сформулированная Лейбницем в XVII веке *проблема проверки правильности любых математических утверждений* также является алгоритмически неразрешимой. Великий немецкий математик и философ Г. Лейбниц (1646–1716) мечтал о построении общего метода решения любой математической задачи. Лейбницу не удалось построить такой алгоритм, однако он считал, что наступит время, когда алгоритм будет найден. Проблема, поставленная Лейбницем, получила уточнение в виде одной из важнейших проблем математической логики — *проблемы распознавания выводимости*.

Как известно, в современной математике почти все математические теории строятся на аксиоматической основе. Суть соответствующего аксиоматического метода состоит в следующем: все предложения (теоремы) данной теории получаются посредством формально-логического вывода из нескольких предложений (аксиом), принимаемых в данной теории без доказательства. Ранее других была осуществлена аксиоматизация геометрии.

Пусть у нас есть посылка R и следствие S, записанные в виде слов (формул) в некотором специальном алфавите, состоящем из букв, скобок, знаков математиче-

ских и логических операций и т. д. Процесс вывода следствия S из посылки R может быть описан в виде процесса формальных преобразований слов; считаем, что система допустимых преобразований указана, т. е. построено логическое исчисление.

Проблему распознавания выводимости можно сформулировать следующим образом: *Существует ли для любых двух слов (формул) R и S дедуктивная цепочка, ведущая от R к S?* Решение понимается в смысле существования алгоритма, дающего ответ на этот вопрос для любых слов R и S.

Несмотря на долгие и упорные усилия многих крупных специалистов, трудности, связанные с таким построением, оказались непреодолимыми. В рамках теории алгоритмов был получен отрицательный ответ на вопрос об алгоритмической разрешимости этой проблемы. В 1936 году американский математик Чёрч доказал следующую теорему.

Теорема Чёрча. Проблема распознавания выводимости алгоритмически неразрешима.

Тем самым выяснилась не только причина безуспешности всех прошлых попыток создания соответствующего алгоритма, но и доказана бессмысленность дальнейших попыток. □

Вопрос. *Какие методы используются для доказательства алгоритмической неразрешимости?*

Ответ. Обычно алгоритмическая неразрешимость новых задач доказывается *методом сведения* к этим задачам известных алгоритмически неразрешимых задач. Тем самым доказывается, что если бы была разрешима новая задача, то можно было бы решить и заведомо неразрешимую задачу. Применяя метод сведения, обычно ссылаются на искусственные задачи, которые не представляют самостоятельного интереса, но для которых легко непосредственно доказать их неразрешимость. □

Но вернемся вновь к задачам, которые имеют алгоритмическое решение. В предыдущих параграфах мы показали, что если задача имеет решение, то можно написать и машину Тьюринга, и машину Поста для решения этой задачи. При этом на входные данные наклады-

ваются формальные ограничения (входные слова кодируются в некотором алфавите), а сами алгоритмы сконструированы в разных алгоритмических моделях (схемах). Возникает вопрос: можно ли в принципе говорить об общих свойствах алгоритмов при такой конкретизации?

В теории алгоритмов строго доказано, что любой алгоритм, описанный в одной модели, может быть описан и в другой. Такая взаимная сводимость алгоритмических моделей позволила создать систему понятий, не зависящую от выбора конкретной формализации понятия алгоритма. В основе этой системы лежит понятие *вычислимой функции*.

Заметим, что относительно каждого алгоритма A можно сказать, что он вычисляет значение функции F_A (реализует функцию F_A) при некоторых значениях входных величин.

Определение 8. *Функция*, вычисляемая некоторым алгоритмом, называется *вычислимой функцией* (*алгоритмически вычислимой*).

Введенное понятие вычислимой функции, так же, как и понятие алгоритма, является интуитивным.

Фактически алгоритм — это способ задания функции. Функции могут задаваться и другими способами, например таблицей или формулой. Однако существуют и такие задачи, в которых связь между входными и выходными параметрами настолько сложна, что нельзя составить алгоритм преобразования входных данных в результат. Такие функции являются не вычислимыми.

Пример 8[1]. Рассмотрим функцию

$$h(n) = \begin{cases} 1, \text{ если в десятичном разложении } \pi \text{ есть отрезок} \\ \quad \text{из } n \text{ девяток, окруженный недевятками;} \\ 0 \text{ в противном случае.} \end{cases}$$

Для каждого конкретного n можно подсчитать значение функции. Однако нам не известен общий способ, позволяющий вычислить значение функции для произвольного n. Анализ первых 800 знаков разложения π показывает лишь, что $h(n) = 1$ для $n = 0, 1, 2, 6$. Если не

[1] Пример приведен в книге В. А. Успенского «Машина Поста».

существует общего правила вычисления этой функции, то не существует и алгоритма, реализующего эту функцию, т. е. решающего поставленную проблему. □

Понятия алгоритма и вычислимой функции являются, пожалуй, наиболее фундаментальными понятиями информатики и математики. Систематическое изучение алгоритмов и различных моделей вычислений привело к созданию особой дисциплины, пограничной между математикой и информатикой — *теории алгоритмов*, в которой выделен раздел «*теория вычислимости*».

Теория вычислимых (с помощью компьютеров) функций появилась в 30-е годы XX столетия, когда никаких компьютеров еще не было. Первые компьютеры появились в 40-х годах, и их появление стало возможным именно благодаря достижениям теории вычислимости. Так, в рамках теории алгоритмов было сформулировано понятие вычислительной машины и было показано, что *для осуществления всевозможных преобразований информации вовсе не обязательно строить каждый раз специализированные вычислительные устройства*: все это можно сделать на одном *универсальном устройстве* при помощи подходящей программы и соответствующего кодирования.

Вопросы и задания

1. Сравните определение функции из курса математики с определением вычислимой функции. Укажите общее и различное.
2. Приведите пример алгоритма, программная реализация которого затруднена.
3. Используя рассуждения примера 6, докажите, что невозможно создать универсальный (пригодный для любой программы) алгоритм отладки программы.
4. Приведите примеры универсальных исполнителей.
5. Выпишите хронологию фундаментальных достижений (с указанием фамилий авторов и дат их жизни) в области теории алгоритмов. Для каждого ученого вычислите, на каком году жизни он выполнил работу, приведшую к фундаментальным результатам в теории алгоритмов. Полученные результаты представьте в виде таблицы.

§ 4.5. Понятие сложности алгоритма

В предыдущих параграфах говорилось, что если для решения задачи существует один алгоритм, то можно придумать и много других алгоритмов для решения этой же задачи.

Вопрос. *Какой алгоритм лучше подходит для решения конкретной задачи? По каким критериям следует выбирать алгоритм из множества возможных?*

Ответ. Как правило, мы высказываем суждение об алгоритме на основе его оценки исполнителем-человеком. Алгоритм кажется нам сложным, если даже после внимательного его изучения мы не можем понять, что же он делает. Мы можем назвать алгоритм сложным и запутанным из-за того, что он обладает разветвленной логической структурой, содержащей много проверок условий и переходов. Однако для компьютера выполнение программы, реализующей такой алгоритм, не составит труда, так как он выполняет одну команду за другой, и для компьютера неважно — операция ли это умножения или проверка условия.

Более того, мы можем написать громоздкий алгоритм, в котором выписаны подряд повторяющиеся действия (без использования циклической структуры). Однако с точки зрения компьютерной реализации практически нет никакой разницы, использован ли в программе оператор цикла (например, 10 раз на экран выводится слово «Привет») или 10 раз последовательно выписаны операторы вывода на экран слова «Привет». Поэтому для оценки эффективности алгоритмов введено понятие сложности алгоритма. □

Определение 9. *Вычислительным процессом*, порожденным алгоритмом, называется последовательность шагов алгоритма, пройденных при исполнении этого алгоритма.

В дальнейшем будем понимать под *сложностью алгоритма* количество элементарных действий в вычислительном процессе этого алгоритма, как функцию от исходных данных.

Обратите внимание, именно в вычислительном процессе, а не в самом алгоритме. Очевидно, для сравнения сложности разных алгоритмов необходимо, чтобы слож-

ность подсчитывалась в одних и тех же элементарных действиях.

Определение 10. *Временна́я сложность алгоритма* — это время T, необходимое для его выполнения в зависимости от исходных данных. Оно равно произведению числа элементарных действий k на среднее время выполнения одного действия t: $T = kt$.

Поскольку t зависит от исполнителя, реализующего алгоритм, то естественно считать, что сложность алгоритма в первую очередь определяется значением k. Очевидно, что в наибольшей степени количество операций при выполнении алгоритма зависит от количества обрабатываемых данных. Действительно, для упорядочивания по алфавиту списка из 100 фамилий требуется существенно меньше операций, чем для упорядочивания списка из 100 000 фамилий. Поэтому сложность алгоритма выражают в виде функции от объема входных данных.

Пусть есть алгоритм A. Для него существует параметр n, характеризующий объем обрабатываемых алгоритмом данных, этот параметр часто называют *размерностью задачи*. Обозначим через $T(n)$ время выполнения алгоритма в худшем случае, через f — некую функцию от n.

Определение 11. Будем говорить, что $T(n)$ алгоритма имеет порядок роста $f(n)$, или алгоритм имеет *теоретическую сложность* $O(f(n))$ (читается «о большое от $f(n)$»), если для $T(n)$ найдется такая константа c, что, начиная с некоторого n_0, выполняется условие $T(n) \le cf(n)$. Здесь предполагается, что функция $f(n)$ неотрицательна, по крайней мере при $n \ge n_0$.

Так, например, алгоритм, выполняющий только операции чтения данных и занесения их в оперативную память, имеет *линейную* сложность $O(n)$. Алгоритм сортировки методом прямого выбора, как это будет показано ниже, имеет *квадратичную* сложность $O(n^2)$, так как при сортировке любого массива этот алгоритм будет выполнять $(n^2 - n)/2$ операций сравнения (при этом операций перестановок вообще может не быть, например, на упорядоченном массиве). А сложность алгоритма умно-

жения матриц (таблиц) размера $n \times n$ будет уже *кубической* $O(n^3)$, так как для вычисления каждого элемента результирующей матрицы требуется n умножений и $n - 1$ сложений, а всего этих элементов n^2.

Для решения задачи могут быть разработаны алгоритмы различной сложности. Логично воспользоваться лучшим среди них, т. е. имеющим наименьшую сложность.

Наряду со сложностью важной характеристикой алгоритма является *эффективность*. Под эффективностью понимается выполнение следующего требования: не только весь алгоритм, но и каждый шаг его должны быть такими, чтобы исполнитель был способен выполнить их за разумное время. Например, если алгоритм, выдающий прогноз погоды на ближайшие сутки, будет выполняться неделю, то такой алгоритм просто-напросто никому не нужен.

Если мы рассматриваем алгоритмы, реализующиеся на компьютере, то к требованию выполнения за разумное время прибавляется требование выполнения в ограниченном объеме оперативной памяти.

Известно, что во многих языках программирования нет операции возведения в степень (x^n), следовательно, алгоритм возведения в целую степень программисту надо реализовывать самостоятельно. Операция возведения в степень выражается через операции умножения; с ростом показателя степени, естественно, растет количество операций умножения, которые при выполнении занимают достаточно долго время процессора. При реализации алгоритма возведения в степень «в лоб» надо выполнить $(n - 1)$ операцию умножения. Существует ли более быстрый универсальный способ?

Метод быстрого вычисления натуральной степени n вещественного числа x

Этот метод был описан еще до нашей эры в Древней Индии.

1. Записать n в двоичной системе счисления.
2. Заменить в этой записи каждую 1 парой букв KX, а каждый 0 — буквой K.
3. Вычеркнуть крайнюю левую пару KX.

4. Полученная строка, читаемая слева направо, дает правило быстрого вычисления x^n, если букву K рассматривать как операцию возведения результата в квадрат, а букву X — как операцию умножения результата на x. В начале результат равен x. △

Пример 9. Возвести x в степень $n = 100$.

1. Перевести число n в двоичную систему счисления: $n = 100_{10} = 1100100_2$.
2. Построить последовательность $KXKKXKKKXKK$.
3. Вычеркнуть KX слева $\Rightarrow$ $KXKKKXKK$.
4. Вычислить искомое значение:
   K: возвести x в квадрат $\Rightarrow x^2$,
   X: умножить результат на $x \Rightarrow x^3$,
   K: возвести результат в квадрат $\Rightarrow x^6$,
   K: возвести результат в квадрат $\Rightarrow x^{12}$,
   K: возвести результат в квадрат $\Rightarrow x^{24}$,
   X: умножить результат на $x \Rightarrow x^{25}$,
   K: возвести результат в квадрат $\Rightarrow x^{50}$,
   K: возвести результат в квадрат $\Rightarrow x^{100}$.

Таким образом, мы вычислили сотую степень числа x всего за 8 умножений. Этот метод достаточно эффективный, и он не требует дополнительной оперативной памяти для хранения промежуточных результатов. Однако заметим, что этот метод не всегда самый быстрый. □

Сама операция умножения реализуется в процессоре не «в лоб», а через эффективные рекурсивные алгоритмы. Мы же рассмотрим алгоритм «быстрого» умножения, который был известен еще в Древнем Египте, его также называют «русским» или «крестьянским» методом, на конкретном примере.

Пример 10. Умножим 23 на 43 «русским» методом.

23 × 43		(нечетное)	Первый столбец состоит из результатов последовательного умножения первого сомножителя (23) на 2. Второй столбец представляет собой результаты последовательного целочисленного деления второго сомножителя (43) на 2
46	21	(нечетное)	
92	10		
184	5	(нечетное)	
368	2		
736	1	(нечетное)	

Результат равен сумме чисел первого столбца, рядом с которыми во втором столбце стоят нечетные числа.
Ответ: $23 \times 43 = 23 + 46 + 184 + 736 = 989$. □

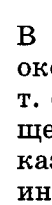
Вопросы и задания

1. Подсчитайте сложность алгоритма перемножения двух натуральных чисел столбиком при условии, что одно из чисел состоит из n десятичных цифр, а второе — из m десятичных цифр.
2. Подсчитайте сложность алгоритма умножения двух натуральных чисел «русским» методом при условии, что одно из чисел состоит из n десятичных цифр, а второе — из m десятичных цифр.
3. Приведите примеры алгоритмов, имеющих линейную сложность.
4. Пусть для некоторого алгоритма $T(n) = an^2 + bn + d$, где $a > 0$, b, d — отличные от нуля константы. Докажите, что такой алгоритм имеет сложность $O(n^2)$. Для этого подберите неотрицательную константу c так, что, начиная с некоторого n_0, $T(n) \le cn^2$.

§ 4.6. Анализ алгоритмов поиска

В этом параграфе мы будем рассматривать классические задачи поиска, анализировать алгоритмы, реализующие эти задачи, т. е. изучать свойства алгоритмов. Из большого множества алгоритмов поиска мы рассмотрим только два алгоритма, предназначенных для решения этой востребованной задачи.

Рассматривая различные алгоритмы решения одной и той же задачи, полезно проанализировать, сколько вычислительных ресурсов они требуют (время работы, память), и выбрать наиболее эффективный. Однако вначале надо договориться, какая модель вычислений будет использоваться. Будем считать, что наши алгоритмы выполняются на обычной однопроцессорной машине с произвольным доступом к памяти (данным).



В алгоритмах поиска существует две возможности окончания работы: либо поиск оказался *удачным*, т. е. позволил определить положение соответствующего элемента, либо он оказался *неудачным*, т. е. показал, что необходимого элемента в данном объеме информации нет.

Хотя целью поиска является значение элемента, алгоритмы поиска в случае удачного окончания выдают местоположение искомого элемента, например номер элемента в массиве.

В качестве критерия оценки алгоритма мы будем использовать такую характеристику, как *сложность*.

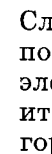
4.6.1. Последовательный поиск в неупорядоченном массиве

Сформулируем алгоритм последовательного поиска в неупорядоченной последовательности (неупорядоченном массиве). Очевидно, что этот алгоритм можно применять и для поиска в упорядоченном массиве.

Алгоритм последовательного поиска в неупорядоченном массиве

Имеется массив $a[1..n]$, требуется найти элемент массива, равный P.

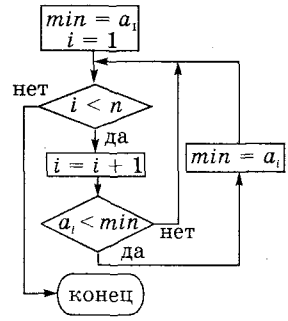
1. Установить $i = 1$.
2. Если $a_i = P$, алгоритм окончен удачно.
3. Увеличить i на 1.
4. Если $i \le n$, то перейти к шагу 2. В противном случае алгоритм окончен неудачно. △



Сложность алгоритмов поиска естественно оценивать по числу сравнений элементов массива с искомым элементом. В худшем случае (искомый элемент P стоит на последнем месте или отсутствует) сложность алгоритма будет равна n.

Усложним задачу. Пусть нам требуется найти минимальный элемент в неупорядоченном массиве. Оказывается, что этот алгоритм также имеет линейную сложность, и для поиска минимального (максимального) элемента в неупорядоченном массиве потребуется $n - 1$ сравнений. Приведем этот алгоритм в текстовой форме и в виде блок-схемы.

Алгоритм поиска минимального элемента в неупорядоченном массиве



1. $min = a_1$.
2. Установить $i = 1$.
3. Если $i < n$, то перейти к шагу 4, иначе алгоритм окончен (минимальный элемент равен min).
4. Увеличить i на 1.
5. Если $a_i < min$, то присвоить min значение a_i.
6. Перейти к шагу 3. △

Еще более усложним задачу. Пусть нам требуется найти в неупорядоченном массиве максимальный и минимальный элементы одновременно. Можно скорректировать вышеприведенный алгоритм следующим образом:

1. $min = a_1$, $max = a_1$.
2. Установить $i = 1$.
3. Если $i < n$, то перейти к шагу 4, иначе алгоритм окончен (максимальный и минимальный элементы равны соответственно min и max).
4. Увеличить i на 1.
5. Если $a_i < min$, то min присвоить значение a_i.
6. Если $a_i > max$, то max присвоить значение a_i.
7. Перейти к шагу 3.

Сложность этого алгоритма равна $2(n-1)$. Зададимся вопросом, можно ли написать алгоритм с меньшей сложностью. Оказывается, можно написать алгоритм, сложность которого будет равна $3 \cdot \left\lceil \frac{n}{2} \right\rceil$.

Эффективный алгоритм поиска в неупорядоченном массиве максимального и минимального элементов одновременно

1. Разбить массив на пары (получим $\left\lceil \frac{n}{2} \right\rceil$ пар, при нечетном n — плюс еще один элемент).
2. Упорядочить по возрастанию каждую пару (при выполнении этого шага будет выполнено $\left\lceil \frac{n}{2} \right\rceil$ сравнений). Тогда

в массиве на всех нечетных местах будет стоять минимальное для данной пары число, а на всех четных местах — максимальное.

3. Найти минимальное число, осуществляя поиск только среди элементов, стоящих на нечетных местах. При этом, если у нас есть неполная пара, то в качестве начального значения переменной *min* взять значение элемента неполной пары (при выполнении этого шага будет выполнено $\left\lceil \frac{n}{2} \right\rceil$ сравнений).

4. Найдем максимальное число, осуществляя поиск только среди элементов, стоящих на четных местах. При этом если у нас есть неполная пара, то в качестве начального значения переменной *max* взять значение элемента неполной пары (при выполнении этого шага будет выполнено $\left\lceil \frac{n}{2} \right\rceil$ сравнений). △

Всего сравнений в данном алгоритме $3 \cdot \left\lceil \frac{n}{2} \right\rceil$.

4.6.2. Алгоритм бинарного поиска в упорядоченном массиве

Пусть нам требуется найти элемент в большом упорядоченном массиве информации, которая расположена в оперативной памяти компьютера. Для решения этой задачи разработаны эффективные алгоритмы, наиболее распространенным из них является *алгоритм бинарного (двоичного) поиска*, его иногда называют *логарифмическим поиском*, или *методом деления пополам (дихотомией)*.

Основная идея бинарного поиска довольно проста, детали же нетривиальны, и правильно работающий алгоритм удается написать далеко не с первого раза. Одна из наиболее популярных реализаций этого алгоритма использует два указателя (l и u), соответствующие нижней и верхней границам поиска. С помощью этого алгоритма ищется элемент k в упорядоченном по возрастанию массиве a, содержащем n элементов.

Алгоритм бинарного поиска в упорядоченном массиве

1. Начальная установка: $l = 1$, $u = n$.
2. Если $u < l$, то алгоритм окончен неудачно. В противном случае найти середину интервала $[l; u]$. В этот момент мы знаем, что если k есть в массиве, то выполняются неравенства $a_l \le k \le a_u$. Установить $i = [(l + u)/2]$. Теперь i указывает примерно в середину рассматриваемой части массива.
3. Если $k < a_i$, то перейти к шагу 4, если $k > a_i$, то перейти к шагу 5, если $k = a_i$, алгоритм окончен удачно.
4. Установить $u = i - 1$ и перейти к шагу 2.
5. Установить $l = i + 1$ и перейти к шагу 2. △

Шаг 3 алгоритма бинарного поиска выполняется порядка $\log_2 n$ раз, т. е. данный алгоритм имеет логарифмическую сложность по числу сравнений.

Вопросы и задания

1. Приведите примеры задач, в которых необходимо выполнять поиск информации в большом объеме данных.
2. При реализации поиска в неупорядоченном массиве применяют следующий прием. Массив увеличивают на один элемент, присваивая этому элементу значение искомого элемента. Какой выигрыш при этом мы можем получить?
3. Для поиска элемента в упорядоченном массиве был применен алгоритм бинарного поиска. В худшем случае алгоритм 5 раз выполнял шаг 3 (осуществлял сравнения). Какое максимальное количество элементов может содержать такой массив?
4. Процедура двоичного поиска нашла искомый элемент в упорядоченном массиве за две итерации (шаг 3 выполнялся 2 раза). Сколько всего элементов может содержать массив, если найденный элемент стоит в массиве на 23-м месте?

§ 4.7. Анализ алгоритмов сортировки

Сортировка — один из наиболее распространенных процессов современной обработки данных. *Сортировкой* называется распределение элементов множества по группам в соответствии с определенными правилами. Например, сортировка элементов массива, в результате которой получается массив, каждый элемент которого,

начиная со второго, не больше стоящего от него слева, называется *сортировкой по невозрастанию*.

В данной главе мы будем рассматривать только так называемые *внутренние* сортировки. Алгоритмы внутренней сортировки применяются для переупорядочивания данных, которые полностью располагаются в оперативной (внутренней) памяти. В этом случае мы имеем так называемый *прямой* (произвольный) *доступ* к элементам массива. В отличие от внутренней сортировки, существует внешняя сортировка, алгоритмы такой сортировки используют память на внешних носителях (например, сортировка данных с последовательным доступом, хранящихся в файлах на жестком диске).

Способов сортировки очень много, их можно разбить на группы в зависимости от идеи, лежащей в их основе.

Рассмотрим и проанализируем несколько алгоритмов сортировки для решения следующей задачи. Дан одномерный массив целых чисел. Требуется отсортировать его так, чтобы все элементы были расположены в порядке неубывания: $a[i] \le a[i+1]$, $i = 1, 2, \ldots, n-1$.

4.7.1. Обменная сортировка методом «пузырька»

Рассматриваемый ниже алгоритм относится к обменным сортировкам. Свое название «сортировка методом «пузырька» он получил на основе следующей ассоциации: если мы будем сортировать этим алгоритмом массив по убыванию, то минимальный элемент «всплывает», а «тяжелые» элементы опускаются на одну позицию к началу массива при каждом шаге алгоритма.

Алгоритм обменной сортировки методом «пузырька»

Алгоритм начинается со сравнения 1-го и 2-го элементов массива. Если элементы расположены не по порядку, то они меняются местами. Этот процесс повторяется со 2-м и 3-м, 3-м и 4-м и т. д. элементами, пока пара ($(n-1)$-й и n-й элемент) не будет обработана. За один просмотр массива самый большой элемент встанет на старшее (n-е) место. Далее алгоритм повторяется, причем на p-м просмотре уже только первые $(n-p)$ элементов сравниваются со своими правыми соседями. Если при очередном просмотре перестановок не было или $p = n$, то алгоритм окончен. △

Для того чтобы оценить сложность этого алгоритма, мы должны подсчитать количество действий, выполняемых при сортировке массива размерностью n. При достаточно большом n мы можем пренебречь всеми действиями, кроме действий сравнения элементов между собой и перестановки двух элементов массива.

Договоримся обработку элементов неупорядоченной части массива во время одного просмотра называть *итерацией* алгоритма.

Подсчитаем количество операций сравнений и присваиваний (одна перестановка реализуется тремя операциями присваивания) для случая, когда на вход алгоритма подается уже отсортированный массив.

Пример 11. Дан массив {1 2 3 4 5}. К нему применен алгоритм сортировки «пузырьком». Сделано 4 сравнения, 0 перестановок. После первого же просмотра массива алгоритм закончил свою работу. □

Очевидно, что максимальное количество сравнений и перестановок будет выполнено в случае, когда на вход алгоритма подается обратно упорядоченный массив.

Пример 12. Дан массив {5 4 3 2 1}.

1-я итерация: {4 3 2 1 5} (4 сравнения, 4 перестановки).
2-я итерация: {3 2 1 4 5} (3 сравнения, 3 перестановки).
3-я итерация: {2 1 3 4 5} (2 сравнения, 2 перестановки).
4-я итерация: {1 2 3 4 5} (1 сравнение, 1 перестановка).
Алгоритм работу закончил. Было сделано 4 + 3 + 2 + 1 = = 10 сравнений и 10 перестановок. □

В общем случае для обменной сортировки методом «пузырька» верны следующие оценки:

Количество сравнений		Количество присваиваний	
минимальное	максимальное	минимальное	максимальное
$n-1$	$n-1+n-2+n-3+$ $+\ldots+1=n(n-1)/2$	0	$3(n-1+n-2+n-3+$ $+\ldots+1) = 3n(n-1)/2$

Алгоритм сортировки «пузырьком» имеет квадратичную сложность $O(n^2)$.

Сортировка методом «пузырька» легко запоминается. Но этот метод сортировки на практике в том виде, как было рассказано, не используется. Этому есть несколько

причин, в частности много раз приходится просматривать массив, и, как следствие, программа работает долго. На основе алгоритма сортировки методом «пузырька» можно построить много улучшенных модификаций, например, если на какой-либо итерации первые k пар не участвовали в обмене, то на следующей итерации просмотр можно начинать с $(k + 2)$-го элемента.

4.7.2. Сортировка выбором

Данная сортировка также относится к обменным. Приведем сначала алгоритм этой сортировки в текстовой форме.

Находится наибольший элемент в массиве из n элементов. Пусть его место имеет номер max. Он меняется местами с элементом, стоящим на n-м месте, при условии, что $n \neq max$. Из оставшихся неупорядоченными $(n - 1)$ первых элементов снова выделяется наибольший и меняется местами с элементом, стоящим на $(n - 1)$-м месте и т. д. Алгоритм заканчивает свою работу, когда элементы, стоящие на 1-м и 2-м местах в массиве, будут упорядочены (для этого понадобится $n - 1$ итерация алгоритма). Аналогично данный алгоритм можно применять и к наименьшим элементам.

Запишем этот алгоритм более подробно с разбивкой по шагам. При описании будем использовать следующие обозначения:

- $a[1..n]$ — исходный массив;
- r — количество элементов в неупорядоченной части массива.

Алгоритм сортировки выбором

1. Положить $r = n$.
2. Найти наибольший элемент в массиве $a[1..r]$. Его место обозначим через max.
3. Если $max \neq r$ и $a[max] \neq a[r]$, то поменять местами элементы $a[max]$ и $a[r]$.
4. Передвинуть границы упорядоченной и неупорядоченной частей массива: $r = r - 1$. Первые r элементов будут образовывать неупорядоченную часть массива. Последние $n - r$ элементов — упорядоченную по возрастанию часть массива.
5. Если $r = 1$, то алгоритм окончен, иначе перейти к шагу 2. △

Для оценки сложности алгоритма сортировки выбором, как и для алгоритма сортировки «пузырьком», подсчитаем количество операций сравнения и перестановок.

Пример 13. Вначале рассмотрим случай, когда на вход алгоритма подается уже отсортированный массив {1 2 3 4 5}.

1-я итерация: 4 сравнения, 0 перестановок.
2-я итерация: 3 сравнения, 0 перестановок.
3-я итерация: 2 сравнения, 0 перестановок.
4-я итерация: 1 сравнение, 0 перестановок.

Алгоритм окончен. Было сделано 4 + 3 + 2 + 1 = 10 сравнений и 0 перестановок. □

Пример 14. Рассмотрим случай, когда на вход алгоритма подается обратно упорядоченный массив {5 4 3 2 1}.

После 1-й итерации получим: {1 4 3 2 5} (4 сравнения, 1 перестановка).
После 2-й итерации получим: {1 2 3 4 5} (3 сравнения, 1 перестановка).
После 3-й итерации получим: {1 2 3 4 5} (2 сравнения, 0 перестановок).
После 4-й итерации получим: {1 2 3 4 5} (1 сравнение, 0 перестановок).

Алгоритм окончен. Было сделано 4 + 3 + 2 + 1 = 10 сравнений и 2 перестановки. □

Максимальное количество перестановок на некотором массиве длины N будет равно $N - 1$, при этом число сравнений всегда будет одним и тем же.

Пример 15. Пусть дан массив {3 4 5 1 2}.

После 1-й итерации получим {3 4 2 1 5} (4 сравнения, 1 перестановка).
После 2-й итерации получим {3 1 2 4 5} (3 сравнения, 1 перестановка).
После 3-й итерации получим {2 1 3 4 5} (2 сравнения, 1 перестановка).
После 4-й итерации получим {1 2 3 4 5} (1 сравнение, 1 перестановка).

Алгоритм окончен. Было сделано 4 + 3 + 2 + 1 = 10 сравнений и 4 перестановки. □

В общем случае для обменной сортировки метод выбора верны следующие оценки:

Количество сравнений	Количество присваиваний	
в любом случае	минимальное	максимальное
$n-1 + n-2 + n-3 + \ldots + 1 = n(n-1)/2$	0	$3(n-1)$

Алгоритм сортировки выбором имеет квадратичную сложность $O(n^2)$ относительно операций сравнения и линейную сложность $O(n)$ относительно операций перестановки.

Получив оценку сложности, мы можем сделать вывод, что данный алгоритм целесообразно применять, если операция обмена над элементами массива трудоемка, например если элементом массива является запись с большим числом полей.

4.7.3. Сортировка вставками

Сортировка выбором и сортировка «пузырьком» относятся к обменным сортировкам с *убывающим шагом*. Действительно, после выполнения каждой итерации алгоритма количество элементов в неотсортированной части уменьшается на единицу. Сортировка вставками построена на ином принципе.

Алгоритм сортировки вставками

Вначале упорядочиваются два первых элемента массива. Они образуют начальное упорядоченное множество S. Далее на каждом шаге берется следующий по порядку элемент и вставляется в уже упорядоченное множество S так, чтобы слева от него все элементы были не больше, а справа — не меньше обрабатываемого. Место для вставки текущего элемента в упорядоченное множество S ищется методом деления пополам. Алгоритм сортировки заканчивает свою работу, когда элемент, первоначально стоящий на n-м месте, будет вставлен на соответствующее ему место. (Именно таким образом игроки обычно упорядочивают свои карты.) △

Для алгоритма сортировки вставками верны следующие оценки:

Количество сравнений (при реализации бинарного поиска)	Количество присваиваний	
в любом случае	мини-мальное	максимальное
$1 + 2 + [\log_2 3] + 1 + [\log_2 4] + + 1 + \ldots + [\log_2(n-1)] + 1 \approx \approx n - 1 + \log_2(n-1)! \approx n\log_2 n$	0	$3 + 4 + 5 + \ldots + n + 1 = = (n + 4)(n - 1)/2$

Алгоритм сортировки вставками имеет квадратичную сложность $O(n^2)$ по числу присваиваний и сложность $O(n\log_2 n)$ по числу сравнений.

Все рассмотренные выше алгоритмы сортировки имеют квадратичную сложность, по крайней мере по одному из параметров.

Вопрос: *Можно ли и до какого предела улучшать алгоритм решения задачи сортировки?*

Ответ. Более эффективные алгоритмы сортировки существуют. Сложность подобных универсальных алгоритмов составляет $O(n\log_2 n)$ по каждому из параметров. □

Для решения одной и той же задачи возможны самые разные подходы, анализ алгоритмов сортировки вскрывает сущность понятия сложности алгоритмов, а сравнение разных алгоритмов сортировки дает богатый материал для поиска путей совершенствования решений этой задачи. Почти все алгоритмы сортировки представляют несомненный практический интерес. Даже, казалось бы, не слишком эффективные прямые методы, и те в ряде случаев могут быть с успехом применены. При этом, к сожалению (или к счастью!), рекомендации на все случаи жизни дать нельзя. Чтобы выбрать самый подходящий для решаемой задачи алгоритм, необходимы знания, интуиция, творчество и опыт.

4.7.4. Сортировка слиянием

Рассмотренные алгоритмы сортировок вставками, выбором, «пузырьком» являются примерами алгоритма, действующего по шагам: в отсортированную часть добавляются новые элементы один за другим.

Рассмотрим алгоритм, основанный на другом подходе. Он основан на использовании метода «разделяй и властвуй».

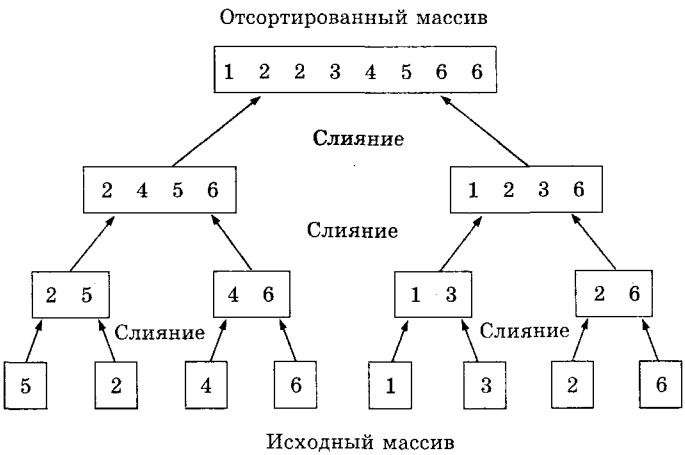
Многие алгоритмы по своей природе рекурсивны: решая некоторую задачу, они вызывают самих себя для решения ее подзадач. Идея метода «разделяй и властвуй» состоит как раз в этом. Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова. Наконец, их решения комбинируются и получается решение исходной задачи.

Для задачи сортировки эти три этапа выглядят следующим образом.

1. Сначала мы разбиваем массив на две половины.
2. Затем сортируем каждую из половин отдельно.
3. После этого соединяем два упорядоченных массива половинного размера в один.

Рекурсивное разбиение задачи на меньшие подзадачи происходит до тех пор, пока размер массива не дойдет до единицы: любой массив длины 1 можно считать упорядоченным.

Процесс сортировки слиянием показан на рисунке.



Вопрос. *Как оценить время работы рекурсивного алгоритма?*

Ответ. Время работы алгоритма складывается из времени, затрачиваемого на рекурсивные вызовы, которое можно выразить через некое рекуррентное соотношение. Далее следует оценить время работы, исходя из полученного соотношения.

Предположим, что алгоритм каждый раз разбивает задачу размера n на a подзадач, каждая из которых имеет размерность в b раз меньшую. Будем считать, что разбиение требует времени $D(n)$, а соединение полученных решений — $C(n)$. Тогда для времени работы алгоритма $T(n)$ получаем следующее рекуррентное соотношение (в худшем случае): $T(n) = aT(n/b) + D(n) + C(n)$. Это соотношение выполняется для достаточно больших n, когда задачу имеет смысл разбивать на подзадачи. Для маленьких n, когда такое разбиение или невозможно, или не нужно, применяется какой-нибудь прямой метод решения. Поскольку n ограничено (задача может состоять в сортировке очень большого массива, но он все равно конечен), время работы также не превосходит некоторой константы. □

Пример 16. Подсчитаем сложность алгоритма сортировки слиянием относительно операций сравнения. Для простоты будем считать, что размер массива есть степень двойки. Тогда на каждом шаге сортируемый массив делится пополам. Разбиение на части (вычисление границы сортируемого массива) требует времени $O(1)$. Слияние двух отсортированных частей в массив большей размерности n — времени $O(n)$. Тогда мы получаем следующее рекуррентное соотношение для вычисления сложности алгоритма сортировки слиянием:

$$T(n) = \begin{cases} O(1), & \text{если } n = 1, \\ 2T(n/2) + O(n), & \text{если } n > 1. \end{cases}$$

Это соотношение влечет $T(n) = O(n\log_2 n)$. Вывод этого равенства приведен в книге Т. Кормена и др. «Алгоритмы: построение и анализ».

Следовательно, для больших n сортировка слиянием эффективнее рассмотренных ранее алгоритмов сортировки, имеющих сложность $O(n^2)$. □

В заключение этого параграфа скажем несколько слов о пользе быстрых алгоритмов. Часто разница между плохим и хорошим алгоритмами более существенна, чем разница между быстрым и медленным компьютерами. Приведем пример, аналогичный описанному в той же книге Кормена.

Пример 17. Требуется отсортировать массив из миллиона чисел. Что быстрее — сортировать его прямым выбором на компьютере, выполняющем 100 млн операций в секунду, или слиянием на компьютере, выполняющем 1 млн операций в секунду? Для сортировки n чисел выбором требуется только $n^2/2$ операций сравнения. Алгоритм сортировки слиянием требует $n\log_2 n$ операций сравнения.

При сортировке 1 млн чисел выбором получаем:

$$\frac{1/2 \cdot (10^6)^2 \text{ операций}}{10^8 \text{ операций в секунду}} = 5000 \text{ секунд} \approx 83 \text{ минуты}.$$

При сортировке этого же массива чисел слиянием получаем:

$$\frac{10^6 \cdot \log_2 10^6 \text{ операций}}{10^6 \text{ операций в секунду}} \approx 20 \text{ секунд} \approx 1/3 \text{ минуты}. \square$$

Вывод: разработка эффективных алгоритмов не менее важна, чем разработка быстрой электроники.

Вопросы и задания

1. Какие из описанных выше обменных алгоритмов работают быстро на почти упорядоченном массиве?
2. Укажите, какой массив надо подать на вход алгоритма сортировки вставками ($N = 5$), чтобы количество выполненных присваиваний было максимальным.
3. На вход алгоритма вставками подается массив {3 4 5 1 2}. Распишите, как изменяется массив в процессе выполнения алгоритма, аналогично примерам 13–15.
4. Требуется упорядочить по весу в возрастающем порядке N непрозрачных банок с чаем, имея в своем распоряжении только чашечные весы без гирь. Напишите наиболее эффективный алгоритм решения этой задачи.
5. Тезис Чёрча говорит о том, что класс вычислимых функций совпадает с классом рекурсивных функций. Запишите один из обменных алгоритмов сортировки в виде рекурсивного алгоритма.
6. Придумайте эффективный алгоритм одновременного поиска двух самых маленьких или самых больших элементов в неупорядоченном массиве.

Заключение

Научные открытия, новые идеи не возникают сами по себе, они рождаются в ответ на спрос, неудовлетворенную потребность. Как только математики высказали предположение, что некоторые задачи, быть может, не имеют решения (что само по себе являлось определенным научным достижением), математические исследования в данной области вышли на новый рубеж: а можно ли строго доказать, что рассматриваемая задача не имеет алгоритмического решения? В данном случае получение ответа на этот вопрос очень значимо: можно перестать тратить усилия и время на отыскание результата, который нельзя найти.

Задача доказательства алгоритмической неразрешимости привела к появлению другой задачи — построения формального определения алгоритма. Математики очень долго пользовались интуитивным понятием алгоритма, и оно их устраивало. Но для строгого доказательства несуществования алгоритма надо знать, несуществование чего мы доказываем. А. Тьюрингом и Э. Постом были предложены формальные определения алгоритмов в виде абстрактных вычислительных конструкций. Достаточно простые по своей сути, эти математические конструкции, названные в дальнейшем «машинами», оказались универсальными вычислителями, на которых можно было реализовать любой известный в математике алгоритм. Дальнейшие исследования в области теории алгоритмов привели к понятию вычислимой функции и вычислимости как таковой.

Более того, некоторые идеи из доказанной А. Тьюрингом в 1936 г. теоремы о существовании универсального вычислителя были использованы в дальнейшем (в 1940-х годах) при построении первого компьютера, а сам А. Тьюринг был одним из его разработчиков.

Мы проследили цепочку зарождения проблем и их решений, которые легли в основу целого раздела математики под названием «Теория алгоритмов», показали, по какому критерию из нескольких алгоритмов решения задачи следует выбирать лучший. Основные понятия теории алгоритмов достойны внимания как математиков, так и специалистов в области информатики. К таким понятиям следует отнести понятия алгоритма и вычислимой функции, понятие сложности алгоритма и ряд других.

Глава 5

Основы теории информации

Развитие математики вновь подтверждает, что деление науки на теоретическую часть и прикладную неправомерно. Наука едина, и можно говорить лишь о самой науке и ее возможных приложениях.

Н. Н. Боголюбов, М. А. Лаврентьев

Теория информации — сравнительно молодой раздел математики — сформировалась как наука во второй половине XX века. Толчок к зарождению этой науки дали попытки нахождения оптимальных решений практических технических задач, в области кодирования и передачи информации. Пионерами теории информации по праву считают Клода Шеннона (США), А. Н. Колмогорова (СССР), Р. Хартли (США).

§ 5.1. Понятие информации. Количество информации. Единицы измерения информации

В данной главе мы возвращаемся к понятию информации, уже неоднократно встречавшемуся ранее.

Информация — одно из фундаментальных понятий современной науки наряду с такими понятиями, как «вещество» и «энергия». Строгое определение этому термину дать невозможно. С точки зрения человека информация — это *сведения*, обладающие такими характеристиками, как понятность, достоверность, новизна и актуальность (ценность).

Например, телефонный справочник обычно хранит информацию об абонентах городской телефонной сети. Однако если мы возьмем телефонный справочник г. Москвы за 1965 год, то сведения, в нем содержащиеся, будут по крайней мере не достоверны (за прошедшие с момента выпуска справочника годы изменились номера телефонов многих абонентов, многие учреждения оказались упраздненными, магазины — закрытыми и т. д.). Следовательно, для абонента московской городской телефонной сети XXI века информации такой справочник почти не содержит.

В разных науках существуют различные формальные подходы к определению понятия информации. В информатике нас прежде всего интересуют два определения: сформировавшееся в математической теории информации и применяемое в *computer science* (данную отрасль научного знания обычно переводят на русский язык как *информатика*, однако последняя является более широким понятием, так как включает в себя и кибернетику, и моделирование, и ту же теорию информации, и многое другое).

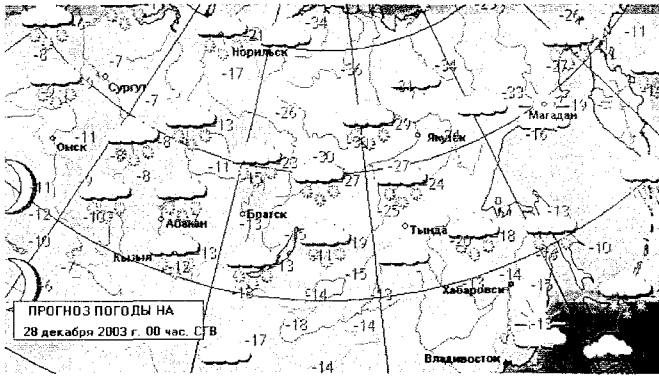
Определение 1. Согласно американскому ученому и инженеру Клоду Шеннону, *информация* — это снятая *неопределенность*.

Шеннон впервые ввел такую трактовку в теории связи. Позднее она нашла применение во всех областях науки, где играет роль передача информации в самом широком смысле этого слова, в частности в математической теории информации, большую роль в становлении которой сыграл и великий российский ученый-математик А. Н. Колмогоров.

Согласно Шеннону, информативность сообщения характеризуется содержащейся в нем *полезной информацией*, т. е. той частью сообщения, которая снимает полностью или уменьшает существовавшую до ее получения *неопределенность* какой-либо ситуации.

Определение 2. Величина *неопределенности* некоторого события — это количество возможных результатов (исходов) данного события.

Такой подход к определению информации называют *содержательным*. Так, например, неопределенность погоды на завтра обычно заключается в диапазоне температуры воздуха и возможности выпадения осадков. Причем в разное время года диапазон возможных исходов данного события различен и колеблется в районе 16 ожидаемых результатов для значения дневной температуры и трех-четырех — для осадков (без осадков, кратковременные дожди, ливень с градом и т. п.). Позднее мы покажем, какое количество информации несет достоверный прогноз погоды.



Вопрос. *Зависит ли количество информации от того, кто и как фиксирует неопределенность соответствующей ситуации?*

Ответ. У разных людей неопределенность знания о некотором предмете может различаться. Так, неопределенность знания о погоде в Москве у рядового американца и российского синоптика может различаться кардинально. Поэтому содержательный подход к трактовке понятия информации часто является *субъективным*. Если же число возможных исходов события не зависит от суждений различных людей, то получаемая информация о наступлении одного из возможных исходов является *объективной*. Примером такой информации является сообщение о результате падения подброшенной монеты или игрального кубика. □

Вместе с тем, всякое сообщение (любые данные) можно закодировать с помощью конечной последовательности символов некоторого алфавита. С точки зрения информатики носителями информации являются любые последовательности символов, которые *хранятся*, *передаются* и *обрабатываются* с помощью компьютера.

Определение 3. Согласно Колмогорову, *количество информации*, содержащейся в последовательности символов, определяется минимально возможным количеством двоичных знаков, необходимых для кодирования этой последовательности безотносительно к содержанию представленного ею сообщения.

А. Н. Колмогоров (1903–1987)

Данный подход к определению информации называют *алфавитным*. При этом для кодирования наиболее часто используется двоичный алфавит, состоящий из нуля и единицы, это так называемое двоичное кодирование информации. Смысл сообщения при этом может быть учтен лишь на этапе выбора исходного алфавита для его записи либо не учтен вообще. В частности, сообщения, записанные на естественном языке с помощью соответствующего алфавита (например, сейчас вы читаете сообщение на русском языке, записанное с помощью русского же алфа-

вита), кодируются без учета их смыслового содержания. Такой подход является *объективным*.

На первый взгляд, определения 1 и 3 кажутся существенно различными. Тем не менее ниже будет показано, что они хорошо согласуются друг с другом.

Хотя информацию нельзя строго определить, ее можно измерить. Вернее, можно задать числом количество информации, подобно тому, как можно задать числом расстояние, время, массу, температуру и т. п. Чтобы стандартизировать измерение количества информации, договорились за единицу измерения брать *бит* (от английского ***bi**nary **dig**it*).

Определение 4. При алфавитном подходе *один бит* — это количество информации, которое можно передать в сообщении, состоящем из одного двоичного знака (0 или 1). С точки зрения содержательного подхода один *бит* — это количество информации, уменьшающее неопределенность знания о предмете в два раза.

Содержательный подход отвечает на вопрос, какое количество «новой» информации мы получаем из сообщения. Так, сообщение о том, что подброшенная монета упала «решкой» вверх, несет в себе один бит информации. В самом деле, неопределенность знания о результате падения монеты заключалась в двух возможных исходах. После того как конкретный исход стал известен, неопределенность уменьшилась в два раза, что и соответствует одному биту информации. Но ведь результат падения монеты как раз можно закодировать одним из двух символов (например, используя все те же 0 и 1). Таким образом, несмотря на различие в трактовках, понятие бита является согласованным для различных подходов.

На практике чаще используется более крупная единица — *байт*, равная 8 битам. Так, один байт информации можно передать с помощью одного символа кодировки ASCII (см. § 2.3). Используются также следующие производные единицы измерения информации:

1 килобайт (1 Кб) = 2^{10} байт = 1024 байт;
1 мегабайт (1 Мб) = 2^{20} байт = 1024 Кб;
1 гигабайт (1 Гб) = 2^{30} байт = 1024 Мб;
1 терабайт (1 Тб) = 2^{40} байт = 1024 Гб.

Для вопросов хранения и передачи информации более важным является не количество информации, а ее объем.

Определение 5. *Информационным объемом* сообщения называется количество двоичных символов, которое используется для кодирования этого сообщения.

В отличие от определения *количества информации* (определение 3), в данном случае не требуется, чтобы число двоичных символов было минимально возможным. Размер текстовых файлов, в которых сообщение на английском языке записано в ASCII-кодировке, характеризует именно информационный объем соответствующего текста, а не количество информации в нем, даже в смысле алфавитного подхода. При оптимальном кодировании, речь о котором пойдет ниже, понятия количества информации и информационного объема совпадают.

Вопросы и задания

1. Расскажите, как вы понимаете термин «информация». Что общего и каковы различия между бытовом понятием этого термина и его научными трактовками?
2. Приведите пример, когда алфавитный подход к трактовке понятия информации оказывается субъективным.
3. При игре в кости используются два игральных кубика, грани которых помечены числами от одного до шести. В чем заключается неопределенность знания о бросании одного кубика? А двух кубиков одновременно?
4. Сколько гигабайт содержится в 2^{18} килобайтах? Сколько мегабайт содержится в 2^{20} килобитах?
5. Вспомните различные жизненные ситуации, при которых мы получаем ровно один бит информации.

§ 5.2. Формула Хартли определения количества информации

Процесс измерения количества информации можно объяснить на примере известной игры в угадывание. Согласно правилам этой игры, один из участников должен отгадать предмет (или одно из возможных состояний некоторого предмета), который задумал кто-то другой. При этом допускается только два вида ответов: «да» и

«нет». В данном случае ответ на один вопрос несет один бит информации, так как из двух возможных исходов выбирается один.

Задача 1. *Предположим, что в классе находятся 32 ученика, и учитель решил спросить одного из них. Какое минимально возможное количество вопросов нам надо задать учителю, чтобы наверняка определить, кого именно он решил спросить?*

Решение. Способ 1. Если в классе 4 ряда парт, и они заполнены равномерно, то сначала зададим учителю вопрос: «Сидит ли задуманный ученик на парте в первом или втором ряду?» Получив ответ «да» или «нет», мы сократим количество «подозреваемых» до 16. Вторым вопросом можно определить конкретный ряд, на котором сидит искомый школьник, сократив выбор до 8 человек. Далее будем поступать аналогично. После каждого ответа число «подозреваемых» сокращается вдвое. После четвертого вопроса выбор останется сделать из двух учеников. Это можно осуществить, задав пятый вопрос. Данный способ поиска, как уже говорилось в предыдущей главе, носит название метода деления пополам (дихотомии).

Записав полученные ответы и заменив все ответы «да» единицами, а «нет» — нулями, мы получим сообщение в виде последовательности из пяти двоичных цифр.

Способ 2. Отгадывать школьника можно было и по-другому. Присвоим каждому школьнику номер от 0 до 31. Запишем номера в двоичной системе счисления. Самые длинные номера состоят из пяти двоичных цифр. Дополним остальные номера до пяти цифр слева нулями. Далее вопросы учителю можно задавать так: «Верно ли, что первая слева цифра в номере задуманного ученика равна единице?», «Верно ли, что вторая цифра в номере задуманного ученика равна единице?» и т. д. Предположим, что ответы на вопросы были: «нет», «да», «нет», «нет», «да». Тогда мы сразу определим, что номер загаданного ученика — $01001_2 = 9$.

В обеих процедурах определения загаданного ученика длина получаемого сообщения равна пяти двоичным символам, т. е. в результате угадывания мы получаем 5 бит информации. □

Мы увидели, что для отгадывания задуманного ученика из 32 школьников *достаточно* задать 5 вопросов указанного выше вида. Но если задавать вопросы не лучшим образом, может случиться так, что необходимо не 5, а, скажем, 7 вопросов. Казалось бы, что так как каждый вопрос подразумевает один из двух возможных ответов, то, получив ответы на первые 5 вопросов, мы всегда получим 5 бит информации. Однако это не всегда справедливо. Как же происходит «недостача» информации?

Дело в том, что при неразумном выборе вопросов может получиться так, что какие-либо два вопроса, ответ на каждый из которых несет один бит информации, в сумме содержат меньше двух бит информации. Это возможно, если ответ на первый вопрос полностью или частично содержит ответ на второй. Например, если сначала спросить, не равна ли первая цифра в номере ученика нулю, а потом — не равна ли она единице, то второй ответ не добавляет ровно никакой информации, и общее количество информации в двух ответах равно одному биту, а не двум. Отсюда напрашивается следующий вывод.

Количество информации, необходимое для угадывания одного из исходов, можно измерить числом вопросов, которые требуется задать *при наиболее рациональной тактике задавания вопросов.*

В результате приходим к следующему определению.

Определение 6. Для того чтобы измерить количество информации в сообщении, надо закодировать сообщение в виде последовательности нулей и единиц *наиболее рациональным способом*, позволяющим получить *самую короткую последовательность*. Длина полученной последовательности нулей и единиц и является *мерой количества информации* в битах.

Вопрос. *Пусть для кодирования всех возможных исходов некоторого события были использованы двоичные последовательности длины k. Означает ли это, что количество информации в сообщении о наступлении одного из исходов данного события равно k?*

Ответ. Вернемся к задаче 1 в случае, когда надо выбирать задуманного ученика уже среди 24 человек. В этом случае нам понадобится не меньше 4 и не больше 5 вопро-

сов, если действовать методом деления пополам. После третьего вопроса у нас останется трое «подозреваемых». Их можно разделить на группу из одного и группу из двух учеников. Тогда после четвертого вопроса мы либо сразу найдем нужного школьника, либо придется задавать пятый вопрос.

Попробуем действовать вторым способом. Закодируем номера 24 школьников двоичными номерами от 00000_2 до 10111_2. Содержит ли закодированный номер 5 бит информации? Нет, не совсем так. Длина кодовой последовательности здесь та же, что и раньше, но число различных номеров меньше. Поэтому, например, если мы узнаем, что старшая цифра номера равна 1, то вопрос о второй слева цифре задавать не придется (объясните почему), и мы сумеем определить номер искомого ученика, задав 4 вопроса, а не 5. Значит, количество информации, требуемой, чтобы выбрать задуманного ученика из 24 человек, больше 4 бит и меньше 5 бит. □

А какое количество информации требуется, чтобы угадать один задуманный предмет из N различных предметов? Пусть $N = 2^k$. Тогда, действуя методом деления пополам, мы можем определить задуманный предмет с помощью k вопросов. При этом количество получаемой информации H составляет k бит. То есть в данном случае $H = k = \log_2 N$. Ниже мы покажем, что эта формула, носящая название *формулы Хартли*, справедлива для любого натурального N.

Лемма 1. Число различных двоичных слов длины k равно 2^k.

Доказательство. Докажем данное утверждение индукцией по длине последовательности k. Для $k = 1$ утверждение очевидно. Пусть уже доказано, что число двоичных слов длины k равно 2^k. Все двоичные слова длины $k + 1$ делятся на два типа: начинающиеся с нуля и начинающиеся с единицы. Выпишем все слова длины k, в силу предположения индукции их 2^k. Припишем к каждому из них слева ноль. Ниже выпишем все слова длины k снова и припишем к ним слева единицу. Так мы записали все слова длины $k + 1$. Таким образом, число последовательностей длины $k + 1$ равно $2^k + 2^k = 2^{k+1}$.

Лемма доказана.

Определение 7. *Двоичным кодированием* множества N называется отображение, ставящее в соответствие каждому элементу множества N его двоичный код: последовательность нулей и единиц. Кодирование называется *однозначным*, если коды различных элементов множества N различны.

Лемма 2. Множество N допускает однозначное двоичное кодирование с длинами кодов, не превосходящими k, в том и только в том случае, когда число элементов множества N не превосходит 2^k.

Доказательство. Одно множество можно однозначно отобразить в другое множество, только если число элементов первого множества не превышает числа элементов второго множества. Так как по лемме 1 число двоичных слов длины k равно 2^k, то доказываемое утверждение справедливо.
Лемма доказана.

Согласно леммам 1 и 2, длина кода при двоичном кодировании одного символа из алфавита *мощности* $N = 2^k$ (то есть алфавита, состоящего ровно из N различных символов) равна k. Это позволяет давать эффективные оценки на минимально необходимый объем памяти компьютера для запоминания различного рода данных. Так, кодирование сообщений на английском языке можно осуществлять с помощью алфавита, состоящего из $32 = 2^5$ различных символов (к 26 латинским буквам необходимо добавить символ пробела, точку, запятую и еще три символа по желанию). Один символ при равномерном двоичном кодировании (одинаковой длине двоичного слова для каждого символа алфавита) тогда будет занимать 5 бит памяти, а не 8, как при ASCII-кодировании текстовой информации вообще.

Задача 2. *Рассмотрим следующую проблему кодирования результатов голосования. Пусть имеются три варианта голосования: «за», «против», «воздержался». Требуется закодировать результаты голосования, содержащиеся в N бюллетенях.*

Решение. Результаты можно было бы кодировать с помощью двухбитовых слов, например 11, 00, 01. При этом комбинация 10 остается неиспользованной. При таком ко-

дировании N результатов голосования займут объем $2N$ бит памяти. Для того чтобы улучшить этот результат, будем кодировать блоки из трех бюллетеней. Тогда число вариантов голосования для тройки бюллетеней равняется $3 \cdot 3 \cdot 3 = 27 < 2^5$, т. е. в данном случае возможно пятибитовое кодирование. Так как число троек равно $N/3$, а каждая из них займет 5 бит памяти, то для хранения понадобится уже $5N/3$ бит памяти.

Но и этот результат можно улучшить, если составить блоки из 5 бюллетеней. В этом случае число исходов голосования для блока составляет $3^5 = 243 < 256 = 2^8$ и его можно записать в один байт. Таким образом, на бюллетень придется $8/5$ бита памяти.

Рассмотрим теперь кодирование бюллетеней, объединенных в блоки по m штук. Пусть натуральное k таково, что $2^k < 3^m < 2^{k+1}$. Тогда мы можем кодировать результаты голосования, потратив на бюллетень не более $(k+1)/m$ бит, но не менее k/m бит (последнее — по лемме 2). Очевидно, что блочное кодирование с ростом m дает результаты, сколь угодно близкие к оптимальному кодированию. □

Вывод, к которому мы пришли при решении задачи 2, позволяет нам вывести следующую формулу, называемую формулой Хартли.

Р. Хартли (1888–1970); область научных интересов: радиоэлектроника, теория информации. *Первым ввел понятие информации как переменной величины и попытался ввести меру количества информации.*

Формула Хартли

Количество информации, которое вмещает один символ N*-элементного алфавита, равно* $\log_2 N$.

По-другому это же утверждение можно сформулировать так. *Количество информации, полученное при выборе одного предмета из* N *равнозначных предметов, равно* $\log_2 N$. То есть именно такое количество информации необходимо для устранения неопределенности из N равнозначных вариантов.

Доказательство. Для алфавита из N различных символов можно составить N^m всех возможных слов длины m (доказывается аналогично лемме 1). Пусть k таково, что

$$2^k < N^m \leq 2^{k+1}. \qquad (5.1)$$

То есть количество информации, содержащееся в одном символе этого алфавита, заключено между k/m и $(k+1)/m$. Логарифмируя неравенства (5.1), получаем:

$$k < m\log_2 N \leq k + 1. \qquad (5.2)$$

Разделив неравенства (5.2) на m, делаем вывод, что разность между количеством информации, которое несет один символ N-элементного алфавита, и $\log_2 N$ не превосходит $1/m$ для любого m. Значит, при оптимальном двоичном кодировании такого символа (которое при росте m достижимо с какой угодно точностью) средняя длина слова составит $\log_2 N$.

Для доказательства утверждения во второй формулировке будем отгадывать один предмет из N возможных не один раз, а m, или отгадывать набор из m предметов, каждый из которых принадлежит множеству N различных, но равнозначных предметов. Тогда аналогично решению задачи 2 среднее число вопросов, необходимое для отгадывания одного предмета из набора, будет отличаться от $\log_2 N$ не более чем на $1/m$.
Формула Хартли доказана.

Вопросы и задания

1. В библиотеке 16 стеллажей, в каждом стеллаже 8 полок. Какое количество информации несет сообщение о том, что нужная книга находится на четвертой полке?
2. Была получена телеграмма: «Встречайте вагон 7 поезд № 32». Какое количество информации получил адресат, если известно, что в этот город приходят 4 поезда, а в каждом поезде в среднем 16 вагонов?
3. Сколько информации получит ученик, если в 10-00 увидит сообщение о том, что классный час состоится в 14 часов, при условии, что в этот день всегда бывает классный час? Ответ обоснуйте.
4. Почему множество ASCII-символов образует алфавит, состоящий именно из 256 символов? Сколько вопросов надо задать, чтобы отгадать один из символов этого алфавита?

5. В классе четыре ряда парт по четыре парты в каждом ряду. Каждая парта имеет два места. Все места на партах заполнены учениками. Учитель задумал одного из них. Какое количество информации мы получим, если зададим два следующих вопроса и получим на них положительный ответ?
   Вопросы:
   Сидит ли задуманный ученик на первых двух рядах?
   Сидит ли задуманный ученик на первой или второй парте?
6. Верно ли, что для любого m при организации в задаче 2 m-блочного кодирования результатов голосования $(m + 1)$-блочное кодирование всегда ближе к оптимальному, чем m-блочное?
7. Докажите, что для алфавита из N различных символов можно составить N^m всех возможных слов длины m.

§ 5.3. Применение формулы Хартли

Формула Хартли уточняет данное ранее определение количества информации и дает нам трактовку нецелого количества информации. Заметим, что для любого алфавита можно выбрать свою единицу измерения информации, такую что количество информации, содержащееся в любом сообщении, записанном с помощью этого же алфавита, будет выражаться целым числом.

Пример 1. В задаче 2 из предыдущего параграфа шла речь о кодировании результатов голосования. При этом каждый бюллетень содержал в себе один из трех возможных исходов голосования. Если за единицу измерения количества информации принять количество информации, которое можно передать одним символом трехсимвольного алфавита (например, «+», «−», «0»), что соответствует уменьшению неопределенности в три раза, то каждый бюллетень будет содержать ровно одну такую единицу количества информации. □

В формуле Хартли логарифм можно брать по любому основанию, это равносильно выбору единицы измерения количества информации.

Пример 2. Введенная в примере 1 единица измерения количества информации удобна и при анализе результатов взвешивания на двухчашечных весах. Как известно, каждое такое взвешивание может привести к одному из трех результатов: тяжелее оказывается груз на левой чаше весов, тяжелее груз на правой чаше или весы находятся в равновесии. В данном случае результатом одного взвешивания может являться количество информации, большее, чем бит — $\log_2 3$. Пусть у нас имеется N монет, одна из которых фальшивая (легче остальных). k взвешиваний дают $k\log_2 3$ бит информации и не могут снять неопределенность, бóльшую, чем $\log_2 3^k$, поэтому для определения фальшивой монеты потребуется не менее $\log_2 N/\log_2 3 = k$ взвешиваний. Здесь k — количество информации в единицах, введенных в примере 1. □

Так, например, при поиске одной более легкой монеты среди 27 монет нам потребуется не менее $\log_2 27/\log_2 3 = 3$ взвешиваний.

Известно, что эта задача как раз и разрешима ровно за три взвешивания.

Пример 3. Задача поиска фальшивой монеты становится более сложной, если заранее неизвестно, легче она или тяжелее остальных монет. В этом случае число различных исходов изначально равно $2N$ (каждая монета может оказаться фальшивой и быть как легче остальных, так и тяжелее). Значит, для решения задачи понадобится не менее $\log_2 2N/\log_2 3$ взвешиваний. Таким образом, за три взвешивания мы теперь сможем исследовать не более 13 монет ($2N < 27$). □

Поиск алгоритма решения такой задачи сложен и для меньшего числа монет.

Задание. *Попробуйте решить эту задачу самостоятельно для 12 монет, и вы убедитесь, что это совсем не просто, а порой вообще кажется невозможным.*

Решение. Для решения задачи нужно использовать такую операцию, как перекладывание монеты с одной чаши весов на другую. Введем следующие обозначения. Монету,

обозначенную знаком «+», во время текущего взвешивания следует положить на весы, причем если она на весах уже была, то на ту же самую чашу, на которой эта монета находилась во время своего предыдущего взвешивания. Монету, обозначенную знаком «−», следует переложить на противоположную чашу весов по отношению к той, на которой она находилась. Заметим, что если монета на весах еще не была, то знак «−» к ней применен быть не может. Наконец, монеты, обозначенные знаком «0», в очередном взвешивании не участвуют.

Составим таблицу путем выписывания всех возможных вариантов расположения символов «+», «−» и «0» в столбцах, соответствующих монетам. Недопустимые варианты: знак «−» расположен выше знака «+». Каждый столбец описывает местонахождение соответствующей ему монеты во время каждого из трех взвешиваний.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	
+	+	+	+	+	+	+	+	+	0	0	0	0	0	первое взвешивание
+	+	+	−	−	−	0	0	0	+	+	+	0	0	второе взвешивание
+	−	0	+	−	0	+	−	0	+	−	0	+	−	третье взвешивание

Так как различных вариантов получилось 14, а мы решаем задачу для 12 монет, то из полученной таблицы вычеркнем два столбца так, чтобы в каждой из трех строк количество ненулевых элементов оказалось четным (ведь бессмысленно во время одного взвешивания класть на чаши весов разное число монет). Это могут быть, например, 4-й и 14-й столбцы. Теперь будем взвешивать 12 монет так, как это записано в оставшихся 12 столбцах. То есть в первом взвешивании будут участвовать 8 произвольных монет. Во втором 3 монеты следует с весов убрать, 2 — переложить на противоположные по отношению к первому взвешиванию чаши весов и 3 монеты положить на весы впервые (на свободные места так, чтобы на каждой из чаш вновь оказалось по 4 монеты). Согласно схеме проведем и третье взвешивание, опять располагая на каждой чаше весов по 4 монеты.

Результат каждого взвешивания в отдельности никак не анализируется, а просто записывается. При этом равновесие на весах всегда кодируется нулем, впервые возникшее неравновесное состояние — знаком «плюс». Если при следующем взвешивании весы отклонятся от равновесия в ту же самую сторону, то результат такого взвешивания также кодируется плюсом, а если в другую

сторону, то минусом. Например, следующие результаты взвешиваний «=<<» и «=>>» кодируются как «0++», а результаты «<=>» и «>=<» — как «+0−». Так как мы не знаем, легче или тяжелее остальных монет окажется фальшивая монета, то нам важно, как изменялось состояние весов от взвешивания к взвешиванию, а не то, какая именно чаша оказывалась тяжелее, а какая легче. Поэтому два на первый взгляд различных результата трех взвешиваний в этом случае кодируются одинаково.

После подобной записи результатов взвешиваний фальшивая монета уже фактически определена. Ею оказывается та, которой соответствует такой же столбец в таблице, как и закодированный нами результат трех взвешиваний. Для первого примера это монета, которая участвовала во взвешиваниях по схеме, указанной в 10-м столбце таблицы, а для второго — в 8-м. В самом деле, состояние весов в нашей задаче меняется в зависимости от того, где оказывается фальшивая монета во время каждого из взвешиваний. Поэтому монета, «поведение» которой согласуется с записанным результатом взвешиваний, такой результат и определяет.

Анализ таблицы показывает, что аналогично эту задачу можно решить и для 13 монет. Для этого следует исключить из рассмотрения любой не содержащий нулей столбец, например, все тот же четвертый. В остальном все действия остаются неизменными. Подход к решению задачи не изменится для любого значения N, но заметим, что для $N = 2$ задача неразрешима. □

В главе 4 приведено несколько различных алгоритмов сортировки массивов, количество операций сравнения в которых для худшего случая различно. Некоторые из них являются более эффективными по числу производимых сравнений.

Вопрос. Какова сложность алгоритмов сортировки, оптимальных по числу сравнений?

Ответ. Ответ и на этот вопрос может дать теория информации.

Отсортировать произвольный массив с точки зрения теории информации — это значит найти одну из $N! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot N$ перестановок всех N элементов данного массива. Сравнения элементов между собой в данном

случае соответствуют задаваемым вопросам с возможными вариантами ответа «да» или «нет». Тогда по формуле Хартли для сортировки элементов в худшем случае потребуется получить не менее $\log_2 N!$ бит информации, т. е. произвести не менее $\log_2 N!$ операций сравнения элементов между собой. Для оценки значения $N!$ при больших N в математике применяют формулу Стирлинга:

$$N! \approx \sqrt{2\pi N} N^N e^{-N}.$$

То есть оптимальная сортировка в худшем случае будет производить около $(N + 0{,}5)\log_2 N + 0{,}5\log_2 2\pi - N\log_2 e$ сравнений. Таким образом, эффективные по числу сравнений сортировки (выполняющие порядка $N\log_2 N$ сравнений) действительно асимптотически оптимальны. Для каждого значения N в отдельности можно построить алгоритм сортировки, количество сравнений в котором будет практически совпадать с теоретической оценкой. Так, массив, состоящий из 5 элементов, действительно можно упорядочить за не более чем 7 сравнений ($\log_2 5! = \log_2 120 \approx 7$). Любому же универсальному алгоритму сортировки для решения этой задачи в некоторых случаях может потребоваться сделать 8 сравнений. Подобные оптимальные алгоритмы можно построить для любого N, не превосходящего 11. Для $N = 12$ наилучший возможный алгоритм сортировки в худшем случае выполняет 30 сравнений, а $[\log_2 12!] + 1 = 29$. Дальнейшие исследования данного вопроса затруднены в силу вычислительной трудоемкости построения оптимального алгоритма сортировки. □

Вопросы и задания

1. Объясните, как, используя формулу Хартли, можно сразу измерить любую, например графическую, информацию в байтах, а не в битах?
2. В анкете предлагаются следующие варианты ответа на вопрос о степени владения английским языком: «не владею», «читаю со словарем», «могу объясняться», «владею хорошо», «могу переводить синхронно». Какое количество информации несет ответ на данный пункт анкеты? Предложите различные способы кодирования ответов на этот вопрос анкеты, при условии, что обработке подлежит большое количество подобных анкет.

3. В некоторых карточных играх используется колода из 32 карт. Придумайте карточный фокус, в результате которого ведущий, задав небольшое количество вопросов, определяет загаданную зрителями карту. Какое количество информации необходимо с точки зрения теории информации, чтобы отгадать карту? Какое количество информации несут ответы на заданные вопросы?
4. Докажите, что гарантированно найти одну фальшивую монету среди 1000 монет, если известно, что она легче, меньше чем за 7 взвешиваний нельзя.
5. Объясните с точки зрения теории информации, почему задачу поиска фальшивой монеты из двух имеющихся монет решить нельзя, если неизвестно, легче или тяжелее она настоящей монеты.
6. Выведите формулу для числа взвешиваний, необходимых для гарантированного нахождения фальшивой монеты среди N монет, если неизвестно, легче или тяжелее она остальных монет.
7. Опираясь на математическую теорию информации, определите, является ли алгоритм двоичного поиска, описанный в предыдущей главе, оптимальным.

§ 5.4. Закон аддитивности информации. Алфавитный подход к измерению информации

Пусть нам теперь необходимо отгадать сразу два независимых предмета x_1 и x_2, про которые известно, что x_1 принадлежит множеству X_1, содержащему N_1 элементов, а x_2 принадлежит множеству X_2, содержащему N_2 элементов. Вполне допустимо считать, что мы должны угадать пару (x_1, x_2) во множестве X всех возможных пар (x_1, x_2), где $x_1 \in X_1$, а $x_2 \in X_2$. Тогда по формуле Хартли для угадывания задуманной пары необходимо задать $\log_2 N_1 N_2$ вопросов, т. е. получить $\log_2 N_1 N_2$ бит информации. Вместе с тем, элементы x_1 и x_2 можно угадывать независимо. Для угадывания x_1 нам понадобится $\log_2 N_1$ вопросов, а для угадывания x_2 — $\log_2 N_2$. Всего при этом понадобится $\log_2 N_1 + \log_2 N_2$ вопросов (бит информации).

Мы получили два выражения для одного и того же количества информации. Согласно основному логарифмическому тождеству, обе величины равны:

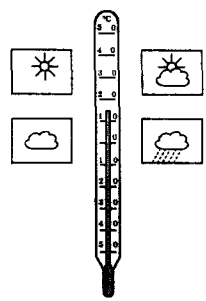
$$\log_2 N_1 N_2 = \log_2 N_1 + \log_2 N_2.$$

Переформулировав данное тождество в терминах количества информации, мы получаем следующий закон.

Закон аддитивности информации. Количество информации $H(x_1, x_2)$, необходимое для установления пары (x_1, x_2), равно сумме количеств информации $H(x_1)$ и $H(x_2)$, необходимых для независимого установления элементов x_1 и x_2:

$$H(x_1, x_2) = H(x_1) + H(x_2). \qquad (5.3)$$

Пример 4. Используя закон аддитивности информации и формулу Хартли, подсчитаем, какое количество информации несет достоверный прогноз погоды.



Предположим, что прогноз погоды на следующий день заключается в предсказании дневной температуры (обычно выбор делается из 16 возможных для данного сезона значений) и одного из четырех значений облачности (солнечно, переменная облачность, пасмурно, дождь).

Получаемое при этом количество информации равно $\log_2 16 + \log_2 4 = 4 + 2 = 6$ бит. □

Закон аддитивности информации справедлив и при алфавитном подходе к измерению информации.

Напомним, что кодирование информации (сообщений) в рамках *алфавитного подхода* осуществляется следующим образом. Для записи сообщений определенного вида вводится свой *алфавит* (набор символов). Количество различных символов в алфавите N называется его *мощностью*. Алфавит вводится так, чтобы любое сообщение интересующего нас вида можно было бы записать только с помощью конечной последовательности символов данного алфавита.

Определение 8. Количество информации, содержащееся в одном символе, называется его *информационным весом*. Согласно формуле Хартли, этот вес равен $\log_2 N$.

В § 5.2 уже был приведен пример минимального алфавита, с помощью которого можно записывать практически произвольные сообщения на английском языке. Но понятие алфавита вовсе не обязательно связано с алфавитами естественных языков (английского, русского, корейского и т. д.). Так, для записи чисел в позиционных системах счисления используется алфавит, мощность которого равна основанию системы счисления P (см. главу 1). Информационный вес символа такого алфавита (цифры) равен $\log_2 P$. Для записи пола человека в анкетах обычно используется двухсимвольный алфавит («м», «ж» или «m», «f»). А для кодирования различных состояний облачности можно использовать четыре фиксированных рисунка, каждый из которых будет представлять собой символ с информационным весом $\log_2 4 = 2$ и т. д. и т. п.

Согласно закону аддитивности информации, можно утверждать, что для хранения двух произвольных символов одного и того же алфавита мощности N потребуется не менее $\log_2 N + \log_2 N = 2\log_2 N$ бит памяти. То есть количество информации, содержащееся в сообщении, состоящем из m символов одного и того же алфавита, равно $m\log_2 N$.

Пример 5. Одна страница художественного прозаического произведения на английском языке может содержать около 2400 символов, включая разделительные пробелы между словами. Тогда для эффективного хранения подобной страницы текста в компьютере понадобится $2400 \cdot \log_2 32$ бит = 12000 бит = 1500 байт памяти вместо 2400 байт при ASCII-представлении текстовой информации. □

Пример 6. Для компьютерной записи столетних наблюдений о погоде, касающихся облачности в Москве, потребуется около $2 \cdot (75 \cdot 365 + 25 \cdot 366)$ бит = 73 050 бит памяти. Столько же вопросов нам придется задать синоптику, чтобы узнать состояние облачности во все дни рассматриваемого столетия. □

Вопросы и задания

1. Выполните упражнение 2 к § 5.2, используя закон аддитивности информации.
2. Определите количество информации в своей фамилии при условии, что для кодирования фамилий будет использоваться 32-символьный алфавит.
3. Алфавит некоторого языка состоит из 32 символов. За сколько секунд мы сможем передать текст из 1600 оптимально закодированных символов этого алфавита, если скорость передачи составляет 100 байт в секунду?
4. В течение 5 секунд было передано сообщение, объем которого составил 375 байт. Каков размер алфавита, с помощью которого записано сообщение, если скорость передачи составила 200 символов в секунду?
5. При игре в кости используют 2 одинаковых кубика, грани которых помечены числами от 1 до 6. Сколько информации несет сообщение о том, что при бросании двух кубиков в сумме выпало 12 очков?
6. В языке некоторого племени всего 16 различных букв. Все слова состоят из 5 букв, всего различных слов в языке 8000. Сколько компьютерной памяти заведомо потребуется для хранения всех слов этого языка?

§ 5.5. Информация и вероятность. Формула Шеннона

Для определения количества информации далеко не всегда возможно использовать формулу Хартли. Ее применяют лишь в частном случае, когда выбор любого элемента из множества, содержащего N элементов, равнозначен. Или, при алфавитном подходе, все символы алфавита встречаются в сообщениях, записанных с помощью этого алфавита, одинаково часто. Однако в действительности так бывает далеко не всегда. В том числе и символы алфавитов естественных языков в сообщениях появляются с разной частотой.

Лемма 3. Пусть наш алфавит состоит из двух символов (a и b), но множество символов сообщения содержит n экземпляров символа a и m экземпляров символа b. Тогда, если нам встретится символ a, мы получаем

$\log_2(m + n) - \log_2 n$ бит информации, а в случае встречи символа b — $\log_2(m + n) - \log_2 m$ бит информации.

Доказательство. Рассмотрим новый алфавит из $n + m$ символов: $a_1, a_2, ..., a_n, b_1, b_2, ..., b_m$. В новом алфавите каждый символ встречается уже с равной частотой и его информационный вес выражается формулой $\log_2(n + m)$. Если при встрече некоторого a_i мы бы распознавали его уникальность, то мы бы получили $\log_2 n$ бит информации, так как таких символов n. Но все символы a_i одинаковы, и нам не важно, какой именно символ a_i нам встретился, и определять это мы не будем. Поэтому в результате отождествления всех символов a_i мы недополучаем (теряем) $\log_2 n$ бит информации, если встречается один из символов a. Следовательно, полученное количество информации равно $\log_2(m + n) - \log_2 n$ бит. Аналогично, мы теряем $\log_2 m$ бит информации, если встречается символ b.

Лемма доказана.

Из леммы 3 следует, что для всех символов нашего алфавита в сумме теряется $n\log_2 n + m\log_2 m$ бит информации. В результате информационный вес одного символа в среднем выражается формулой:

$$H = \log_2(m + n) - \frac{n}{n + m}\log_2 n - \frac{m}{n + m}\log_2 m =$$

$$= \frac{n}{n + m}\log_2 \frac{n + m}{n} + \frac{m}{n + m}\log_2 \frac{n + m}{m}.$$

Назовем *частотой встречаемого* символа a в некотором множестве символов величину $p = n_a/N$, где N — общее количество символов в множестве, а n_a — количество символов a в этом же множестве.

Тогда частота p_1 появления символа a в нашем множестве равна $n/(n + m)$. Аналогично $p_2 = m/(n + m)$. При этом $p_1 + p_2 = 1$. В этих терминах средний информационный вес символа двухсимвольного алфавита выражается формулой

$$H = p_1\log_2(1/p_1) + p_2\log_2(1/p_2).$$

Это и есть *формула Шеннона* для алфавита, состоящего из двух символов.

Пусть у нас есть источник информации, который выдает последовательность символов некоторого алфавита,

при этом разные символы могут появляться с разной частотой. Обозначим $n(a)$ количество символов a среди N символов последовательности.

Определение 9. Если при увеличении длины последовательности n величина $n(a)/N$ стремится к некоторому числу p_a, то это число называется *вероятностью* появления символа a из данного источника.

Для конечных последовательностей понятие вероятности просто совпадает с понятием частоты появления данного символа среди множества символов последовательности. Выведем теперь формулу Шеннона для случая N-символьного алфавита.

Формула Шеннона

Пусть мы имеем алфавит, состоящий из N символов, с частотной характеристикой p_1, p_2, ..., p_N, где p_i выражает вероятность появления i-го символа, так что все вероятности неотрицательны и их сумма равна единице. Тогда средний информационный вес символа такого алфавита выражается формулой

$$H = p_1\log_2(1/p_1) + p_2\log_2(1/p_2) + \ldots + p_N\log_2(1/p_N). \quad (5.4)$$

К. Шеннон (1916–2001)

Доказательство. Для $N = 2$ формула уже доказана. Пусть формула верна для любого алфавита, состоящего из $N - 1$ символа. Отождествим два последних символа нашего алфавита. Вероятность появления любого из этих двух символов равна $(p_{N-1} + p_N)$. Тогда мы имеем алфавит, состоящий из $N - 1$ символа, и, по предположению, средний информационный вес будет выражаться так:

$$H = p_1\log_2(1/p_1) + p_2\log_2(1/p_2) + \ldots + p_{N-2}\log_2(1/p_{N-2}) + \\ + (p_{N-1} + p_N)\log_2(1/(p_{N-1} + p_N)). \quad (5.5)$$

Определим, как часто среди двух отождествленных символов будет появляться каждый из них в отдельности. Такие частоты назовем *относительными вероятностями* появления каждого из двух символов. Они равны

$$q_{N-1} = p_{N-1}/(p_{N-1} + p_N) \text{ и } q_N = p_N/(p_{N-1} + p_N).$$

Согласно лемме 3, при отождествлении мы потеряли

$$q_{N-1}\log_2 1/q_{N-1} + q_N\log_2 1/q_N$$

бит информации на каждый из отождествленных символов, а их доля среди всех символов составляет $p_{N-1} + p_N$. Следовательно, в среднем на один символ потеряно

$$p_{N-1}\log_2 1/q_{N-1} + p_N\log_2 1/q_N \text{ бит.}$$

Если прибавить полученное выражение к формуле (5.5), то после несложных преобразований мы и получим формулу Шеннона.
Формула Шеннона доказана.

Свою формулу Шеннон вывел в 1948 году. Однако еще в XIX веке аналогичную формулу получил Больцман. Он искал ее как формулу для *энтропии*, когда занимался проблемами статистической физики. Энтропия физической системы служит *мерой неупорядоченности* системы. С точки зрения теории информации можно считать, что энтропия системы является *мерой неопределенности состояния элементов системы*, например состояния молекул газа, образующих систему. Эта интерпретация позволяет легко понять, почему Больцман получил такую же формулу, какую Шеннон получил для информации.

Л. Больцман (1844–1906)

Вопрос. *Согласуются ли между собой формулы Хартли и Шеннона?*

Ответ. Чтобы ответить на этот вопрос, применим формулу Шеннона к алфавиту, символы которого в сообщениях равновероятны. То есть $p_1 = p_2 = \ldots = p_N = 1/N$. Тогда формула Шеннона примет вид

$$H = (1/N)\log_2 N + (1/N)\log_2 N + \ldots + (1/N)\log_2 N = \log_2 N,$$

т. е. совпадет с формулой Хартли. □

Вопрос. *Всегда ли ответ «да» или «нет» (а в алфавитном подходе — один символ двухсимвольного алфавита) содержит ровно один бит информации?*

Ответ. Рассмотрим формулу Шеннона в случае $N = 2$. Если вероятность одного ответа равна p, то другого — $(1 - p)$.

Тогда формулу Шеннона можно рассматривать как функцию одной переменной:

$$H(p) = p\log_2 1/p + (1 - p)\log_2 1/(1 - p).$$

Исследуя производную этой функции, несложно доказать, что максимум $H(p)$ на отрезке $[0, 1]$ равен 1 и достигается при $p = 1/2$. Следовательно, один бит информации мы получаем только в том случае, когда ответы «да» и «нет» равновероятны. Так, сообщение о том, что бутерброд упал маслом вниз, несет в себе менее одного бита информации (как известно, бутерброд со стола на пол практически всегда падает маслом вниз). Более того, если мы будем бросать бутерброд несколько раз, а результаты наших экспериментов будем записывать с помощью 0 и 1, то и в среднем один символ полученного двоичного кода будет нести менее одного бита информации. □

Из вышесказанного следует очень важный результат. Кодирование с учетом вероятностей появления различных символов в сообщении можно сделать более экономным, нежели кодирование, осуществленное в предположении равной частоты их появления.

Формула Шеннона показывает средний информационный вес символов того или иного алфавита или *энтропию распределения* частот появления символов соответствующего алфавита.

Дело обстоит проще, если нас интересует количество информации, которое несет тот или иной символ в отдельности (или же, что то же самое, количество вопросов, которое придется задать, чтобы отгадать предмет, вероятность нахождения которого в данном множестве известна). Причем при выводе соответствующей формулы используются те же соображения, что и при выводе формулы Шеннона.

Пусть в нашем множестве находится N предметов. При этом интересующий нас предмет является одним из $m \le N$ одинаковых предметов. То есть вероятность его появления есть $p = m/N$. Если каждый предмет нашего множества считать уникальным, то по формуле Хартли его можно угадать в среднем за $\log_2 N$ вопросов. Но в данном случае, угадав любой из интересующих нас предметов, мы зададим на $\log_2 m$ вопросов меньше, так как

не будем определять, какой именно экземпляр нам попался. Общее число вопросов при этом будет выражаться формулой

$$\log_2 N - \log_2 m = \log_2 N/m = \log_2 1/p. \qquad (5.6)$$

Формула (5.6) верна и в случае, если все N предметов во множестве уникальны (т. е. имеется N различных предметов), но вероятности их появления различны. Для вывода формулы в этом случае используются те же рассуждения, что и при доказательстве леммы 3.

Задача 3. *В результате многолетних наблюдений учитель информатики знает, что у половины его учеников итоговой отметкой за год будет «четверка», у 1/4 учеников — «пятерка», у 1/8 — «тройка», а остальные ученики по разным причинам окажутся неаттестованными. Какое количество информации мы получим после того, как узнаем, какую именно отметку получил ученик?*

Решение. Формула (5.6) говорит о том, что все зависит от конкретной отметки. Если это «четверка», то мы получим всего $\log_2 1/(1/2) = \log_2 2 = 1$ бит информации. Для «пятерки» количество информации равно $\log_2 1/(1/4) = \log_2 4 = 2$ битам. И «тройка», и неудовлетворительный результат каждый в отдельности несут по $\log_2 1/(1/8) = \log_2 8 = 3$ бита информации. Конечно же, «тройка» сама по себе здесь ни при чем! Важно только, насколько часто у учеников встречается именно такая отметка. Причем чем более редкой является отметка, тем больше информации несет сообщение о том, что получена именно она. □

Решить задачу 3 можно было и без использования формулы (5.6). Пусть отметка ученика нам неизвестна и мы хотим ее определить, задавая ему вопросы, подразумевающие ответ «да» или «нет». Сначала разумно спросить, не получил ли ученик отметку «4». Если это не так, то далее имеет смысл спросить про «пятерку». В случае неудачи последним третьим вопросом мы уточним, оказался ли ученик аттестованным вообще. Порядок вопросов здесь не случаен. Фактически мы определяем подмножество, к которому принадлежит ученик согласно своей отметке (всего таких подмножеств 4). По-

сле получения ответа на каждый следующий вопрос количество учащихся, среди которых может находиться наш ученик, уменьшается в два раза, что соответствует получению одного бита информации.

Пример 7. Зная приблизительные частоты, с которыми встречаются буквы русского алфавита (в таблице ниже приведены средние округленные частоты употребления букв русского языка и символа «пробел» на каждую тысячу символов текста на русском языке), можно более точно, чем при использовании алфавитного подхода, не учитывающего вероятности появления различных символов, ответить на вопрос, сколько информации несет то или иное слово, например ваша фамилия.

Буква	пробел	о	е,ё	а	и	т	н	с	р	в	л	к	м	д	п	у
Частота	175	90	72	62	62	53	53	54	40	38	35	28	26	25	23	21
Буква	я	ы	з	ь,ъ	б	г	ч	й	х	ж	ю	ш	ц	щ	э	ф
Частота	18	16	16	14	14	13	12	10	9	7	6	6	4	3	3	2

Для кодирования фамилий мы можем использовать 32-символьный алфавит (отождествив буквы «е» и «ё»). Как уже было показано выше, для кодирования одного символа такого алфавита достаточно 5 бит. То есть, например, фамилия Белов несет 25 бит информации согласно алфавитному подходу. С учетом частотных характеристик букв русского алфавита и закона аддитивности информации информативность этой же фамилии равна 23 битам (или около 21,5, если учесть, что пробел в написании фамилий не используется).

Можно предложить другой подход к измерению информации, если отдельно кодировать корень фамилии, а отдельно — ее окончание. Тогда можно обойтись и меньшим количеством бит. Так, окончание «ов» встречается как минимум в половине мужских фамилий россиян, т. е. несет всего один бит информации. Корень «Бел» встречается, предположим, один раз на 128 фамилий. То есть для его кодирования достаточно 7 бит. Тогда общий код для фамилии Белов теоретически может занимать всего один байт!

Наконец, можно просто узнать, как часто встречается фамилия Белов среди российских фамилий, и подсчитать ее информационный вес. Если предложенные выше цифры относительно частоты употребления в фамилиях корня «Бел» и окончания «ов» верны, то ин-

формационный вес фамилии в целом заключен между 7 и 8 битами. □

Пример 8. В игре «Поле чудес» иногда разрешается открыть произвольную букву слова. Какую позицию следует выбрать? Считается, что искомое слово содержится в имеющемся у нас словаре.

Выберем из словаря все слова, состоящие из того же числа букв, что и отгадываемое слово, и у которых на местах уже разгаданных нами ранее букв стоят те же буквы. Для каждой из неоткрытых позиций подсчитаем энтропию распределения частот появления в этой позиции различных букв. Тогда открывать следует букву в той позиции, энтропия которой максимальна. □

Вопросы и задания

1. Определите количество информации в своей фамилии по таблице частот встречаемости русских букв (см. пример 7).
2. При игре в кости используют 2 одинаковых кубика, грани которых помечены числами от 1 до 6. Сколько информации несет сообщение о том, что при бросании двух кубиков в сумме выпало 4 очка?
3. В урне находятся 8 белых и 24 черных шара. Какое количество информации несет сообщение о том, что из урны достали белый шар? А черный шар?
4. Сообщение о том, что найден цветок сирени с 5 лепестками, несет 7 бит информации. Как часто встречаются подобные цветы на ветках сирени?
5. Для ремонта школы использовали белую, синюю и желтую краски. Израсходовали одинаковое количество белой и синей краски. Сообщение о том, что закончилась банка белой краски, несет 2 бита информации. Синей краски израсходовали 8 банок. Сколько банок желтой краски израсходовали на ремонт школы?
6. Постройте график функции $H(p) = \log_2 1/p$ на интервале (0, 1]. Какие выводы можно сделать при исследовании этого графика?

7. В упражнении 6 из § 5.4 был описан словарь языка некоего племени. Сколько информации на самом деле несет этот словарь, если 16 букв, с помощью которых записаны все слова этого племени, встречаются в словаре со следующими частотными характеристиками: 1/4, 1/8, 1/16, 1/32, 1/32, 1/8, 1/32, 1/16, 1/64, 1/64, 1/16, 1/16, 1/32, 1/32, 1/32, 1/32?

§ 5.6. Оптимальное кодирование информации и ее сложность

Одной из важнейших задач современной информатики, как уже было показано в главе 2, является кодирование информации наиболее коротким образом.

Следствием из формулы Шеннона является то, что не существует универсального способа посимвольного кодирования произвольного файла с частотными характеристиками встречающихся в нем символов $p_1, p_2, \ldots, p_N$, который обеспечивал бы сжатие до величины меньшей, чем

$$p_1\log_2(1/p_1) + p_2\log_2(1/p_2) + \ldots + p_N\log_2(1/p_N)$$

бит на один символ.

Отметим, что если исследовать каждый файл в отдельности, то в каждом конкретном случае более экономный способ кодирования придумать можно, например алгоритм RLE (см. главу 2). При этом на сегодняшний день разработаны универсальные способы кодирования информации, например, алгоритм Хаффмана или арифметического кодирования, которые для произвольного файла строят код, близкий к оптимальной оценке Шеннона.

Рассмотрим идею построения одного из универсальных алгоритмов кодирования. Так как различные символы встречаются в тексте с различной частотой, то естественно кодировать их так, чтобы те, которые встречаются чаще, кодировались более коротко, а те, которые встречаются реже, кодировались длиннее. Но если мы имеем неравномерный код (разные символы закодированы последовательностями разной длины), то возникает проблема: как понять, где кончился код одного символа и начался код другого? Эту проблему можно решать двумя способами:

1) закодированная последовательность имеет вид {длина кода, код}{длина кода, код}...{длина кода, код}. Так устроены, например, коды Rice и Дельта-коды;
2) можно построить так называемый *префиксный код*: в нем не требуется указывать длину кода, но коды получаются несколько длинее.

Напомним, что код называется префиксным, если код одного символа не может быть началом кода другого символа.

Пример 9. Пусть исходный файл состоит только из символов A, B, C и D. При этом имеется 64 буквы A, 32 буквы B, 16 букв C и 16 букв D. Поскольку алфавит в данном случае состоит из четырех символов, каждый символ можно было бы закодировать двумя битами и весь файл поместить в 256 бит. Если же мы закодируем символ A нулевым битом, B — последовательностью битов 10, а символы C и D соответственно последовательностями 110 и 111, то файл сожмется до $64 + 2 \cdot 32 + 3 \cdot 16 + 3 \cdot 16 = 224$ бит. Правда, нам дополнительно придется где-то хранить кодовую таблицу, указывающую, какие символы имеют какие коды.

В данном случае предложенный код 0, 10, 110, 111 является префиксным. Более того, для любого четырехсимвольного алфавита с частотами встречаемости 1/2, 1/4, 1/8, 1/8 рассмотренный способ кодирования является оптимальным, так как каждый символ кодируется двоичной последовательностью длины, совпадающей с информационным весом каждого символа в отдельности. То есть в данном случае достигнута нижняя теоретическая граница сжатия. □

Префиксный код Хаффмана

Для произвольного алфавита одним из наиболее применимых алгоритмов построения префиксного кода, близкого к оптимальному, является уже известный вам *алгоритм Хаффмана*. Докажем, что код, построенный с его помощью, является префиксным.

Приведем конструктивное доказательство. Если текст содержит всего два символа, то один из них кодируется нулем, а второй — единицей. Предположим, что для алфавита, состоящего из N символов, код Хаффмана уже

построен. Рассмотрим алфавит, состоящий из $N + 1$ символов. Отождествим два наиболее редко встречающихся символа этого алфавита и построим код Хаффмана для получившегося N-символьного алфавита. Если $b_1b_2...b_k$ — код Хаффмана для «склеенного» символа, то, не изменяя кодов остальных символов, определим коды «слипшихся» символов как $b_1b_2...b_k0$ и $b_1b_2...b_k1$. Так как последовательность $b_1b_2...b_k$ не является началом никакого другого кода символа нашего алфавита, то построенный код является префиксным.

Пример 10. Построим код Хаффмана для алфавита, состоящего из пяти символов a, b, c, d, e с частотами 0,37(a), 0,22(b), 0,16(c), 0,14(d), 0,11(e). Отождествляя d и e, получаем 0,37(a), 0,25(de), 0,22(b), 0,16(c). Объединяя b и c, имеем 0,38(bc), 0,37(a), 0,25(de). Затем 0,62(ade), 0,38(bc). Сопоставим ade код 0, bc — код 1. Возвращаясь к предпоследнему шагу, расщепим символ ade на a и de с кодами 00 и 01. Затем символ bc расщепляется на b и c с кодами 10 и 11. Наконец, расщепив de, получим для исходного алфавита следующие коды: 00(a), 10(b), 11(c), 010(d), 011(e). □

Коэффициент сжатия информации может быть существенно увеличен за счет учета зависимости появления следующего символа в слове. Так, например, вероятность появления согласной буквы в текущей позиции существенно возрастает, если на предыдущей позиции стоит гласная буква. Новая идея посимвольного кодирования заключается в следующем: для каждого символа алфавита мы вычисляем распределение символов, непосредственно за ним следующих. И для каждого такого распределения мы построим свой код Хаффмана.

Другой способ сжатия файлов основан на анализе *сложности* содержащейся в них информации. Так, последовательность 01010101010101 является менее сложной, чем 1001110000, так как первую из них можно заменить на более короткую, но построенную по другим правилам: $7(01) = 111_2(01)$, а пути сокращения второй последовательности не очевидны. Идея анализа сложности кодируемой информации лежит в основе алгоритмов RLE, LZ (см. главу 2).

Формальное понятие сложности введено, в частности, А. Н. Колмогоровым.

Определение 10. *Сложность объекта или явления* (по Колмогорову) — это минимальное число двоичных знаков (например, нулей и единиц), последовательностью которых можно описать (закодировать) всю информацию об объекте (явлении), достаточное для его дальнейшего воспроизведения (декодирования).

Пример 11. Является ли число $\pi = 3{,}14159256...$ сложным? Как ни странно, нет. Для этого числа существует простое описание:

π = *длина любой окружности/ее диаметр.*

В этом сообщении, которое можно закодировать и с помощью двоичного кода, содержится информация сразу обо всех цифрах этого числа, что невозможно осуществить путем их последовательного выписывания. □

Вопрос. *Можно ли определить, что построенный код для конкретной последовательности — минимальный?*

Ответ. Увы, нет. В главе 4 было рассказано об алгоритмически неразрешимых проблемах, одной из которых является проблема самоприменимости. Основываясь на теореме о самоприменимости, можно доказать, что невозможно построить универсальный алгоритм, определяющий минимальность той или иной двоичной последовательности, описывающей объект, не говоря уже об алгоритме построения такой минимальной двоичной последовательности для объектов вообще. Это не означает, что ни для какого объекта минимальную последовательность построить нельзя. Это значит, что в большинстве случаев невозможно доказать минимальность построенной последовательности. Фактически это означает, что логическим путем нельзя установить ценность информации, содержащейся в сообщении. Исключение составляют, например, последовательности, состоящие из одного двоичного знака (кодовая запись результата падения монеты). Очевидно, что их сократить уже невозможно. □

Вопросы и задания

1. Является ли азбука Морзе примером префиксного кода?
2. Четырехсимвольный алфавит имеет следующее частотное распределение: 0,1; 0,2; 0,3; 0,4. Постройте код Хаффмана для записи сообщений с помощью этого алфавита.

3. Постройте код Хаффмана для 16-буквенного алфавита с частотными характеристиками, указанными в задании 7 предыдущего параграфа. Подсчитав энтропию данного распределения, ответьте на вопрос, является ли полученный код оптимальным.
4. В § 5.6 говорится, что если длина файла невелика, а символы встречаются примерно с одинаковой частотой, то кодирование по Хаффману не приведет к сжатию файла. А для файлов большого размера с существенно различными частотными характеристиками символов выигрыш может быть значительным. Подтвердите (или опровергните) данное высказывание численным моделированием.
5. Напишите программу, позволяющую кодировать и декодировать по Хаффману файлы с известными частотными характеристиками. Сравните работу вашей программы с результатами сжатия тех же файлов с помощью стандартных архиваторов. Назовите преимущества и недостатки оптимального префиксного кода.
6. В главе 2 рассказывалось о современных способах компьютерного хранения звуковой, графической и видеоинформации. Объясните, почему эффективного уменьшения размера соответствующих файлов можно добиться только при некоторой потере информации.

Заключение

В этой главе вы познакомились с основами математической теории информации. Узнали несколько согласованных подходов к измерению количества информации. Формулы Хартли и Шеннона, изученные в данной главе, помогут вам правильно вычислять количество информации в различных случаях. Надеемся, что вы сможете применять их на практике во всевозможных жизненных ситуациях.

Одним из наиболее значимых применений математической теории информации является создание способов записи информации, близких к оптимальному кодированию, о котором было рассказано в материалах учебника. Кодирование по Хаффману, изученное вами, демонстрирует один из таких способов.

Определение сложности объекта или явления по Колмогорову позволяет понять основы всех современных алгоритмов сжатия. А теория информации в целом объясняет, почему невозможно без потери некоторой доли информации сжать тот или иной файл до размера, меньшего определенной величины. Таким образом, становится ясно, что эффективное хранение звуковой, графической и видеоинформации, широко распространенное в настоящее время (форматы MPEG, JPEG и т. п.), невозможно организовать без некоторой потери качества.

Если данная тема вас заинтересовала, то в материалах учебника можно найти немало проблем, требующих более детального исследования с помощью методов математической теории информации. Попробуйте придумать оптимальные стратегии поведения в тех или иных телевизионных играх или постарайтесь изобрести свой собственный способ сжатия информации, запрограммируйте его и сравните с известными программами-архиваторами. Творите, и у вас все получится!

Глава 6

Математические основы вычислительной геометрии и компьютерной графики

Геометрия есть искусство правильно рассуждать на неправильных чертежах.

Д. Пойа

С тех пор, как Евклид написал свои «Элементы геометрии», не нашлось никого, кто бы их отверг... В этих «Элементах...» содержится непреложная истина, с которой разум, раз познав ее, не может не согласиться.

П. Гассенди

Когда вы играете в компьютерные игры, смотрите рекламные ролики или становитесь свидетелями гибели «Титаника» на экране кинематографа, то невольно погружаетесь в волшебный мир компьютерной графики. Задачи компьютерной графики возникли относительно недавно — *в середине 80-х годов прошлого* столетия вместе с появлением графических дисплеев. Параллельно с совершенствованием аппаратных средств росла и сложность математических задач, решение которых становилось при этом возможным. Настоящую революцию среди 3D-игр (от английского *3-dimensional*) произвел Doom. Его так называемый «движок» получил широкое распространение среди разработчиков компьютерных игр, т. е. «стал властителем дум» ☺. При создании подобных шедевров компьютерной графики должны сойтись воедино мастерство художника, искусство программиста и знания математика.

Большинство проблем компьютерной графики решается с помощью методов вычислительной геометрии. Компьютерная графика занимается представлением различных линий и геометрических фигур, но сам компьютер, как уже было показано выше, может обрабатывать только числовую информацию, поэтому в первую очередь мы должны научиться описывать различные геометрические объекты формулами, т. е. представлять их аналитически. Эти проблемы решаются в математике в рамках раздела «Аналитическая геометрия». В настоящей главе мы рассмотрим те из них, которые не требуют привлечения аппарата высшей математики.

Раздел информатики, изучающий алгоритмы решения задач аналитической геометрии, называется *вычислительной геометрией*. Одной из ее задач является получение аналитических формул и алгоритмов обработки геометрических фигур в виде, удобном именно для компьютерной реализации. Как «не все йогурты одинаково полезны», так и не всякое математическое описание геометрических объектов может быть при этом использовано.

Задачи аналитического описания геометрических объектов возникают не только в компьютерной графике, но и при проектировании интегральных схем, технических устройств и др. Иметь дело с геометрией приходится и мэру любого города. Ведь перед ним может встать

любая из следующих задач: как разделить город на районы примерно равной площади и с одинаковым числом жителей, где построить новый культурный центр, а где — химический завод и т. д. и т. п. Ответы на эти и многие другие вопросы тоже может дать вычислительная геометрия.

Исходными данными в задачах подобного рода могут быть множество точек, набор отрезков или многоугольников и т. п. Результатом является либо ответ на какой-то вопрос (типа «пересекаются ли проекции этих объектов»; подобная проблема постоянно возникает как раз при программировании компьютерных игр), либо какой-то геометрический объект (например, наименьший выпуклый многоугольник, содержащий заданные точки; на практике эта задача может соответствовать проектированию кольцевой дороги вокруг населенного пункта).

Цель настоящей главы — показать такие подходы к решению геометрических задач, которые позволяют достаточно быстро и максимально просто получать решения большинства элементарных задач вычислительной геометрии.

§ 6.1. Координаты и векторы на плоскости

Для того чтобы геометрические образы перевести на язык формул, в первую очередь необходимо ввести систему координат (СК). В аналитической геометрии используются различные СК на плоскости и в пространстве, но мы будем работать с прямоугольной декартовой СК. Общепринято выбирать координатные оси так, чтобы поворот на угол $\pi/2$, при котором ось Ox совмещается с осью Oy, происходил против часовой стрелки. Такую СК называют *правой*. В дальнейшем подразумевается, что наша СК правая. В такой СК направление поворота против часовой стрелки называется *положительным*.

Теперь с каждой точкой на плоскости будут связаны два числа — координаты этой точки в данной СК, и геометрические объекты могут получить аналитическое выражение. Так, чтобы задать отрезок, достаточно указать координаты его концов. Прямую можно задать, указав пару ее точек, либо координатами одной ее точки и век-

тором, характеризующим направление этой прямой и т. д. Основным инструментом при решении задач для нас будут векторы. Напомним некоторые сведения о них.

Определение 1. Отрезок AB, у которого точку A считают началом (точкой приложения), а точку B — концом, называют *вектором* AB и обозначают либо $\overline{AB}$, либо жирной строчной латинской буквой, например $\mathbf{a}$. Для обозначения длины вектора (т. е. длины соответствующего отрезка) будем пользоваться символом модуля (например, $|\boldsymbol{a}|$). Два вектора называются *равными*, если они совмещаются параллельным переносом.

Пусть точки A и B имеют координаты (x_1, y_1) и (x_2, y_2) соответственно. Координатами вектора $\overline{AB}$ называется пара чисел $(x_2 - x_1, y_2 - y_1)$. Наоборот, если вектор $\mathbf{a}$ имеет координаты (a_x, a_y) и приложен к точке (x_1, y_1), то легко вычислить координаты (x_2, y_2) его конца:

$$x_2 = x_1 + a_x, \quad y_2 = y_1 + a_y.$$

Длина вектора $\overline{AB}$ (обозначается $|\overline{AB}|$) по теореме Пифагора равна $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. Равенство двух векторов $\mathbf{a} = (a_x, a_y)$ и $\mathbf{b} = (b_x, b_y)$ эквивалентно равенству их соответствующих координат:

$$a_x = b_x, \; a_y = b_y.$$

Любая упорядоченная пара чисел (x, y) может также рассматриваться как вектор. Геометрически вектор (x, y) — это направленный отрезок, начинающийся в точке $(0, 0)$ и заканчивающийся в точке (x, y). Вектор, заданный двумя своими координатами (x, y), называют *свободным вектором*. Всем векторам, заданным координатами начала и конца и равным между собой, соответствует один и тот же свободный вектор. Для такого вектора не определена точка приложения.

Векторы можно складывать и умножать на числа. Сложение векторов производится по правилу треугольника или по правилу параллелограмма (рис. 6.1). Под разностью векторов $\mathbf{a}$ и $\mathbf{b}$ понимают сумму вектора $\mathbf{a}$ и вектора, противоположного вектору $\mathbf{b}$ (т. е. совпадающего с ним по длине, но противоположно направленного). При умножении вектора $\mathbf{a}$ на число t получается

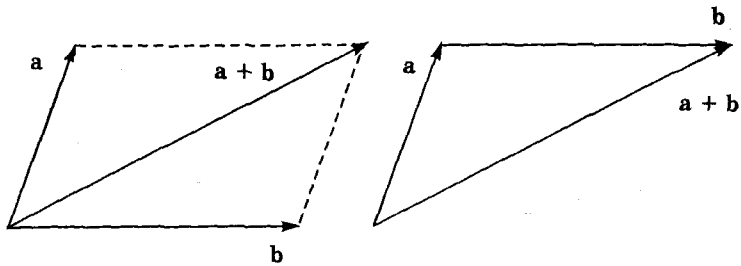


Рис. 6.1

вектор, имеющий длину $|t|\cdot|\mathbf{a}|$; его направление совпадает с направлением $\mathbf{a}$, если $t > 0$, и противоположно ему, если $t < 0$.

Операция умножения вектора на числовую константу дает возможность ввести понятие коллинеарности и неколлинеарности векторов.

Определение 2. Два вектора $\mathbf{a}$ и $\mathbf{b}$ называются *коллинеарными* (сонаправленными или противоположно направленными), если хотя бы один из них может быть выражен через другой в виде формулы $\mathbf{b} = t\cdot\mathbf{a}$. При этом будем считать, что $\dfrac{\mathbf{b}}{\mathbf{a}} = t$.

Выразим введенные операции над векторами через их координаты. Если если $\mathbf{a} = (a_x,\ a_y)$ и $\mathbf{b} = (b_x,\ b_y)$, то $\mathbf{a} + \mathbf{b} = (a_x + b_x,\ a_y + b_y)$, $\mathbf{a} - \mathbf{b} = (a_x - b_x,\ a_y - b_y)$ и $t\cdot\mathbf{a} = (t\cdot a_x,\ t\cdot a_y)$.

Отметим еще, что вектор, сонаправленный с данным вектором $\mathbf{a}$ и имеющий заданную длину l, можно выразить следующим образом: $\dfrac{l}{|\mathbf{a}|}\mathbf{a}$. В дальнейшем мы неоднократно будем этим пользоваться.

Вопрос. *Пусть нам известно, что $\overline{AB}/\overline{AC} < 0$. Что вы можете сказать о взаимном расположении точек A, B и C?*

Ответ. Из приведенного выше неравенства следует, что векторы $\overline{AB}$ и $\overline{AC}$ коллинеарны и $\overline{AB} = t\cdot\overline{AC}$, где $t < 0$. Это означает, что точки A, B и C лежат на одной прямой, и векторы $\overline{AB}$ и $\overline{AC}$ отложены от одной точки A в противоположных направлениях. Следовательно, точка A лежит между B и C. □

Определение 3. *Скалярным произведением двух ненулевых векторов* $\mathbf{a} = (a_x, a_y)$ и $\mathbf{b} = (b_x, b_y)$ называется число $(\mathbf{a}, \mathbf{b}) = |\mathbf{a}|\cdot|\mathbf{b}|\cos\varphi$, где φ — угол между этими векторами. В координатах оно вычисляется так: $(\mathbf{a}, \mathbf{b}) = a_x b_x + a_y b_y$.

Вопрос. *Как можно использовать скалярное произведение для характеристики угла между векторами?*

Ответ. Из приведенных в определении 3 формул следует, что два ненулевых вектора перпендикулярны тогда и только тогда, когда их скалярное произведение равно нулю ($\cos \pi/2 = 0$). Так как $\cos\varphi$ положителен для острых углов и отрицателен для тупых, угол между векторами острый (тупой) в том и только том случае, когда их скалярное произведение положительно (отрицательно). □

Пусть $\mathbf{a}$ и $\mathbf{b}$ — два ненулевых вектора, отложенные от одной точки. В школьном курсе геометрии под углом между векторами понимается меньший из двух углов между лучами, на которых лежат векторы $\mathbf{a}$ и $\mathbf{b}$. Значение такого угла всегда находится в промежутке $[0; \pi]$.

Для вычислений часто более удобным оказывается понятие *ориентированного угла*, т. е. угла, учитывающего порядок перечисления векторов. Ориентированный угол по абсолютной величине равен обычному углу между векторами. Ориентированный угол между векторами $\mathbf{a}$ и $\mathbf{b}$ положительный, если поворот от вектора $\mathbf{a}$ к вектору $\mathbf{b}$ совершается в положительном направлении (в нашей СК против часовой стрелки), и отрицательный в противном случае.

В связи с этим говорят также, что пара векторов $\mathbf{a}$ и $\mathbf{b}$ *положительно* (*отрицательно*) *ориентирована*. Ориентированный угол может принимать значения от $-\pi$ до π. На рис. 6.2 ориентированные углы, например, между векторами $\overline{OA}$ и $\overline{OB}$ и между векторами $\overline{OB}$ и $\overline{OA}$ равны по модулю, но первый из них отрицательный, а второй — положительный.

Вопрос. *Как, зная координаты векторов, найти угол между ними?*

Ответ. Очевидный способ следует из формулы для скалярного произведения: $\cos\varphi = (\mathbf{a}, \mathbf{b})/(|\mathbf{a}|\cdot|\mathbf{b}|)$. Однако при этом получится значение неориентированного угла и часть

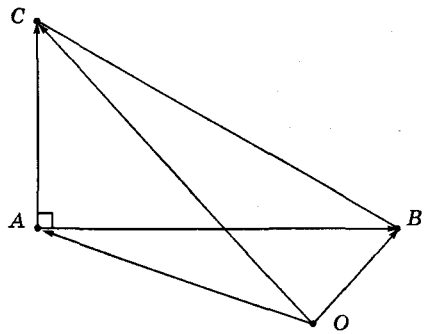


Рис. 6.2

информации (возможно, полезная) будет нами потеряна. Кроме того, использование этой формулы для программирования не всегда удобно. Например, в языке Паскаль, как и в ряде других языков программирования, из обратных тригонометрических функций реализована только функция arctg φ. Мы покажем, как найти угол иначе, после того как познакомимся с ориентированной площадью треугольника. □

Ориентированная площадь треугольника — это его обычная площадь, снабженная знаком. Знак у ориентированной площади треугольника ABC такой же, как у ориентированного угла между векторами $\overline{AB}$ и $\overline{AC}$. То есть ее знак зависит от порядка перечисления вершин. Если вершины треугольника перечисляются против часовой стрелки, ориентированная площадь положительна, а если по часовой стрелке — отрицательна. На рис. 6.2 треугольник ABC — прямоугольный. Его ориентированная площадь равна $|AB|\cdot|AC|/2$. Эту же величину можно вычислить другим способом. Пусть O — произвольная точка плоскости. На нашем рисунке площадь треугольника ABC получится, если из площади треугольника OBC вычесть площадь треугольника OAB и прибавить площадь треугольника OCA. Таким образом, нужно просто сложить ориентированные площади треугольников OAB, OBC и OCA. Это правило работает при любом выборе точки O.

Посмотрим, как выразить ориентированную площадь треугольника в координатах. Пусть S — ориентированная площадь треугольника OAB, построенного на векто-

рах $\mathbf{a} = (a_x, a_y)$ и $\mathbf{b} = (b_x, b_y)$. Вычислим ее для конкретного расположения векторов (рис. 6.3). Величина S здесь положительна (пара векторов $\overline{OA}$ и $\overline{OB}$ положительно ориентирована). Достроим наш треугольник до параллелограмма $OACB$ площади $2S$ (здесь $\overline{OC} = \overline{OA} + \overline{OB}$). Тогда площадь прямоугольника OC_1CC_2 равна $|OC_1|\cdot|OC_2| = (a_x + b_x)(a_y + b_y)$. Вместе с тем ее можно выразить так: $2S + 2S_1 + 2S_2 + 2S_3 = 2S + a_xa_y + b_xb_y + 2b_xa_y$ (здесь S_1, S_2, S_3 — обычные неориентированные площади). Раскрыв скобки в первом выражении для площади прямоугольника и выразив $2S$ из равенства двух представлений, получим:

$$2S = a_xb_y - b_xa_y. \qquad (6.1)$$

Нетрудно убедиться, что и при других вариантах расположения векторов формула (6.1) также остается справедливой. Она показывает, что ориентированная площадь параллелограмма, построенного на векторах $\mathbf{a} = (a_x, a_y)$ и $\mathbf{b} = (b_x, b_y)$, равна $a_xb_y - b_xa_y$ (а обычная площадь — модулю этой же величины: $|a_xb_y - b_xa_y|$).

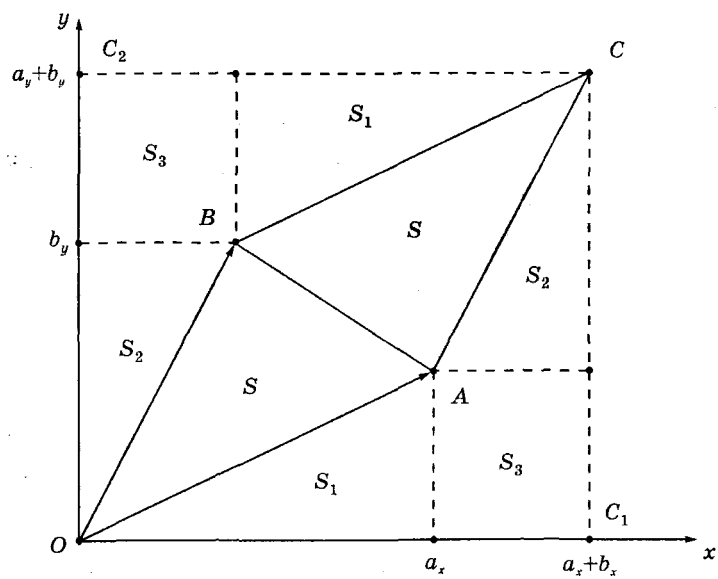


Рис. 6.3

Определение 4. Величина $a_x b_y - b_x a_y$ называется *косым* (или *псевдоскалярным*) *произведением векторов* **a** и **b**.

Для косого произведения мы будем употреблять обозначение [**a**, **b**]. Заметим, что в дальнейшем при определении операций над векторами в пространстве мы будем использовать это же обозначение в несколько другом смысле. Название рассматриваемой здесь величины связано со свойством косой симметрии: [**a**, **b**] = −[**b**, **a**].

Так как неориентированная площадь параллелограмма, построенного на векторах **a** и **b**, равна $|\mathbf{a}|\cdot|\mathbf{b}|\cdot|\sin\varphi|$, а знак $\sin\varphi$ совпадает со знаком ориентированного угла φ, то [**a**, **b**] = $|\mathbf{a}|\cdot|\mathbf{b}|\cdot\sin\varphi$. Величина [**a**, **b**] больше нуля, если пара векторов **a** и **b** положительно ориентирована и меньше нуля в противном случае. Косое произведение ненулевых векторов равно нулю тогда и только тогда, когда они коллинеарны ($\sin 0 = \sin\pi = 0$).

Теперь, как и обещали, найдем в координатах угол между двумя векторами. Пусть φ — ориентированный угол между ненулевыми векторами $\mathbf{a} = (a_x, a_y)$ и $\mathbf{b} = (b_x, b_y)$. Сопоставляя формулы для скалярного и косого произведения этих векторов, имеем

$$\operatorname{tg}\varphi = [\mathbf{a}, \mathbf{b}]/(\mathbf{a}, \mathbf{b}) = (a_x b_y - b_x a_y)/(a_x b_x + a_y b_y).$$

Зная тангенс угла между векторами, мы легко найдем угол между прямыми, на которых лежат **a** и **b**: он равен $|\operatorname{arctg}([\mathbf{a}, \mathbf{b}]/(\mathbf{a}, \mathbf{b}))|$. Чтобы получить ориентированный угол между векторами, осталось выяснить, острый он или тупой. Это мы определим по знаку скалярного произведения. Учтем еще, что знак ориентированного угла совпадает со знаком косого произведения. Тогда окончательно имеем:

$$\varphi = \begin{cases} \pi/2, & \text{если } (\mathbf{a}, \mathbf{b}) = 0,\ [\mathbf{a}, \mathbf{b}] > 0; \\ -\pi/2, & \text{если } (\mathbf{a}, \mathbf{b}) = 0,\ [\mathbf{a}, \mathbf{b}] < 0; \\ \operatorname{arctg}([\mathbf{a}, \mathbf{b}]/(\mathbf{a}, \mathbf{b})), & \text{если } (\mathbf{a}, \mathbf{b}) > 0; \\ \operatorname{arctg}([\mathbf{a}, \mathbf{b}]/(\mathbf{a}, \mathbf{b})) + \pi, & \text{если } (\mathbf{a}, \mathbf{b}) < 0,\ [\mathbf{a}, \mathbf{b}] \geq 0; \\ \operatorname{arctg}([\mathbf{a}, \mathbf{b}]/(\mathbf{a}, \mathbf{b})) - \pi, & \text{если } (\mathbf{a}, \mathbf{b}) < 0,\ [\mathbf{a}, \mathbf{b}] < 0. \end{cases} \quad (6.2)$$

Величина обычного угла равна модулю значения ориентированного угла.

Отметим, что все сказанное об ориентированных углах и площадях относилось к правой СК. Может статься, что для конкретной задачи удобнее ввести левую СК. К примеру, координаты пикселей на экране монитора даются именно в левой СК (ось абсцисс смотрит вправо, ось ординат — вниз). При таком выборе осей положительным является поворот *по часовой стрелке*. С этой поправкой все вышеизложенное применимо и к левой СК.

Вопросы и задания

1. С какими новыми понятиями, не известными вам из курса геометрии, вы познакомились при изучении данного параграфа?
2. Выпишите формулы, аналогичные (6.2), для определения неориентированного угла между двумя векторами, значение которого лежит в диапазоне $[0; \pi]$.
3. Пусть три точки на плоскости заданы своими координатами. Используя скалярное и псевдоскалярное произведение векторов, образованных указанными точками, сформулируйте алгоритм определения одного из следующих взаимных расположений данных точек:
   а) все три точки совпадают;
   б) совпадают ровно две точки;
   в) три точки различны и лежат на одной прямой;
   г) три точки образуют прямоугольный треугольник;
   д) три точки образуют остроугольный треугольник;
   е) три точки образуют тупоугольный треугольник.

§ 6.2. Способы описания линий на плоскости

6.2.1. Общее уравнение прямой

В школьной математике для описания прямой на плоскости используется формула

$$y = kx + b, \text{ где } k = \operatorname{tg}\alpha,\ b = y(0).$$

Здесь α — угол наклона описываемой прямой к оси Ox. Этот способ описания называется *формулой с угловым коэффициентом*. Она проста и понятна, однако не описывает прямые, параллельные оси Oy. Данный случай можно было бы учитывать отдельно, но при программировании это очень неудобно. Поэтому выведем общее

уравнение прямой, используемое в аналитической геометрии.

Пусть на прямой заданы две несовпадающие точки: P_1 с координатами (x_1, y_1) и P_2 с координатами (x_2, y_2). Соответственно вектор с началом в точке P_1 и концом в точке P_2 имеет координаты $(x_2 - x_1, y_2 - y_1)$. Если $P(x, y)$ — произвольная точка на нашей прямой, то координаты вектора $\overline{P_1P}$ равны $(x - x_1, y - y_1)$. С помощью косого произведения условие коллинеарности векторов $\overline{P_1P}$ и $\overline{P_1P_2}$ можно выразить так:

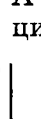
$$[\overline{P_1P}, \overline{P_1P_2}] = (x - x_1)(y_2 - y_1) - (y - y_1)(x_2 - x_1) = 0, \text{ или}$$

$$(y_2 - y_1)x + (x_1 - x_2)y + x_1(y_1 - y_2) + y_1(x_2 - x_1) = 0. \quad (6.3)$$

Второе из уравнений (6.3) перепишем следующим образом:

$$ax + by + c = 0, \quad (6.4)$$

где $a = y_2 - y_1$, $b = x_1 - x_2$, $c = x_1(y_1 - y_2) + y_1(x_2 - x_1)$. Уравнение (6.4) и носит название *общего уравнения прямой*. Итак, всякую прямую можно задать уравнением вида (6.4). Ниже мы покажем, что и наоборот, при любых значениях коэффициентов (кроме $a = b = 0$) уравнение такого вида задает на плоскости некоторую прямую. А также рассмотрим геометрический смысл его коэффициентов.



При программировании первую из формул (6.3) нельзя использовать в форме отношения $(x_2 - x_1)/(x - x_1) = (y_2 - y_1)/(y - y_1)$, так как, во-первых, даже если все координаты заданных точек целые, как это бывает в компьютерной графике, особенности реализации операции деления в вещественной арифметике не позволят с помощью указанного соотношения проверить принадлежность той или иной точки данной прямой, во-вторых, если точка P совпадет с P_1, программа будет прервана в силу деления на ноль.

Определение 5. Любой ненулевой вектор, ортогональный прямой (т. е. ортогональный направляющему вектору прямой), называется *нормалью* к этой прямой.

Пусть заданная точка P_0 прямой имеет координаты (x_0, y_0), а некоторый вектор нормали $\mathbf{n}$ к ней — координ-

наты (a, b). Если $P(x, y)$ — произвольная точка на нашей прямой, то координаты вектора $\overline{P_0P}$ равны $(x - x_0, y - y_0)$. Теперь скалярное произведение ортогональных векторов $(\mathbf{n}, \overline{P_0P})$ можно выразить так:

$$(\mathbf{n}, \overline{P_0P}) = a(x - x_0) + b(y - y_0) = 0. \qquad (6.5)$$

Очевидно, что уравнение (6.5) также несложно привести к виду (6.4). Тогда становится понятно, что коэффициенты a и b из уравнения (6.4) представляют собой координаты одного из векторов нормали к описываемой данным уравнением прямой. Отсюда следует, что при любых значениях коэффициентов a, b и c (кроме $a = b = 0$) уравнение (6.4) задает прямую. Ею будет прямая, перпендикулярная вектору (a, b) и проходящая через точку, чьи координаты удовлетворяют (6.4). При $a \neq 0$ такой точкой будет, например, точка $(-c/a, 0)$, при $a = 0$ — точка $(0, -c/b)$.

Заметим, что в общем уравнении прямой коэффициенты a, b и c зависят от длины вектора нормали, т. е. определяются с точностью до множителя. Если в качестве вектора нормали мы выберем вектор единичной длины, то мы придем к нормированному уравнению прямой.

6.2.2. Нормированное уравнение прямой

Через начало координат (точка O) проведем прямую, перпендикулярную нашей прямой L, точку их пересечения обозначим через M, длину отрезка OM обозначим через r (рис. 6.4). На построенной прямой возьмем вектор $\mathbf{n}$ единичной длины, направление которого совпадает с направлением $\overline{OM}$ (если наша прямая L проходит через начало координат, то можно выбрать любое из двух направлений).

Попробуем выразить общее уравнение нашей прямой через два параметра: r — длину вектора $\overline{OM}$ и угол θ между вектором $\mathbf{n}$ и осью Ox. Для этого воспользуемся двумя достаточно очевидными фактами.

1) Так как $|\mathbf{n}| = 1$, то его координаты можно представить в виде $\mathbf{n} = (\cos\theta, \sin\theta)$.

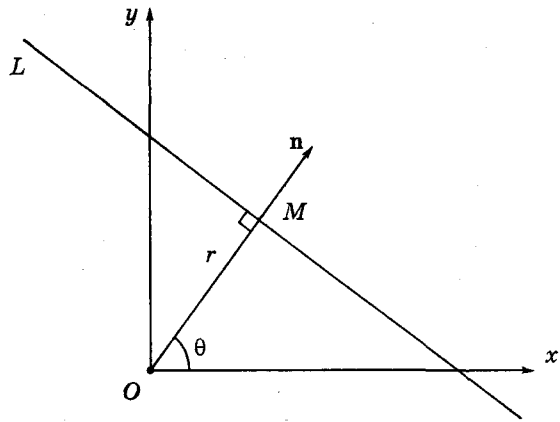


Рис. 6.4

2) Произвольная точка $P(x, y)$ принадлежит нашей прямой L тогда и только тогда, когда
$\overline{OP}\cdot\cos(\mathbf{n}, \overline{OP}) = r$.

Тогда, с одной стороны,

$(\mathbf{n}, \overline{OP}) = x\cos\theta + y\sin\theta$,

а с другой стороны

$(\mathbf{n}, \overline{OP}) = |\mathbf{n}|\cdot|\overline{OP}|\cdot\cos(\mathbf{n}, \overline{OP}) = r$, так как $|\mathbf{n}| = 1$.

Таким образом, $x\cos\theta + y\sin\theta = r$ или

$$x\cos\theta + y\sin\theta - r = 0. \qquad (6.6)$$

Полученное уравнение носит название *нормированного уравнения прямой*. Данное уравнение определяет прямую всего двумя параметрами: углом наклона θ к оси Ox вектора нормали, проведенного из начала координат в направлении нашей прямой L, и расстоянием r от начала координат до нашей прямой.

Вычислить величины r и θ по общему уравнению прямой можно следующим образом.

1. Добьемся, чтобы коэффициент c в общем уравнении прямой был отрицательным. Если изначально это не так, умножим исходное уравнение на -1.
2. Разделим все члены уравнения на величину $\sqrt{a^2 + b^2}$.

Тем самым получим:

$$\frac{a}{\sqrt{a^2+b^2}}x+\frac{b}{\sqrt{a^2+b^2}}y-\frac{|c|}{\sqrt{a^2+b^2}}=0.$$

Здесь $\frac{a}{\sqrt{a^2+b^2}}=\cos\theta$, $\frac{b}{\sqrt{a^2+b^2}}=\sin\theta$, $\frac{|c|}{\sqrt{a^2+b^2}}=r$.

6.2.3. Параметрические уравнения прямой, луча, отрезка

Уравнение прямой, проходящей через несовпадающие точки $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$, можно записывать и в параметрическом виде. Любой вектор, приложенный к точке P_1 и заканчивающийся в произвольной точке $P(x, y)$, лежащей на той же прямой, по определению 2 можно получить из $\overline{P_1P_2}$ путем умножения на некоторое вещественное число t:

$$\overline{PP_1}=t\cdot\overline{P_1P_2}.$$

Тогда для каждой из координат в отдельности справедливо:

$$(x-x_1)=t(x_2-x_1) \text{ и } (y-y_1)=t(y_2-y_1).$$

Выразив отсюда x и y, получаем систему параметрических уравнений, которой удовлетворяют координаты каждой точки $P(x, y)$ нашей прямой:

$$\begin{cases} x=x_1+t(x_2-x_1), \\ y=y_1+t(y_2-y_1). \end{cases} \qquad (6.7)$$

Наоборот, если координаты (x, y) точки P удовлетворяют соотношениям (6.7), вектор $\overline{P_1P}$ коллинеарен $\overline{P_1P_2}$ и, значит, точка P лежит на прямой P_1P_2. Таким образом, система уравнений (6.7), где параметр t пробегает всю действительную ось, задает прямую P_1P_2.

Эта же система, но со введенными ограничениями на значения t, будет задавать и отрезок P_1P_2, и луч P_1P_2, начинающийся в точке P_1 и проходящий через точку P_2, не совпадающую с P_1. Координата x отрезка P_1P_2 меняется в диапазоне $[x_1; x_2]$, а y — в диапазоне $[y_1; y_2]$. Следовательно, $t\in[0; 1]$. Для всех точек $P(x, y)$, принадлежащих лучу P_1P_2, вектор $\overline{P_1P}$ сонаправлен с вектором $\overline{P_1P_2}$. То есть для луча $t\in[0; \infty)$.

6.2.4. Способы описания окружности

Известно, что точка $M(x, y)$ принадлежит окружности с центром в точке $M_0(x_0, y_0)$ и радиусом r тогда и только тогда, когда расстояние между M_0 и M равно r. Записав формулу для вычисления квадрата расстояния между двумя точками, мы придем к следующему уравнению окружности:

$$(x - x_0)^2 + (y - y_0)^2 = r^2. \qquad (6.8)$$

На практике оказывается полезным знание также и параметрических уравнений окружности. Обратимся сначала к окружности с центром в начале координат. Если обозначить как t угол между радиус-вектором OM (здесь $M(x, y)$ — произвольная точка окружности) и осью Ox, отсчитываемый против часовой стрелки, то очевидно, что

$$x = r\cos t,\ y = r\sin t.$$

Значит, для произвольной окружности параметрические уравнения будут выглядеть так:

$$x = x_0 + r\cos t,\ y = y_0 + r\sin t. \qquad (6.9)$$

Уравнения, аналогичные (6.9), используются для описания таких кривых, как циклоида, эпициклоида, гипоциклоида и многих других, широко применяемых в компьютерной графике для получения различных узоров (рис. 6.5). Именно с помощью параметрических уравнений достаточно просто запрограммировать рисование подобных кривых на графическом экране.

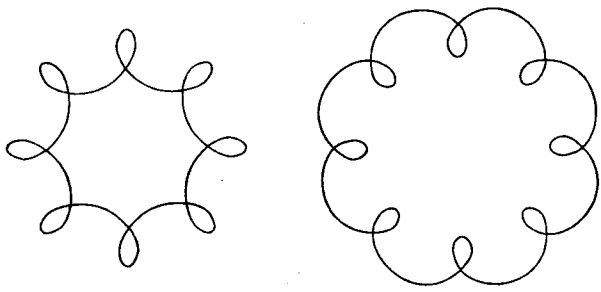


Рис. 6.5. Гипоциклоида и эпициклоида

Вопросы и задания

1. Исходя из геометрического смысла коэффициентов нормированного уравнения прямой (6.6), выведите уравнение прямой, параллельной данной и находящейся от нее на заданном расстоянии r.
2. Покажите, что выражение $|x_1\cos\theta + y_1\sin\theta - r|$ для произвольной точки плоскости (x_1, y_1) обозначает расстояние от этой точки до прямой $x\cos\theta + y\sin\theta - r = 0$.
3. Используя (6.7), составьте и решите систему линейных уравнений для нахождения точки пересечения двух прямых. С помощью решения этой же системы сформулируйте, как можно определить, пересекаются ли соответствующие отрезки (лучи).
4. Найдите в справочной литературе параметрические уравнения циклоиды, эпициклоиды и гипоциклоиды. С помощью любого известного вам языка программирования или электронной таблицы получите изображения данных кривых. Исследуйте их при различных значениях параметров.

§ 6.3. Задачи компьютерной графики на взаимное расположение точек и фигур

6.3.1. Прямая, перпендикулярная данной и проходящая через заданную точку

В компьютерной графике часто возникает необходимость восстановить квадрат, стороны которого не обязательно параллельны осям координат, по одной из его сторон. Для решения этой задачи надо уметь строить прямую, перпендикулярную данной и проходящую через заданную точку.

Пусть заданная точка P_0 искомой прямой имеет координаты (x_0, y_0). Если $P(x, y)$ — произвольная точка на той же прямой, то координаты вектора $\overline{P_0P}$ равны $(x - x_0, y - y_0)$. Этот вектор перпендикулярен вектору $\overline{P_1P_2}$, где $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$ — точки на исходной прямой. Тогда скалярное произведение ортогональных векторов $(\overline{P_1P_2}, \overline{P_0P})$ можно выразить так:

$$(x_2 - x_1)(x - x_0) + (y_2 - y_1)(y - y_0) = 0 \quad \text{или}$$
$$(x_2 - x_1)x + (y_2 - y_1)y + (x_1 - x_2)x_0 + (y_1 - y_2)y_0 = 0. \quad (6.10)$$

Если же исходная прямая задана коэффициентами a, b и c своего уравнения, то легко заметить, что вектор ее нормали с координатами (a, b) коллинеарен вектору $\overline{P_0P}$. Тогда, записывая косое произведение этих векторов, получим:

$$b(x - x_0) - a(y - y_0) = 0. \qquad (6.11)$$

6.3.2. Расположение точки относительно прямой, луча или отрезка

В первую очередь, в этой задаче нас интересует принадлежность данной точки $P(x, y)$ указанному геометрическому объекту, уравнение которого нам известно (либо может быть легко получено). Чтобы ответить на этот вопрос для прямой, достаточно подставить координаты заданной точки в уравнение прямой, например вида (6.4). Равенство нулю значения полученного выражения (для вещественных координат или коэффициентов уравнения проверку на равенство нулю необходимо осуществлять с учетом погрешности) означает, что точка принадлежит данной прямой. Если значение выражения меньше нуля, то точка лежит в одной полуплоскости от прямой, если больше нуля — в другой. Если нам заданы $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$ — две точки нашей прямой, то запишем выражение для косого произведения векторов $\overline{P_1P_2}$ и $\overline{P_1P}$. Его знак определяет ориентацию этой пары векторов, иначе говоря, принадлежность точки P одной из полуплоскостей (а равенство нулю — принадлежность прямой).

Если прямая задана двумя своими точками, то для определения расположения точки плоскости относительно этой прямой уравнение прямой выписывать не нужно.

В случае проверки принадлежности точки лучу при равенстве $[\overline{P_1P_2}, \overline{P_1P}]$ нулю (здесь P_1 — начало луча, а P_2 — любая точка, принадлежащая лучу) полезно вычислить и скалярное произведение этих же векторов. Если оно меньше нуля, то P_1 лежит на прямой между P_2 и P, следовательно, P лучу не принадлежит. Чтобы

в аналогичной ситуации убедиться в принадлежности точки P отрезку P_1P_2, необходимо вычислить еще и значение скалярного произведения $(\overline{P_2P_1}, \overline{P_2P})$. Если оно неотрицательно, то точка P лежит на отрезке.

Подобные задачи в компьютерной графике приходится решать для точки графического экрана и линий (отрезков), описывающих границы тех или иных объектов.

Вопрос. *Как определить, на каком расстоянии находится заданная точка P от определенной прямой, луча или отрезка?*

Ответ. Формула для расстояния от точки до прямой получается из сопоставления двух способов вычисления площади треугольника: $S = 1/2 \cdot |P_0P| \cdot |P_1P_2| = 1/2 \cdot |[\overline{PP_1}, \overline{PP_2}]|$ (рис. 6.6). То есть расстояние от точки P до прямой, заданной координатами точек P_1 и P_2, можно подсчитать как отношение модуля косого произведения векторов $\overline{PP_1}$ и $\overline{PP_2}$ к длине отрезка P_1P_2. □

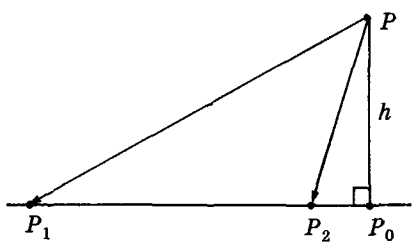


Рис. 6.6

Для луча или отрезка указанный способ нахождения расстояния нужно слегка подкорректировать. Точка P_0 (рис. 6.6) принадлежит лучу P_1P_2 в том и только том случае, когда скалярное произведение $(\overline{P_1P_2}, \overline{P_1P}) \geq 0$. Для отрезка P_1P_2, конечно, нужно еще и выполнение условия $(\overline{P_2P_1}, \overline{P_2P}) \geq 0$. Тогда применима формула расстояния от точки до прямой. В противном случае расстояние от точки до луча (отрезка) будет равно расстоянию от точки до начала луча (до ближайшего конца отрезка).

6.3.3. Взаимное расположение прямых, отрезков, лучей

Данные задачи очень часто встречаются в компьютерной графике. В частности, при программировании компьютерных игр. Например, если требуется проанализировать, насколько удачно игрок произвел выстрел из того или иного оружия.

Легко выяснить, пересекаются ли две прямые или параллельны. Напомним еще раз, что условие коллинеарности двух векторов — это равенство нулю их косого произведения. Если прямые заданы уравнениями $a_1x + b_1y + c_1 = 0$ и $a_2x + b_2y + c_2 = 0$, то удобно перейти к их нормалям $\boldsymbol{n}_1 = (a_1, b_1)$ и $\boldsymbol{n}_2 = (a_2, b_2)$. Тогда условие коллинеарности нормалей (а значит, и параллельности прямых): $[\boldsymbol{n}_1, \boldsymbol{n}_2] = a_1b_2 - a_2b_1 = 0$. Если прямые заданы парами точек, то таким же способом проверяется коллинеарность направляющих векторов. Проверка наличия пересечения прямой и отрезка производится путем анализа взаимного расположения концов отрезка относительно прямой, как это было показано в п. 6.3.2.

Пусть прямая $a_1x + b_1y + c_1 = 0$ и отрезок пересекаются в одной точке. Найдем ее, предварительно выписав уравнение прямой $a_2x + b_2y + c_2 = 0$, проходящей через концы отрезка $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$. Для этого достаточно решить систему двух линейных уравнений, каждое из которых представляет собой уравнение соответствующей прямой относительно x и y:

$$\begin{aligned} x &= (b_1c_2 - b_2c_1)/(a_1b_2 - a_2b_1); \\ y &= (a_2c_1 - a_1c_2)/(a_1b_2 - a_2b_1). \end{aligned} \qquad (6.12)$$

Вопрос. *Как проверить наличие пересечения двух отрезков?*

Ответ. Проверить наличие пересечения двух отрезков (а в компьютерной графике нас в основном интересует лишь сам факт пересечения) несложно опять же с использованием косого произведения. Пусть первый отрезок задан точками P_1 и P_2, а второй — P_3 и P_4. Обозначим x_{max1} и x_{min1} — максимальную и минимальную из первых координат первого отрезка, x_{max2} и x_{min2} — то же для второго отрезка. Для второй координаты аналогично имеем: y_{max1}, y_{min1}, y_{max2} и y_{min2}.

Упомянутые отрезки пересекаются тогда, когда одновременно выполняются следующие три условия:

1) пересекаются ограничивающие их прямоугольники, т. е. $x_{max1} \geq x_{min2}$, $x_{max2} \geq x_{min1}$, $y_{max1} \geq y_{min2}$ и $y_{max2} \geq y_{min1}$;
2) косые произведения $[\overline{P_1P_3}, \overline{P_1P_2}]$ и $[\overline{P_1P_4}, \overline{P_1P_2}]$ имеют разные знаки, точнее $[\overline{P_1P_3}, \overline{P_1P_2}] \cdot [\overline{P_1P_4}, \overline{P_1P_2}] \leq 0$;
3) $[\overline{P_3P_1}, \overline{P_3P_4}] \cdot [\overline{P_3P_2}, \overline{P_3P_4}] \leq 0$.

Последние два условия означают, что концы одного отрезка лежат по разные стороны от прямой, которой принадлежит другой отрезок (см. п. 6.3.2). А первое условие исключает из специального рассмотрения случай равенства нулю всех четырех косых произведений, при котором отрезки лежат на одной прямой и могут как пересекаться, так и нет. □

Если же факт наличия пересечения нами установлен, то для отрезков, находящихся на пересекающихся прямых, точка пересечения ищется так же, как и в предыдущей задаче. Для отрезков одной прямой их пересечение (точка или отрезок) ищется путем подсчета значения нескольких скалярных произведений.

Для проверки наличия пересечения двух лучей P_1P_2 и P_3P_4 следует изучить взаимное расположение соответствующих прямых. Равенство нулю косого произведения $[\overline{P_1P_2}, \overline{P_3P_4}]$ означает принадлежность лучей параллельным прямым. Если эти прямые различны, то векторы $\overline{P_1P_3}$ и $\overline{P_1P_2}$ неколлинеарны и, значит, косое произведение $[\overline{P_1P_3}, \overline{P_1P_2}]$ отлично от нуля. В этом случае лучи не пересекаются. Когда лучи лежат на одной прямой, с помощью знака скалярного произведения $(\overline{P_1P_2}, \overline{P_3P_4})$ можно понять, в одну или в разные стороны они направлены. В первом случае скалярное произведение будет положительным, а во втором — отрицательным. Чтобы определить, какой из двух сонаправленных лучей является их пересечением, можно подсчитать значение скалярного произведения $(\overline{P_1P_2}, \overline{P_1P_3})$. Если оно больше нуля, то пересечением является луч P_3P_4, в противном случае — луч P_1P_2. В случае противоположной направленности лучей их пересечение — либо отрезок P_1P_3, и тогда начало любого из двух лучей лежит внутри дру-

гого луча: $(\overline{P_1P_2}, \overline{P_1P_3}) > 0$, либо одна точка $P_1 = P_3$: $(\overline{P_1P_2}, \overline{P_1P_3}) = 0$, либо оно пусто: $(\overline{P_1P_2}, \overline{P_1P_3}) < 0$.

Наконец, если прямые P_1P_2 и P_3P_4 пересекаются в одной точке M: $([\overline{P_1P_2}, \overline{P_3P_4}] \neq 0)$, то найти эту точку можно так же, как в п. 6.3.2. Затем следует проверить, что M принадлежит каждому из лучей:

$$(\overline{P_1P_2}, \overline{P_1M}) \geq 0 \text{ и } (\overline{P_3P_4}, \overline{P_1M}) \geq 0.$$

6.3.4. Взаимное расположение окружности и прямой

Прямая может пересекать окружность в двух точках, касаться ее или не иметь с окружностью общих точек. Эти случаи легко определяются. Достаточно найти расстояние от центра окружности до данной прямой (по формуле расстояния от точки до прямой, см. п. 6.3.2). Если это расстояние (обозначим его l) меньше радиуса окружности r, прямая пересекает окружность в двух точках, если равно ему, то прямая касается окружности, а если оно больше радиуса, то общих точек нет. В последнем случае нас может интересовать и расстояние от прямой до окружности. Оно равно $l - r$. Более сложной является задача поиска общих точек прямой и окружности.

Задача. *Пусть окружность имеет центр в точке $O(x_0, y_0)$ и радиус r. Требуется найти уравнение касательных к ней, проходящих через точку $P_1(x_1, y_1)$.*

Решение. Здесь возможны три случая. Если $|OP_1| < r$, то P_1 лежит внутри окружности, и касательных, проходящих через нее, не существует. Если $|OP_1| = r$, то P_1 лежит на окружности. Тогда у искомой касательной нам известны точка P_1 и нормаль $\overline{P_1O}$, и ее уравнение легко выписывается (см. п. 6.2.1). Наконец, в случае $OP_1 > r$ точек касания две, и, обозначив одну из них P_2, мы имеем прямоугольный треугольник OP_2P_1 (рис. 6.7).

Мы будем искать координаты $a = x_2 - x_1$ и $b = y_2 - y_1$ вектора $\overline{P_1P_2}$. Длины сторон прямоугольного треугольника OP_2P_1 легко находятся. Выпишем скалярное произведение векторов $\overline{P_1P_2}$ и $\overline{P_1O}$:

$$(\overline{P_1P_2}, \overline{P_1O}) = |P_1P_2| \cdot |P_1O| \cdot \cos\varphi = |P_1P_2|^2.$$

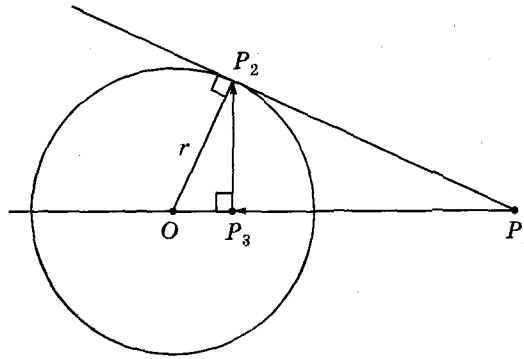


Рис. 6.7

Геометрический смысл косого произведения $[\overline{P_1P_0}, \overline{P_1P_2}]$ — удвоенная площадь треугольника $P_0P_2P_1$, взятая со знаком «плюс» для одной из точек касания и с минусом — для другой:

$$[\overline{P_1O}, \overline{P_1P_2}] = \pm|OP_2|\cdot|P_1P_2|.$$

Записывая эти же произведения в координатах, получим систему линейных уравнений относительно a и b:

$$\begin{cases} (x_0 - x_1)\cdot a + (y_0 - y_1)\cdot b = |P_1P_2|^2, \\ (x_0 - x_1)\cdot b - (y_0 - y_1)\cdot a = \pm|OP_2|\cdot|P_1P_2|. \end{cases} \qquad (6.13)$$

Такую систему решить уже несложно. Далее по точке $P_1(x_1, y_1)$ и направляющему вектору $\overline{P_1P_2} = (a, b)$ выписывается уравнение касательной. Задача решена. Если же нам требуется еще найти и координаты точки касания, то это можно сделать, используя координаты точки P_1 и найденные координаты вектора $\overline{P_1P_2}$.

К решению этой же задачи есть подход, при котором не приходится решать даже систему линейных уравнений. Опустим из вершины P_2 прямого угла высоту P_2P_3 (см. рис. 6.7). Из подобия треугольников P_1P_2O и $P_1P_3P_2$ найдем длины отрезков P_1P_3 и P_3P_2: $|P_1P_3| = |P_1P_2|^2/|OP_1|$; $|P_3P_2| = |P_1P_2|\cdot|OP_2|/|OP_1|$. Теперь последовательно находим координаты вектора $\overline{P_1P_3}$, точки $P_3(x_3, y_3)$ и, наконец, используя известные координаты вектора

$\mathbf{n} = (y_0 - y_1,\ x_1 - x_0)$, перпендикулярного прямой P_1P_3, координаты точки P_2: $\overline{P_1P_3} = \overline{P_1P_3} \cdot |P_1P_3|/|P_1O|$;

$x_3 = x_1 + (\overline{P_1P_3})_x$, $y_3 = y_1 + (\overline{P_1P_3})_y$; $\overline{P_3P_2} = \boldsymbol{n} \cdot |P_3P_2|/|\boldsymbol{n}|$;

$x_2 = x_3 + (\overline{P_3P_2})_x$, $y_2 = y_3 + (\overline{P_3P_2})_y$. □

Пусть теперь прямая и окружность пересекаются в двух точках (рис. 6.8). Координаты этих точек можно найти по следующему алгоритму. Найдем вектор $\mathbf{n}$ нормали к прямой. Отложим в направлении этого вектора вектор $\overline{OA}$ длины l. Вычислим расстояние $|AP_1| = |AP_2| = \sqrt{r^2 - l^2}$. От точки A вдоль прямой отложим в обе стороны векторы длины $|AP_1|$. Их концы дадут нам две искомые точки P_1 и P_2. Каждый из шагов этого алгоритма в отдельности нами уже рассматривался ранее. Заметим только, что на первом шаге необходимо правильно выбрать одно из двух возможных направлений нормали к прямой. Для этого достаточно проверить, что скалярное произведение $(\mathbf{n}, \overline{OM}) \geq 0$, где M — произвольная точка прямой.

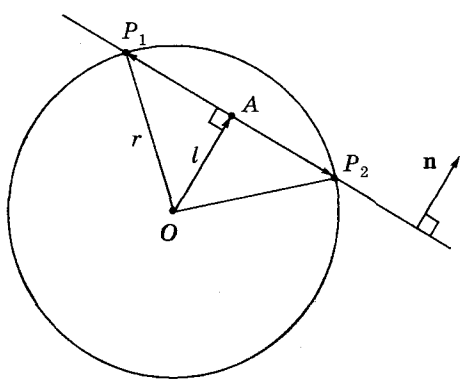


Рис. 6.8

6.3.5. Взаимное расположение двух окружностей

Две различные окружности также могут пересекаться в двух точках, касаться друг друга или не иметь общих точек. В последнем случае либо одна из окружностей располагается внутри другой (назовем такие окружности вложенными), либо каждая из окружностей лежит вне другой.

Проверка наличия пересечения или касания аналогично предыдущей задаче осуществляется путем сравнения расстояния между центрами окружностей (обозначим его l) и их радиусами. Если $l > r_1 + r_2$ или $l < |r_1 - r_2|$, то окружности общих точек не имеют. Второе условие как раз и обозначает вложенность одной окружности в другую. При замене знака любого из этих двух неравенств на равенство мы получим случай касания окружностей (внешнего или внутреннего). Если же $|r_1 - r_2| < l < r_1 + r_2$, то окружности имеют ровно две точки пересечения.

Координаты точки касания окружностей найти очень просто. Ведь центры окружностей $O_1(x_1, y_1)$ и $O_2(x_2, y_2)$ задают прямую, на которой лежит и точка касания $P(x_3, y_3)$. Будем считать, что точка O_1 является центром окружности большего радиуса. Тогда вектор $\overline{O_1P}$ сонаправлен с вектором $\overline{O_1O_2}$. Длины обоих векторов также известны. Искомые координаты равны $(x_1 + (x_2 - x_1)r_1/l,\ y_1 + (y_2 - y_1)r_1/l)$. Проверьте, что если $r_1 < r_2$, то в случае вложенности окружностей полученная формула нуждается в корректировке.

При поиске координат двух точек пересечения окружностей воспользуемся механизмом, уже описанным в п. 6.3.4. Рассмотрим треугольник O_1O_2P (рис. 6.9). В нем нам известны длины всех трех сторон ($r_1 > r_2$ и l). Проведем в треугольнике высоту PP_0. В получившемся прямоугольном треугольнике O_1P_0P неизвестны длины катетов. Найдем O_1P_0, записав теорему косинусов для треугольника O_1O_2P:

$$r_2^2 = r_1^2 + l^2 - 2lr_1 \cos\varphi = r_1^2 + l^2 - 2l|O_1P_0|.$$

Отсюда $|O_1P_0| = (r_1^2 + l^2 - r_2^2) / 2l$. По теореме Пифагора $|P_0P| = \sqrt{r_1^2 - |O_1P_0|^2}$. Теперь последовательно находим:

- вектор $\overline{O_1P_0} = |O_1P_0|/l$;
- точку P_0 по известной точке O_1 и вектору $\overline{O_1P_0}$;
- вектор $\boldsymbol{n} = (y_2 - y_1, x_1 - x_2)$, перпендикулярный O_1O_2;
- вектор $\overline{P_0P} = |P_0P| \cdot \mathbf{n}/|\mathbf{n}|$;
- наконец, точку P по известной точке P_0 и вектору $\overline{P_0P}$.

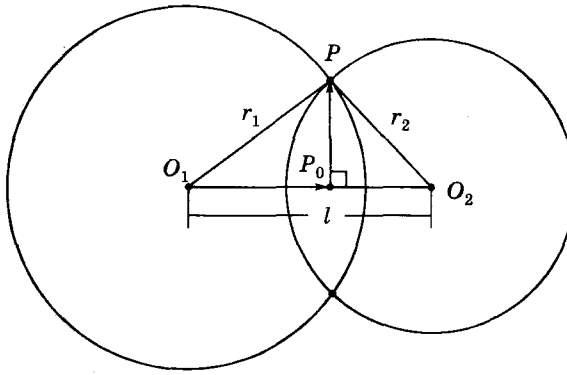


Рис. 6.9

Заменив в последнем действии вектор **n** на противоположный, получим и вторую точку пересечения окружностей.

Вопросы и задания

1. Научитесь определять взаимное расположение двух точек относительно одной прямой.
2. Пусть известно, что два отрезка, заданные своими концами, лежат на одной прямой. По значениям каких скалярных произведений можно определить, пересекаются они или нет?
3. Решите систему линейных уравнений (6.13) для нахождения направляющего вектора касательной к окружности.
4. Пусть векторы $\overline{P_0P_1}$ (x_1, y_1) и $\overline{P_0P_2}$ (x_2, y_2) приложены к точке $P_0(x_0, y_0)$. Найдите уравнение биссектрисы угла $P_1P_0P_2$.
5. На плане местности болото обозначено окружностью, координаты центра которой и радиус известны. Составьте алгоритм нахождения длины кратчайшего пути между двумя точками плоскости, если болото непроходимо (обе точки находятся вне болота).

§ 6.4. Многоугольники

Практически любой объект на графическом экране компьютера представляется с помощью выпуклого или невыпуклого многоугольника. При анализе подобных объектов и программировании их взаимодействия воз-

никает множество различных задач вычислительной геометрии. Рассмотрим наиболее важные из них.

6.4.1. Проверка выпуклости многоугольника

Очевидно, что большинство задач вычислительной геометрии, связанных с многоугольниками, имеют более простое решение в случае их выпуклости. Поэтому сначала научимся проверять для многоугольника именно это свойство.

Выпуклость многоугольника с вершинами P_1, P_2, ..., P_n, перечисленными в порядке его обхода, легко проверить, если вычислить знаки косых произведений $[\overline{P_iP_{i+1}}, \overline{P_{i+1}P_{i+2}}]$, $i = 1, ..., n$ (здесь P_{n+1} есть P_1, а P_{n+2} — P_2, т. е. при программировании удобно добавить после точки P_n две лишние точки, совпадающие с точками P_1 и P_2 соответственно). У выпуклого многоугольника знаки указанных произведений либо все неположительны, либо все неотрицательны (т. е. знаки ненулевых произведений совпадают). Если мы знаем направление обхода, то знак косых произведений для выпуклого многоугольника определен: при обходе по часовой стрелке все косые произведения неположительны, а против часовой стрелки — неотрицательны.

6.4.2. Проверка принадлежности точки внутренней *области многоугольника*

Задача. *Пусть M — некоторая точка плоскости. Требуется определить ее местонахождение относительно замкнутой ломаной, являющейся границей выпуклого многоугольника.*

Решение. Пусть заданные своими координатами вершины многоугольника P_1, P_2, ..., P_n перечислены в порядке его обхода против часовой стрелки. Тогда, если точка M лежит внутри многоугольника, то ориентированный угол между векторами $\overline{P_iM}$ и $\overline{P_iP_{i+1}}$ для любого i отрицателен. Поэтому нам достаточно подсчитать значения косых произведений $[\overline{P_iM}, \overline{P_iP_{i+1}}]$, $i = 1, 2, ..., n$; здесь, как и в п. 6.4.1, точка P_{n+1} совпадает с точкой P_1. Если все полученные при этом значения отрицательны, то точка M — внутренняя. Если же одно из них равно нулю, а все остальные отрицательны, то M принадлежит границе

многоугольника (убедитесь, что просто равенства нулю одного из значений недостаточно). В противном же случае точка M лежит вне нашего многоугольника. □

Рассмотрим теперь *произвольный* многоугольник. Проведем горизонтальный луч из точки M, например, влево. Так как ломаная ограничена, то всегда легко указать на этом луче точку $P(x, y)$, заведомо лежащую вне многоугольника, образованного ломаной. Далее подсчитаем количество пересечений отрезка PM с границей многоугольника, рассмотрев его пересечение с каждым из звеньев ломаной. Если количество таких пересечений равно нулю или четно, то точка M лежит вне многоугольника, в противном случае — внутри него.

На самом деле этот алгоритм нуждается в уточнении. При подсчете количества пересечений отрезка PM со звеньями ломаной для каждого из пересечений важно удостовериться, что отрезок действительно пересек ломаную, а не просто касается нее. При этом возможны следующие особые случаи:

а) одно из звеньев ломаной целиком содержится внутри отрезка PM;
б) звено ломаной касается отрезка PM;
в) точка M лежит на одном из звеньев ломаной.

В последнем случае M принадлежит границе многоугольника и в подсчете общего числа пересечений необходимости нет. Для двух первых случаев поступим следующим образом. В случае а) пересечение будем игнорировать. А в случае б) дополнительно проверим, «нижним» или «верхним» концом звено ломаной касается отрезка PM. Если точкой касания является «нижний» конец звена, то пересечение игнорируется, а если «верхний», то засчитывается. С учетом этого соглашения касание отрезка PM границы многоугольника в одних точках игнорируется (как и требуется с точки зрения алгоритма), а в других точках считается дважды, что, однако, не изменяет четности числа пересечений, а только она важна при поиске ответа на вопрос данной задачи. Если же отрезок действительно пересекает ломаную в ее вершине, то по нашему соглашению число пересечений как раз увеличится на единицу (пересечение с верхним ребром засчитано не будет, а с нижним — будет). Например, на рис. 6.10 количество пересечений для верхней из исследуемых точек

будет равно четырем (касание засчитано дважды), а для нижней точки — трем (касание не учтено, а пересечение в вершине ломаной учтено один раз).

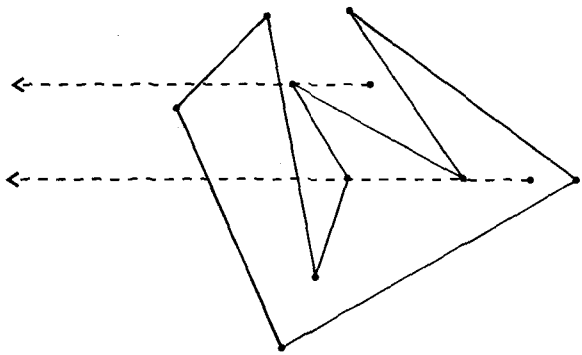


Рис. 6.10

6.4.3. Вычисление площади простого многоугольника

Под простым мы понимаем такой многоугольник, граница которого не имеет самокасаний и самопересечений. Пусть вершины P_1, P_2, ..., P_n простого многоугольника перечислены в порядке обхода его границы. Площадь произвольного простого многоугольника с вершинами P_1, P_2, ..., P_n, перечисленными в порядке его обхода против часовой стрелки, равна ориентированной площади многоугольника (определенной выше для треугольника). В аналитической геометрии доказывается, что последняя представляет собой сумму ориентированных площадей треугольников, образованных векторами $\overrightarrow{OP_i}$ и $\overrightarrow{OP_{i+1}}$, $i = 1, 2, ..., n$, где точка P_{n+1} совпадает с точкой P_1, а O — произвольная точка (рис. 6.11).

Для вычисления обычной площади многоугольника нужно взять модуль ориентированной площади:

$$S = \frac{1}{2}\left|\sum_{i=1}^{n}[\overrightarrow{OP_i}, \overrightarrow{OP_{i+1}}]\right|, \qquad (6.14)$$

где полагается $P_{n+1} = P_1$. В качестве точки O в некоторых задачах бывает разумно выбрать одну из вершин

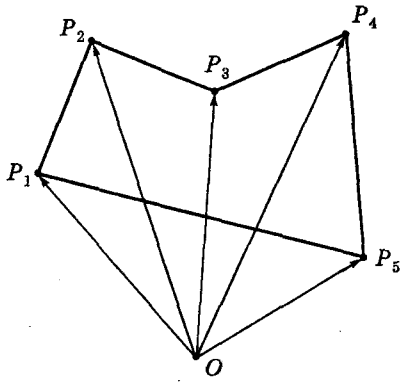


Рис. 6.11

многоугольника. Если же в качестве точки O взять начало координат, то формула (6.14) запишется в другом виде, также удобном для программирования:

$$S = 1/2|(x_1y_2 - x_2y_1) + (x_2y_3 - x_3y_2) + \ldots + (x_ny_1 - x_1y_n)| =$$
$$= 1/2|x_1(y_2 - y_n) + x_2(y_3 - y_1) + \ldots + x_n(y_1 - y_{n-1})|, \quad (6.15)$$

где (x_i, y_i) — координаты точки P_i.

Вопросы и задания

1. Выпишите все формулы, необходимые для реализации решения задачи из п. 6.4.2 проверки принадлежности точки внутренней области произвольного многоугольника.
2. Отрезок, соединяющий две не соседние вершины многоугольника, называется его диагональю. Составьте алгоритм проверки принадлежности диагонали внутренней области соответствующего невыпуклого многоугольника.
3. Выпуклой *оболочкой* некоторого заданного множества точек называется выпуклый многоугольник, все вершины которого являются точками исходного множества. Составьте алгоритм построения выпуклой оболочки заданного конечного множества точек.
4. Составьте алгоритм нахождения расстояния от точки до простого многоугольника на плоскости, если многоугольник задан путем перечисления координат его вершин в порядке их обхода против часовой стрелки.

§ 6.5. Геометрические объекты в пространстве

Несмотря на то что экран компьютера является частью плоскости, и, соответственно, компьютерная графика оперирует двумерными изображениями, наибольший эффект в компьютерных играх, рекламных роликах и различных презентациях достигается при создании на дисплее эффекта трехмерности. Достигается это различными способами, в частности путем проектирования трехмерных объектов на плоскость с учетом их взаимного расположения в пространстве и освещенности. Но прежде чем рассматривать подобные задачи, научимся аналитически описывать объекты трехмерного мира.

6.5.1. Основные формулы

В декартовой системе координат точка в трехмерном пространстве представляется упорядоченной тройкой вещественных чисел (x, y, z). Соответственно и произвольный вектор **a** в трехмерном пространстве можно охарактеризовать тремя координатами (a_x, a_y, a_z). Длина такого вектора равна $\sqrt{a_x^2 + a_y^2 + a_z^2}$.

Будем считать систему координат в пространстве правой, если системы координат xOy, yOz и zOx являются правыми (соответствующее определение для системы координат на плоскости см. в § 6.1).

Определение 6. *Скалярным произведением* двух векторов $\mathbf{a} = (a_x, a_y, a_z)$ и $\mathbf{b} = (b_x, b_y, b_z)$ называется число $(\mathbf{a}, \mathbf{b})$, которое в координатах вычисляется по формуле

$$(\mathbf{a}, \mathbf{b}) = a_x \cdot b_x + a_y \cdot b_y + a_z \cdot b_z. \qquad (6.16)$$

Хотя мы ввели понятие скалярного произведения для правой декартовой системы координат, оно справедливо для любой декартовой системы координат.

Определение 7. Ненулевой вектор, перпендикулярный заданной плоскости, называется *нормалью* к ней.

Определение 8. *Векторным произведением* любых двух векторов **a** и **b** называется вектор $\mathbf{c} = [\mathbf{a}, \mathbf{b}]$, декартовы координаты которого определяются формулой

$$[\mathbf{a}, \mathbf{b}] = (a_y \cdot b_z - a_z \cdot b_y, a_z \cdot b_x - a_x \cdot b_z, a_x \cdot b_y - a_y \cdot b_x). \qquad (6.17)$$

Векторное произведение (вектор **c**) характеризуется следующим образом:

1) вектор **c** ортогонален каждому из векторов **a** и **b**;
2) вектор **c** направлен так, что тройка векторов **a**, **b**, **c** является правой;
3) длина вектора **c** равна произведению длин векторов **a** и **b** на синус угла между ними, т. е.
$|\mathbf{c}| = |[\mathbf{a}, \mathbf{b}]| = |\mathbf{a}| \cdot |\mathbf{b}| \sin\varphi$.

Свойства 1)–3) определяют вектор **c** в пространстве в любой декартовой системе координат правой ориентации однозначно.

Приведем для примера доказательство первого свойства. Чтобы доказать, что вектор $\mathbf{c} = [\mathbf{a}, \mathbf{b}]$ перпендикулярен плоскости, определяемой векторами **a** и **b**, достаточно показать, что скалярные произведения $(\mathbf{a}, [\mathbf{a}, \mathbf{b}])$ и $(\mathbf{b}, [\mathbf{a}, \mathbf{b}])$ равны нулю. Для первого из них мы имеем:

$$(\mathbf{a}, [\mathbf{a}, \mathbf{b}]) = a_x(a_y \cdot b_z - a_z \cdot b_y) + a_y(a_z \cdot b_x - a_x \cdot b_z) + a_z(a_x \cdot b_y - a_y \cdot b_x) = 0.$$

Второе равенство проверяется аналогично.

Взаимное расположение векторов **a** и **b** и векторного произведения показано на рис. 6.12.

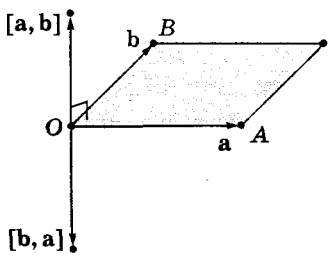


Рис. 6.12

Треугольник *OAB* ориентирован положительно, если на него смотреть в пространстве из конца вектора [**a**, **b**]. Вектор нормали, имеющий такую же длину, но противоположное направление, определяется как [**b**, **a**].

В аналитической геометрии часто используются также следующие свойства векторного произведения:

1) $[\mathbf{a}, \mathbf{b}] = -[\mathbf{b}, \mathbf{a}]$;
2) $[\mathbf{a}_1 + \mathbf{a}_2, \mathbf{b}] = [\mathbf{a}_1, \mathbf{b}] + [\mathbf{a}_2, \mathbf{b}]$;

3) $[(\lambda \mathbf{a}), \mathbf{b}] = \lambda[\mathbf{a}, \mathbf{b}]$;

4) $[\mathbf{a}, \mathbf{a}] = \mathbf{0}$.

Из двух последних свойств следует, что векторное произведение двух любых коллинеарных векторов равно нулевому вектору.

Заметим, что если векторы **a** и **b** лежат в плоскости xOy, то длина вектора их векторного произведения равна модулю их косого произведения:

$$\|[\mathbf{a}, \mathbf{b}]\| = |a_x \cdot b_y - a_y \cdot b_x|.$$

Именно поэтому ранее мы использовали для косого произведения обозначение, принятое для векторного произведения.

6.5.2. Определение пересечения прямой линии и треугольника в пространстве

Пусть нам требуется определить, проходит ли прямая линия в пространстве сквозь треугольник. Решение такой задачи имеет важное значение в трехмерной компьютерной графике, в частности при исследовании проблемы загораживания объектами друг друга. Рассмотрим один из подходов к ее решению.

Определение 9. *Проекцией* называется отражение пространства более высокого порядка на пространство более низкого порядка (в нашем случае — трехмерного на двумерное).

Метод проекции часто используется для замены исходной задачи на задачу «меньшей размерности», т. е. решается задача в пространстве более низкого порядка, где нам уже известны способы ее решения.

Пусть нам требуется определить, проходит ли данная бесконечная прямая линия через заданный треугольник $P_1P_2P_3$. Вначале вычислим координаты точки Q, в которой бесконечная прямая линия пересекает плоскость, в которой лежит треугольник. Затем выполним ортогональное проецирование треугольника $P_1P_2P_3$ и точки Q на плоскость xOy. Для этого достаточно приравнять z-координаты точек P_1, P_2, P_3 и Q нулю. При этом мы получаем треугольник $P_1'P_2'P_3'$ и точку Q'. Задача, эквивалентная исходной, — в двумерном пространстве опреде-

лить, принадлежит ли точка Q' треугольнику $P_1'P_2'P_3'$, — уже была решена нами в 6.4.2.

При проецировании иногда, в результате потери информации, возникает неоднозначная ситуация, когда мы не сможем ответить на поставленный вопрос. Происходит это тогда, когда треугольник проецируется в отрезок на плоскости. Таким образом, чтобы использовать описанный выше алгоритм решения задачи, перед проецированием необходимо выполнить проверку на вырождение. Если вектор нормали треугольника перпендикулярен оси z (а это и есть условие вырождения), то проецирование будем выполнять на плоскость yOz. Если и она окажется вырожденной, то будем использовать проекцию на плоскость zOx. Поскольку вектор нормали не может быть перпендикулярен одновременно всем трем осям, по крайней мере одна из проекций окажется невырожденной.

6.5.3. Вращение точки вокруг заданной прямой в пространстве

Вам наверняка приходилось наблюдать за одним из впечатляющих эффектов трехмерной компьютерной графики — вращением выпуклых или невыпуклых многогранников в пространстве. Насколько сложно реализовать подобное вращение? Если мы научимся вращать требуемым образом одну точку, то, организовав соответствующее движение всех вершин многогранника, несложно будет добиться нужного результата и для многогранника в целом.

Пусть требуется повернуть точку с координатами $A_0(x_0, y_0, z_0)$ относительно оси, заданной единичным вектором $\mathbf{n}(x_n, y_n, z_n)$, проходящим через начало координат, на угол φ. Обозначим через $A_h(x_h, y_h, z_h)$ ортогональную проекцию точки A_0 на ось вращения (рис. 6.13). Точка A_0 и вектор $\mathbf{n}$ определяют плоскость P, перпендикулярную оси вращения. Таким образом, задача сводится к нахождению координат точки A_1, принадлежащей P, такой что $\angle A_0A_hA_1 = \varphi$.

Для решения этой задачи будет построена новая система координат, в которой координаты точки A_1 определяются достаточно просто, а затем найденные координаты будут переведены в исходную систему координат.

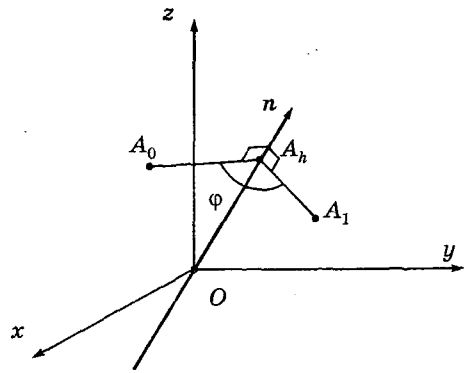


Рис. 6.13

Найдем ортогональную проекцию $A_h(x_h, y_h, z_h)$ точки A_0 на ось вращения. Вектор $\overline{A_hA_0}$ ортогонален вектору **n**, поэтому их скалярное произведение равно 0:

$$x_n(x_0 - x_h) + y_n(y_0 - y_h) + z_n(z_0 - z_h) = 0. \qquad (6.18)$$

Вместе с тем вектор $\overline{OA_h}$ (x_h, y_h, z_h) коллинеарен вектору **n**, поэтому

$$x_h = h \cdot x_n;\ \ y_h = h \cdot y_n;\ \ z_h = h \cdot z_n. \qquad (6.19)$$

Здесь $|h| = |OA_h|$. Подставляя (6.19) в (6.18) и учитывая то, что вектор **n** имеет единичную длину, найдем значение h:

$$h = (x_n \cdot x_0 + y_n \cdot y_0 + z_n \cdot z_0)/(x_n^2 + y_n^2 + z_n^2) =$$
$$= (x_n \cdot x_0 + y_n \cdot y_0 + z_n \cdot z_0).$$

Теперь из (6.19) мы знаем координаты A_h, следовательно, и сам вектор $\overline{A_hA_0}$, и можем вычислить его длину r. Разделив этот вектор на его длину, получим вектор $\mathbf{a}(x_a, y_a, z_a)$ единичной длины.

Введем новую декартову правую систему координат с центром в точке O так:

1. Направим ось z' вдоль оси, задаваемой вектором **n** (базисный вектор, который мы обозначим **k**′, при этом совпадет с вектором **n**).
2. Направим ось x' в направлении вектора $\overline{A_hA_0}$ (обозначим соответствующий базисный вектор **i**′, в нашем случае он совпадает с вектором **a** по построению).

3. Вычислим координаты направляющего вектора оси y' из условия правой ортогональной тройки базиса:

$$\mathbf{j}' = [\mathbf{k}', \mathbf{i}'] = (y_n \cdot z_a - z_n \cdot y_a,\ z_n \cdot x_a - x_n \cdot z_a,\ x_n \cdot y_a - y_n \cdot x_a).$$

То есть координаты базисных векторов новой системы через координаты старой можно выразить следующим образом:

$$\begin{aligned} \mathbf{i}' &= x_a \cdot \mathbf{i} + y_a \cdot \mathbf{j} + z_a \cdot \mathbf{k}, \\ \mathbf{j}' &= (y_n \cdot z_a - z_n \cdot y_a)\mathbf{i} + (z_n \cdot x_a - x_n \cdot z_a)\mathbf{j} + (x_n \cdot y_a - y_n \cdot x_a)\mathbf{k}, \\ \mathbf{k}' &= x_n \cdot \mathbf{i} + y_n \cdot \mathbf{j} + z_n \cdot \mathbf{k}. \end{aligned} \quad (6.20)$$

Координаты искомой точки A_1 в новой системе координат равны ($r\cos\varphi$, $r\sin\varphi$, h) (см. уравнение окружности (6.9)).

$$\overline{OA_1} = x_1 \cdot \mathbf{i} + y_1 \cdot \mathbf{j} + z_1 \cdot \mathbf{k} = r\cos\varphi \cdot \mathbf{i}' + r\sin\varphi \cdot \mathbf{j}' + h \cdot \mathbf{k}'. \quad (6.21)$$

Подставляя в правую часть (6.21) выражения (6.20), получим координаты искомой точки в изначальной системе координат:

$$\begin{aligned} x_1 &= r\cos\varphi\, x_a + r\sin\varphi\,(y_n \cdot z_a - z_n \cdot y_a) + h \cdot x_n; \\ y_1 &= r\cos\varphi\, y_a + r\sin\varphi\,(-x_n \cdot z_a + z_n \cdot x_a) + h \cdot y_n; \\ z_1 &= r\cos\varphi\, z_a + r\sin\varphi\,(x_n \cdot y_a - y_n \cdot x_a) + h \cdot z_n. \end{aligned} \quad (6.22)$$

Вопросы и задания

1. Проверьте, что $(\mathbf{b}, [\mathbf{a}, \mathbf{b}]) = 0$ для любых векторов $\mathbf{a}$ и $\mathbf{b}$.
2. Докажите, что если векторы $\mathbf{a}$ и $\mathbf{b}$ лежат в плоскости xOy, то длина их векторного произведения равна модулю их косого произведения: $|[\mathbf{a}, \mathbf{b}]| = |a_x \cdot b_y - a_y \cdot b_x|$.
3. Модифицируйте алгоритм из п. 6.5.3 так, чтобы он эффективно работал в случае необходимости поворота относительно заданной оси на заданный угол сразу нескольких точек.
4. Составьте алгоритм определения видимости линий, с помощью которых изображается ортогональная проекция выпуклого многогранника на плоскость xOy.
5. На любом известном вам языке программирования напишите программу вращения многогранника относительно некоторой оси.

Заключение

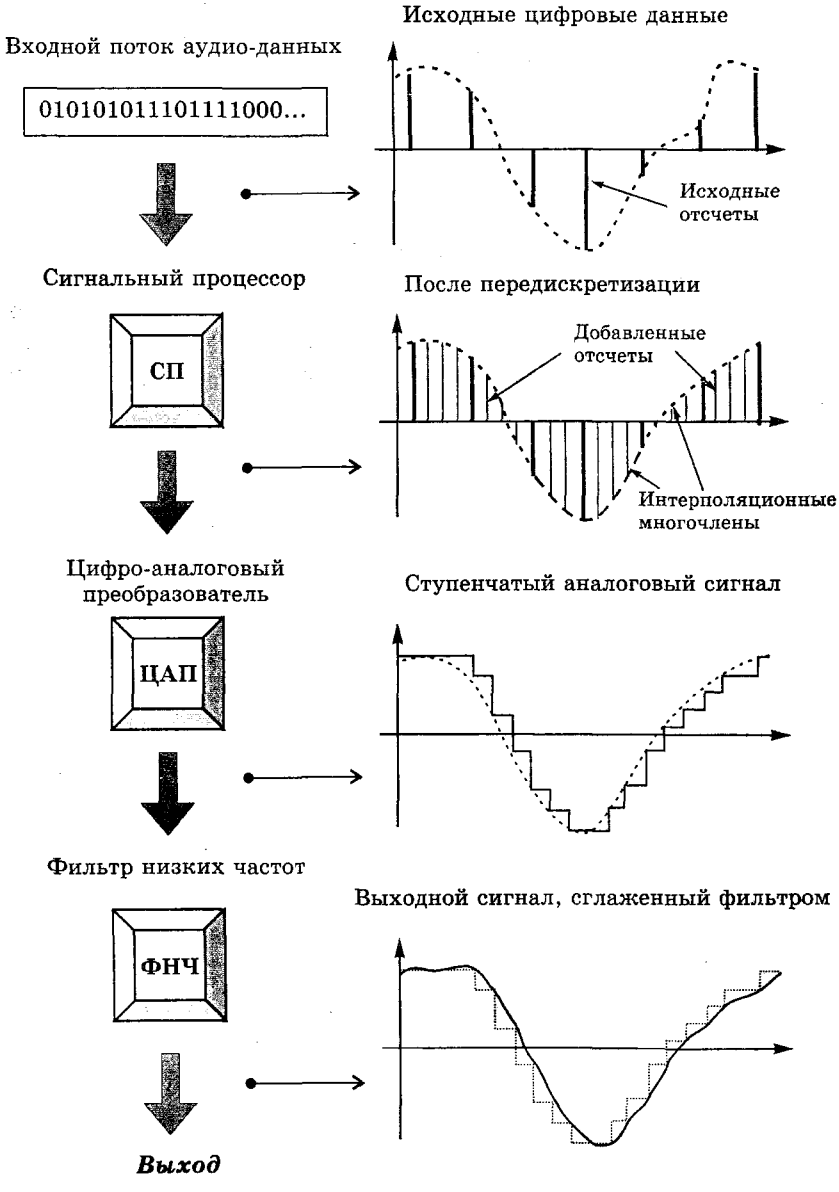
В данной главе вы познакомились с базовыми понятиями вычислительной геометрии, на основе которой построено большинство алгоритмов компьютерной графики. Мы надеемся, что, используя новые для вас понятия косого и векторного произведений двух векторов, а также известного из курса геометрии скалярного произведения, вы научились решать большинство предложенных задач вычислительной геометрии достаточно простым способом.

Используя современные программные средства, вы сможете реализовать многие из рассмотренных алгоритмов, в том числе и в случае решения реальных задач. Научитесь применять предложенные методы решения геометрических задач и в повседневной жизни. Например, задача вычисления площади земельного участка сложной формы уже не должна вызывать у вас затруднений.

Многие алгоритмы вычислительной геометрии и компьютерной графики достаточно сложны и продолжают в настоящее время развиваться и совершенствоваться. Мы ждем открытий в этой области и от вас!

Приложение

Реконструкция аналогового сигнала



Предметный указатель

Победитель конкурса по созданию учебной литературы
нового поколения для средней школы,
проводимого НФПК - Национальным фондом подготовки кадров
и Министерством образования Российской Федерации

Это учебное пособие поможет вам расширить свои теоретические представления о математике в информатике и информатике в математике

Вы получите знания по

- **системам счисления**
- **представлению информации в компьютере**
- **алгебре логики**
- **теории алгоритмов**
- **теории информации**
- **математическим основам компьютерной графики**

Для информационно-технологического и физико-математического профилей

ISBN 5-94774-139-3