МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет информационных технологий
Кафедра систем информатики

В. А. Цикоза, Т. Г. Чурина

# МЕТОДЫ ПРОГРАММИРОВАНИЯ:
## представление и кодирование информации

Учебное пособие

Часть 1

Новосибирск

2003

УДК 519.6
ББК В185

Цикоза В. А., Чурина Т. Г. Методы программирования: представление и кодирование информации. Часть 1: Учебное пособие / Новосиб. гос. ун-т. Новосибирск, 2003. 48 с.

Данное учебное пособие составлено к курсу «Программирование на языке высокого уровня», в котором изучаются традиционные методы программирования и охватывается широкий круг вопросов программирования на ЭВМ. В нем рассмотрены вопросы из первой части курса, касающиеся основ арифметики в позиционных системах счисления, представления и кодирования числовой информации. Кроме того, приведены оценки сложности рассмотренных алгоритмов.

Пособие предназначено для студентов первого курса этого факультета информационных технологий. Кроме того, данное пособие полезно для студентов первого курса механико-математического факультета и слушателей межфакультетского спецкурса «Разработка сложных программ и методы программирования». Может быть рекомендовано студентам первого и второго курсов Высшего колледжа информатики НГУ.

Рецензент
канд. физ.-мат. наук Л. В. Городняя

# Оглавление

# 1. Основы арифметики в позиционных системах счисления

Предметом изучения в этой главе являются соотношения между сущностями и названиями чисел, выражающиеся в способах записи последних и основных арифметических алгоритмах.

## 1.1. Число: значение и обозначение. Системы счисления

Сущность, или *значение*, выражаемое числом, — это абстрактное понятие, по-разному определяемое в различных *аксиоматических системах* математики. Так, в системах счета значение числа выражает меру количественного соотношения (в целых частях или долях) измеряемого объекта к некоторому эталону — единице.

Название, или *обозначение*, числа — это внешняя форма числа, необходимая для его изображения в сообщениях с использованием заданных *знаковых систем* (например, в речи или письме).

В разных языках и знаковых системах, используемых людьми, одно и то же числовое значение может иметь разные названия (обозначения). Значение же числа инвариантно, т. е. не зависит от обозначения — как свойства любого объекта и смысл любого понятия не зависят от их названий.

**Пример.** Одно и то же числовое значение «двенадцать единиц» имеет в разных языках такие обозначения:

- «12»;
- «$XII$»;
- «$twelve$»;
- «$1100_{(2)}$»;
- «$-\dagger =$».

Поскольку чисел бесконечно много, необходима система правил, позволяющая конструировать названия (обозначения) чисел некоторым регулярным способом. Такие системы правил называются *системами счисления* (далее для краткости — с. с.). Название (запись) числа образуется как цепочка названий (знаков) единиц из небольшого выделенного

набора – подобно тому, как груз на чашечных весах уравновешивается гирьками-разновесами заданного ассортимента. В математической символике знаки этих единиц называются *цифрами*.

В дальнейшем при рассмотрении различных с. с. нам придется различать запись обозначения и запись значения (т. е. еще одно обозначение) числа. Проблема в том, что «естественным» названием значения для нас является название десятичного представления числа на русском языке или в традиционной записи арабскими цифрами. Поэтому, чтобы избежать двусмысленности, запись числа будет даваться в кавычках, а без кавычек будут записываться выражения и формулы, вычисляющие числовые значения.

Все арифметические операции с числами определены над значениями, поэтому значения их результатов инвариантны относительно с. с., в которых заданы аргументы. В алгебраических формулах буквы переменных также представляют значения, поэтому формулы инвариантны относительно с. с. Практические вычислительные алгоритмы формулируются для цифровых представлений чисел; они манипулируют цифрами и выдают результат в цифровом виде; поэтому они являются с. с.-зависимыми.

## 1.2. Позиционные и непозиционные системы счисления

С. с. бывают *позиционными* и *непозиционными*. Непозиционные с. с. исторически возникли первыми и в большинстве своем основаны на простом суммировании «весов» цифр-«разновесов», участвующих в записи числа. Такова, например, римская с. с., в которой вес цифры может браться с плюсом или минусом (что аналогично добавлению гирьки на правую или левую чашу весов) в зависимости от позиции данной цифры по отношению к более «тяжелым». Такие с. с. неудобны тем, что для записи все больших чисел требуется все больше цифр-«разновесов».

Со временем непозиционные с. с. были усовершенствованы и превратились в более удобные позиционные. В позиционной с. с. число цифр-«разновесов» конечно, а вклад каждой цифры в значение числа зависит от «веса» ее позиции (*разряда*) в записи числа (растущего как степенная функция от номера разряда). Алгоритмы арифметических действий в позиционных с. с. более регулярны и просты.

Далее рассматриваются важнейшие свойства класса так называемых $b$-*ичных* позиционных с. с. (к которым относится и общепринятая

десятичная), активно применяемых в кодировании и обработке информации, в том числе в компьютерах. Этот класс замечателен тем, что в нем для любой с. с. все основные алгоритмы, а именно: прямой и обратный перевод «значение — обозначение», перевод из одной с. с. в другую, алгоритмы арифметических действий, — выполняются однообразно с точностью до элементарных вычислений над цифрами. Знание универсальных свойств позволяет перенести известные навыки работы с десятичной с. с. на другие системы, в частности, на двоичную и кратные ей, имеющие важное значение в программировании.

## 1.3. Представление целых чисел в позиционной системе счисления с произвольным основанием

Общие свойства всех $b$-ичных позиционных с. с. определяются параметром $b$ – *основанием с. с.*, которое определяет количество различных цифр, требуемых для записи числа. В с. с. с основанием $b \leq 10$ используются первые $b$ арабских цифр: от 0 до $b-1$. В с. с. с основанием $b > 10$ используются дополнительные знаки: например, в шестнадцатиричной с. с. для обозначения следующих за девяткой цифр используются латинские буквы $A$ (числовое значение 10), $B(11)$, $C(12)$, $D(13)$, $E(14)$, $F(15)$.

Целое число в позиционной с. с. с основанием $b$ (далее для краткости $b$-с. с.) записывается строкой цифр, обозначаемых далее $a_i (0 \leq a_i < b)$, где $i$ — индекс (номер позиции, или разряда) цифры в $b$-ичной записи числа; нумерация разрядов начинается с 0, с конца числа, слева направо:

$$«a_{k-1}a_{k-2}a_{k-3}...a_2a_1a_{0(b)}». \tag{1}$$

Слева к записи числа можно дописывать незначащие нули, не влияющие на его значение. Если индекс разряда самой левой значащей цифры числа есть $k-1$, то запись числа называется *$k$-значной*. Исключение: число 0 не имеет значащих цифр, но записывается одной цифрой. В конце числа нижним индексом указывается основание системы счисления, если оно неясно из контекста.

**Примеры**

- Правильно: $101010101_{(2)}$, $14357_{(8)}$, $7BAD_{(16)}$, $0$, $9281$.
- Неправильно: $123023_{(3)}$ — так как есть цифры, большие или равные 3.

## 1.4. Соотношение записи целого числа со значением

Для $k$-значной записи числа $s =$«$a_{k-1}a_{k-2}...a_1a_0\ _{(b)}$» можно определить функцию *значения* $N(s)$, связывающую запись с числовым значением следующей формулой:

$$N(s) = a_{k-1}b^{k-1} + a_{k-2}b^{k-2} + ... + a_2b^2 + a_1b^1 + a_0b^0 = \sum_{i=1}^{k-1} a_ib^i. \quad (2)$$

В отличие от записи (1) здесь множители $b^i$ явно указывают «веса» разрядов, определяющих вклад каждой цифры в значение. В десятичной с. с. они называются разрядами «единиц», «сотен», «тысяч», ... в двоичной — «единиц», «двоек», «четверок», «восьмерок» и т. д. Чтобы исключить из названий зависимость от $b$, в общем случае $b^i$ называют *единицей* $i$-го разряда $b$-ичного числа. Нетрудно видеть, что цифра $a_i$ указывает количество полных единиц $i$-го разряда, которое остается в числе после вычета всевозможного числа единиц старших разрядов.

Другой вид формулы (2) получается последовательным выносом $b$ за скобки:

$$N(s) = (((...((a_{k-1}b + a_{k-2})b + a_{k-3})b... + a_1)b + a_0, \quad (3)$$

откуда видна закономерность, позволяющая определить $N$ рекуррентной функцией (с регрессией аргумента до 0):

$$\begin{array}{l} N_k = 0; \\ N_{i-1} = N_i \cdot b + a_{i-1}, \text{ где } i = k..1; \\ N(s) = N_0. \end{array} \quad (4)$$

Значение $N_i$ равно количеству полных единиц $i$-го разряда в числе.

Можно определить и обратную к $N(s)$ функцию $S(N, b)$, выдающую запись числа по его значению $N$ и основанию с. с. $b$.

**Примеры**

- $N(«1011_{(2)}») = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 2 + 1 = 11.$
- $N(«1011_{(2)}») = ((1) \cdot 2 + 0) \cdot 2 + 1) \cdot 2 + 1 = 11.$
- $N(«8BAD_{(16)}») = 8 \cdot 16^3 + 11 \cdot 16^2 + 10 \cdot 16^1 + 13 \cdot 16^0 = 32768 + 2816 + 160 + 13 = 35757.$

Имеет место следующая теорема:

**(T1) Любое число однозначно представимо в виде цифр заданной b-с. с.**

Допустим, напротив, что некоторое число $N$ имеет в данной $b$-с. с. две записи:

$$s1 = «a_{k-1}a_{k-2}...a_1a_0\ _{(b)}»$$

и

$$s2 = «c_{k-1}c_{k-2}...c_1c_0\ _{(b)}»$$

(длины выравнены приписыванием незначащих нулей к более короткой записи), и не все $a_i = c_i$. Тогда по формуле (2) $N(s1) - N(s2) = \sum_{i=0}^{k-1}(a_i - c_i)b^i$. Вынося, пока возможно, множитель $b$ за скобки, эту сумму можно представить в виде $(p \cdot b + q)b^m$, где $p$, $q$, $m$ — целые, $m \geq 0$, $0 < |q| = |a_m - c_m| < b$. Поскольку $N(s1) = N(s2)$, то $p \cdot b + q = 0$, следовательно, $q = -p \cdot b$, что невозможно, так как $q$ не делится на $b$. Полученное противоречие доказывает невозможность существования различных представлений числа.

## 1.5. Алгоритмы перевода для целых чисел

Непосредственно из формулы (2) выводится алгоритм вычисления числового значения по степеням числа $b$ (его можно использовать для перевода $b$-ичного числа в 10-с. с.).

**Алгоритм A1:**

**вход:** $b > 0$, $k > 0$ (число цифр), набор $a_i$;

$N := a_0$; $S := b$ (S накапливает степень,
N — значение);

цикл по $i$ от 1 до $k - 1$

$N := N + a_i \cdot S$;

$S := S \cdot b$;

конец_цикла;

**выход:** $N$

Алгоритм $A1$ выполняет $2k-2$ операций умножения и $k-1$ операцию сложения.

Из формулы (4) выводится другой алгоритм (схема Горнера).

**Алгоритм A2:**

**вход:** $b > 0$, $k > 0$ (число цифр), набор $a_i$;

$N := a_{k-1}$;
цикл по $i$ от $k-2$ вниз до 0
$N := N \cdot b + a_i$
конец_цикла;

**выход:** $N$

Алгоритм $A2$ выполняет $k-1$ операцию умножения и $k-1$ операцию сложения. Доказано, что меньшего числа операций в общем случае недостаточно. Оба алгоритма допускают наличие незначащих нулей в старших разрядах числа.

Обращение схемы Горнера дает алгоритм получения по числу его $b$-ичной записи (его можно использовать для перевода числа из 10-с. с. в $b$-с. с.).

**Алгоритм A3:**

**вход:** $N \geq 0$, $b > 0$;
i := 0;
цикл
$a_i := N \ mod \ b$; ($mod$ — остаток от деления нацело)
$N := N \ div \ b$; ($div$ — деление нацело)
i := i+1
пока $N \neq 0$;
k := i;

**выход:** набор $a_i$, $k$ (число значащих цифр)

Алгоритм $A3$ также выполняет минимальное ($k$) число операций деления (замечание: при реализации на машинном уровне частное и остаток можно получить одновременно).

Для перевода вручную по алгоритму $A3$ надо поочередно делить на $b$ «столбиком» сначала $N$, затем частное первого деления, второго и т. д., пока не достигнем 0. Искомая запись числа получается последовательным выписыванием всех остатков справа налево.

Перевод числа из $b_1$-с. с. в $b_2$-с. с. можно выполнить в два приема через десятичную или напрямую: в алгоритмах $A1$ и $A2$ вычисления следует вести в $b_2$-с. с., а в алгоритме $A3$ — в $b_1$-с. с.

**Примеры (алгоритм А3)**

• N = 23, b = 2:

```
N0 = 23
N1 = 11   a0 = 1
N2 = 5    a1 = 1
N3 = 2    a2 = 1
N4 = 1    a3 = 0
N5 = 0    a4 = 1
```

$S(N,b) = «10111_{(2)}»$

• N = 19253, b = 16:

```
N0 = 19253
N1 = 120  a0 = 5
N2 = 75   a1 = 3
N3 = 4    a2 = 11
N4 = 0    a3 = 4
```

$S(N,b) = «4B35_{(16)}»$

• N = 26507, b = 10

```
N0 = 26507
N1 = 2650 a0 = 7
N2 = 265  a1 = 0
N3 = 26   a2 = 5
N4 = 2    a3 = 6
N5 = 0    a4 = 2
```

$S(N,b) = «26507_{(10)}»$

## 1.6. Представление действительных чисел

Представление действительного (вещественного) числа в произвольной $b$-с. с. получается естественным обобщением проведенных рассуждений для целого. Запись вида

$$a_{l-1}a_{l-2}a_{l-3}...a_2a_1a_0.a_{-1}a_{-2}a_{-3}...a_{-i}...{}_{(b)}$$

представляет смешанную $b$-ичную дробь (возможно, бесконечную), в которой $b$-ичная точка разделяет 0-й и $-1$-й разряды (целую и дробную части). Числовое значение этой дроби:

$$N(s) = a_{l-1}b^{l-1} + ... + a_0b^0 + a_{-1}b^{-1} + ... + a_{-i}b^{-i} + ... = \sum_{i=-\infty}^{l-1} a_i \cdot b^i. \quad (5)$$

Если в дробной части числа конечное число знаков $k$, то нижний индекс суммы равен $-k$. Конечную дробную часть числа можно также представить в виде (сравните с формулой (3)):

$$N_f(s) = (a_{-1} + (a_{-2} + (a_{-3} + (...)/b)/b)/b)/b, \quad (6)$$

что приводит к следующему рекуррентному определению:

$$\begin{array}{l} N_{-k-1} = 0; \\ N_{-i} = (a_{-i} + N_{-i+1})/b; \text{ где } i = k, ..., 1; \\ N_f(s) = N_{-1}. \end{array} \quad (7)$$

Дробная часть может быть бесконечной и периодической.

**Примеры**

- $N_f(\text{«}1.101_{(2)}\text{»}) = 1 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} = 1 + 0.5 + 0.125 = 1.625.$
- $N_f(\text{«}0.101_{(2)}\text{»}) = (1 + (0 + (1 + 0)/2)/2)/2 = (1 + (0 + 0.5)/2)/2 = (1 + 0.25)/2 = 0.625.$
- $N_f(\text{«}0.01_{(3)}\text{»}) = 1 \cdot 3^{-2} = 1/9 = 0.(1).$

## 1.7. Алгоритмы перевода для действительных чисел

Действительное число можно переводить в другую систему по частям: отдельно целую часть, отдельно дробную. Первый предлагаемый алгоритм перевода *дробной* части из десятичной с. с. в любую $b$-с. с. основан на том, что целая часть числа $N_f \cdot b$ $(0 \le N_f < 1)$ равна первой цифре дробной части числа $N_f$ в $b$-ичном представлении (это видно из формулы (5)) и вычисляет таким образом все цифры дробной части $N_f$:

**Алгоритм А4:**

**вход:** $N_f$ $(0 \le N_f < 1)$, $b > 0$;
i := –1;
цикл
$a_i := [N_f \cdot b]$; ($[x]$ – взятие целой части числа)
$N_f := N_f \cdot b - a_i$; ($N_f$ остается в том же диапазоне)
i := i–1
пока $N_f \neq 0$;
k := i;

**выход:** набор $a_i$, $k$ (число значащих цифр)

Алгоритм $A4$ аналогичен алгоритму $A3$, но может не завершиться, если данное число не представимо конечной дробью в $b$-с. с. На практике в условии окончания цикла дополнительно проверяют, не достигло ли $i$ некоторого предельного значения (требуемой точности).

Алгоритму $A4$ требуется $k$ умножений (выражение $N_f \cdot b$ можно вычислять в цикле один раз и хранить в промежуточной переменной). Обратный перевод (вычисление значения) дробной части по формуле (5) аналогичен алгоритму $A1$.

**Алгоритм А5:**

**вход:** $b > 0$, $k \ge 0$ (число дробных цифр), набор $a_i$;
$N_f := 0$; $S := 1/b$; ($S$ накапливает степень, $N_f$ — значение)

цикл по $i$ от $-1$ вниз до $-k$
$N_f := N_f + a_i \cdot S$;
$S := S/b$
конец_цикла;
**выход:** $N_f$

Этому алгоритму требуется $2k$ операций умножения и деления, а также $k$ операций сложения, и он не является оптимальным. Применяя формулу (7), получаем алгоритм вычисления $N_f(s)$ по схеме Горнера, аналогичный алгоритму $A2$ и требующий $k$ сложений и $k$ делений.

**Алгоритм А6:**
**вход:** $b > 0$, $k > 0$ (число цифр), набор $a_i$;
$N_f := 0$;
цикл по $i$ от $-k$ до –1
$N_f := (a_i + N_f)/b$
конец_цикла;
**выход:** $N_f$

Перевод конечного действительного числа сводится к переводу целого (без разделения на целую и дробную части), если заметить, что число $N$ в $b$-с. с., имеющее $k$ дробных цифр, при умножении на $b^k$ становится целым (это умножение соответствует сдвигу точки на $k$ позиций вправо). Алгоритм перевода из $b_1$-с. с. в $b_2$-с. с. в этом случае имеет вид

**Алгоритм А7**

- найти целое $N_1 = N \cdot b_1^k$ (умножением или сдвигом точки;
- выполнить для $N_1$ один из алгоритмов $A1, A2, A3$;
- разделить полученный результат на $b_1^k$ в системе $b_2$.

**Примеры**

Алгоритм А4

- $N_f = 0.375$, $b = 2$
  $N_{-1} = 0.375$
  $N_{-2} = 0.75$ $a_{-1} = 0$
  $N_{-3} = 0.5$ $a_{-2} = 1$
  $N_{-4} = 0.0$ $a_{-3} = 1$
  $N_{-5} = 0$
  $S(N_f, b) =$«$0.011_{(2)}$»

- $N_f = 0.1$, $b = 2$
  $N_{-1} = 0.1$
  $N_{-2} = 0.2$ $a_{-1} = 0$
  $N_{-3} = 0.4$ $a_{-2} = 0$
  $N_{-4} = 0.8$ $a_{-3} = 0$
  $N_{-5} = 0.6$ $a_{-4} = 1$
  $N_{-6} = N_{-2}$ $a_{-5} = 1$
  $S(N_f, b) =$«$0.0(0011)_{(2)}$»

Алгоритм А3

- $s =$«101.101(2)»
  $s_1 =$«101101(2)»
  $N(s_1) = 2^3 \cdot N(s)$
  $N(s_1) = 45$ (A1)
  $N(s) = 45/8 = 5.625$

## 1.8. Возможность конечного перевода действительных чисел

Конечность представления данного действительного числа $N$ в заданной $b$-с. с. зависит, как видно, например, из алгоритма $A4$, от того, завершается ли процесс отсечения целой части от произведения очередной дроби на $b$ получением нулевого остатка. Это равносильно тому, будет ли число $N \cdot b^k$ целым для какого-либо натурального $k$.

Очевидно, что иррациональные числа, не представимые в виде отношения двух целых, не могут быть конечно представлены ни в какой с. с. Для рациональных критерием конечности представления служит следующая теорема:

**(Т2) Несократимая дробь $\frac{p}{q}$ конечно представима в с. с. с основанием $b$ в том и только в том случае, когда все числа из разложения $q$ на простые множители входят в такое же разложение $b$ (количество повторений не учитывается).**

Для доказательства достаточно показать, что данное условие равносильно тому, что $\frac{1}{q} = nb^{-k}$ для некоторых целых $n$ и $k$. Существенно также условие несократимости.

**Пример**

Дробь 123/675 конечна в 15-с. с., так как $675 = 3^3 \cdot 5^2, 15 = 3 \cdot 5$ и все сомножители 675 (3 и 5) входят в разложение 15; нетрудно видеть, что $1/675 = 5 \cdot 15^{-3} = 0.005_{(15)}$, поэтому исходная дробь тоже конечна. Та же дробь будет бесконечной в 10-с. с. и 25-с. с., в разложении основания которых нет 3.

Для практического программирования с использованием машинной арифметики важно следствие из этой теоремы: дробь $\frac{1}{10}$ не имеет конечного представления в двоичной с. с. Следовательно, большинство конечных десятичных дробей представимы в двоичном компьютере лишь приближенно и вычисления с ними приводят к появлению погрешности.

## 1.9. Кратные системы счисления

Если основания двух с. с. $b_1$ и $b_2$ связаны соотношением $b_2 = b_1^m$ для некоторого натурального $m$, то перевод числа из одной с. с. в другую можно выполнить проще. Такие с. с. называются *кратными*.

Сгруппируем цифры в $b_1$-записи числа по $m$ от точки влево и вправо (добавив при нехватке цифр нужное количество незначащих нулей):

$$\underbrace{0...a_k}...\underbrace{a_{im+m-1}...a_{im}}...\underbrace{a_{m-1}...a_0}.\underbrace{a_{-m+m-1}...a_{-m+0}}...\underbrace{a_{-jm+m-1}...a_{jm+0}},$$

$$A'_k \quad ... \qquad A_i \qquad ... \qquad A_0 \quad . \qquad A_{-1} \qquad ... \qquad A_{-j}$$

затем также сгруппируем слагаемые в формуле (5) (они содержат множитель $b_1$ в степени, равной индексу цифры), вынесем за скобки из каждой группы $i$ общий множитель ($b_1^{im} = (b_1^m)^i = b_2^i$) и обозначим для каждой группы

$$A_i = a_{im+m-1}b_1^{m-1} + a_{im+m-2}b_1^{m-2} + ... + a_{im+1}b_1^1 + a_{im}b_1^0. \qquad (8)$$

Тогда значение исходного числа может быть представлено в виде $N(s') = A_{k'} \cdot b_2^{k'} + ... + A_i \cdot b_2^i + ... + A_0 \cdot b_2^0 + A_{-1} \cdot b_2^{-1}...A_{-j} \cdot b_2^{-j}...$, что по определению совпадает со значением записи того же числа в $b_2$-с. с. с цифрами $A_i$, если заметить, что $A_i$ действительно могут принимать все значения от 0 до $b_1^m - 1 = b_2 - 1$.

Итак, показано, что перевод целого или дробного числа в кратную систему с большим основанием может быть выполнен без промежуточного вычисления значения (или перевода в 10-с. с.) более простым алгоритмом.

**Алгоритм А8:**

**вход:** $b_1 > 0$, $b_2 = b_1^m$, $b_1$-представление числа;

• разбить число на группы по $m$ цифр, начиная от точки, в обе стороны (если в крайних группах цифр меньше $m$, добавить незначащие нули: в целой части спереди, в дробной сзади);

• заменить каждую группу $b_2$-цифрой по формуле (8) или таблице;

**выход:** $b_2$-представление исходного числа.

Аналогичные рассуждения обосновывают и алгоритм обратного перевода.

**Алгоритм А9:**

**вход:** $b_1 > 0$, $b_2 = b_1^m$, $b_2$-представление числа;

• заменить каждую $b_2$-цифру цепочкой из $m$ $b_1$-цифр по формуле (8) или таблице;

• отбросить незначащие нули слева и справа;

**выход:** $b_1$-представление исходного числа.

**Примеры**

$b_1 = 2$, $b_2 = 16 = 2^4$, $b_3 = 8 = 2^3$

Таблицы перекодировки

$b_1 \to b_2$

$0000_{(2)} = 0_{(16)}$ $0100_{(2)} = 4_{(16)}$ $1000_{(2)} = 8_{(16)}$ $1100_{(2)} = C_{(16)}$

$0001_{(2)} = 1_{(16)}$ $0101_{(2)} = 5_{(16)}$ $1001_{(2)} = 9_{(16)}$ $1101_{(2)} = D_{(16)}$

$0010_{(2)} = 2_{(16)}$ $0110_{(2)} = 6_{(16)}$ $1010_{(2)} = A_{(16)}$ $1110_{(2)} = E_{(16)}$

$0011_{(2)} = 3_{(16)}$ $0111_{(2)} = 7_{(16)}$ $1011_{(2)} = B_{(16)}$ $1111_{(2)} = F_{(16)}$

$b_1 \to b_3$

$000_{(2)} = 0_{(8)}$ $100_{(2)} = (4_8)$

$001_{(2)} = 1_{(8)}$ $101_{(2)} = 5_{(8)}$

$010_{(2)} = 2_{(8)}$ $110_{(2)} = 6_{(8)}$

$011_{(2)} = 3_{(8)}$ $111_{(2)} = 7_{(8)}$

Перевод

$1\ 7\ 2\ 5\ 7\ 5\ 0\ .\ 5\ 2\ 2\ 5\ 5\ 4_{(8)}$

001 111 010 101 111 101 000.101 010 010 101 101 100

$111101010111101000.1010100101011011_{(2)}$

0111 1010 1011 1110 1000.1010 1001 0101 1011

$7\ A\ B\ E\ 8\ .\ A\ 9\ 5\ B_{(16)}$

Кратные с. с. могут использоваться и для ускорения перевода между некратными с. с. Например, для перевода написанного выше двоичного числа в десятичное разумно выполнить сначала повышение основания, скажем, до 16: для перевода числа в 10-с. с. из 16-с. с. требуется вчетверо меньше элементарных операций сложения и умножения (деления), что снижает вероятность ошибки при ручных вычислениях.

## 1.10. Универсальные алгоритмы для арифметических операций

Теперь, когда установлены взаимнооднозначное соответствие между числовым значением и его цифровой записью и правила их взаимного перевода, обратим внимание на то, что все так называемые *численные*

алгоритмы для арифметических операций сложения, вычитания, умножения и деления (в том числе, известные еще с начальной школы вычисления «столбиком») являются *символьными*, потому что оперируют входными, выходными и промежуточными данными как строками символов (знаков). Символьные вычисления являются *формальными* в том смысле, что манипулируют только знаками, не обращаясь к их значениям (смыслу). *Абстрагирование* (отвлечение) от смысла данных различной природы и описание алгоритма в терминах чисто символьных преобразований является одним из основных методов программирования обработки данных произвольной природы. Покажем это на примере алгоритма, позволяющего складывать два целых или действительных числа «столбиком» в десятичной системе.

**Алгоритм А10:**

**вход**: две строки цифр, представляющие слагаемые; позиции

• выравнивание: расположить слагаемые одно под другим в произвольном порядке так, чтобы разряды с одинаковым весом находились друг под другом; если какое-то число короче других слева или справа, дополнить его нулями;

• начальные установки: обнулить цифру переноса в следующий разряд; установить результат равным пустой строке;

• *цикл* по текущему разряду от младшего до старшего: определить сумму переноса и цифр в столбце текущего разряда чисел; младшую цифру суммы записать в текущий разряд результата, старшую — в перенос; *конец_ цикла;*

• окончание: если перенос не равен 0, дописать перенос в начало результата;

**выход**: строка, представляющая результат.

Единственное место в этом алгоритме, где присутствует обращение к значениям цифровых символов, — это поразрядное сложение в цикле. Действительно, из одного лишь вида знаков «2» и «3» нельзя извлечь информацию, что результатом их сложения будет знак «5». Эти сведения можно, задать, например, двумя таблицами сложения: в одной для каждой пары цифр записать младшую цифру результата, в другой — цифру переноса («0» или «1»); исчерпав таким образом все немногочисленные случаи, можно заменить операцию сложения значений операцией выборки знака из таблицы. (Заучивая в детстве таблицу умножения, мы поступали также.) Чтобы учесть сложение с переносом, можно завести две пары таблиц или записать в каждую клетку по две цифры.

Теперь первое действие в цикле можно заменить таким: определить из таблиц сложения по переносу и двум цифрам в текущем разряде слагаемых цифры результата и переноса, и из алгоритма исчезнет всякое обращение к значениям цифр.

Алгоритм $A10$ замечателен тем, что применим к произвольной позиционной с. с. при соответствующей замене таблиц сложения. Нетрудно обобщить алгоритм $A10$ для одновременного сложения нескольких чисел, а также аналогичными рассуждениями показать, что алгоритмы вычисления «столбиком» для вычитания, умножения и деления универсально применимы к произвольной с. с. при замене соответствующих таблиц.

**Примеры**

Ниже приведена таблица для троичной с. с. (в скобках указан результат операции, если был сделан перенос):

сложение(с переносом)

| + | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0(1) | 1(2) | 2(0) |
| 1 | 1(2) | 2(0) | 0(1) |
| 2 | 2(0) | 0(1) | 1(2) |

перенос сложения

| ++ | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0(1) |
| 1 | 0 | 0(1) | 1 |
| 2 | 0(1) | 1 | 1 |

умножение

| * | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 11 |

```
    12012        2010001 | 12012
*     102      -  12012 |--------
 ---------     ---------    102
   101101        101101
  +12012       -  101101
 ----------    ---------
   2010001            0
```

## 1.11. Особенности умножения и деления на основание системы счисления

В $b$-с. с. число $b$ всегда имеет представление «$10_{(b)}$». Умножение на $b$ сводится просто к дописыванию 0 справа к целому числу или (что то же), сдвигом $b$-ичной точки на один разряд вправо. Обратно: деление на $b$ равносильно сдвигу точки на один разряд влево или отбрасыванию младшей цифры целого числа — при делении нацело.

Аналогично число $b^k$ всегда представляется единицей с $k$ нулями, а умножение (деление) на $b^k$ сводится к сдвигу точки на $k$ позиций вправо (влево). Остатком от деления целого числа нацело на $b^k$ является число, составленное из $k$ младших цифр. Добавление $k$ нулей справа и

отбрасывание $k$ младших цифр можно рассматривать как две новые операции *арифметического сдвига* на $k$ позиций.

## 1.12. Особенности двоичной арифметики

Двоичная система примечательна среди прочих позиционных с. с. использованием минимального количества цифр — двух, причем особенных — 0 и 1. При сохранении всех общих свойств позиционных с. с. двоичная имеет ряд интересных особенностей.

Таблицы сложения, переноса при сложении и умножения в двоичной с. с. имеют следующий вид:

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

| ++ | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

| * | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Если сопоставить нулю логическую «ложь», а единице — «истину», то таблица сложения совпадет с таблицей значений для логической операции «исключающее или» (которую поэтому еще называют «сложением по модулю 2»), а таблицы умножения и переноса при сложении — с операцией «и». На этом совпадении традиционно основана схемная реализация в компьютерах поразрядной двоичной арифметики с помощью примитивных логических элементов (вентилей). Другая аналогия — «минимаксная»: нетрудно видеть, что $ab = min(a, b)$, а $a + b = min(a, b) + max(a, b)$.

Наконец, так как умножение любого числа на 0 или 1 тривиально, умножение «столбиком» многозначных чисел в двоичной с. с. реализуется только с помощью операций сложения и сдвига.

**Пример**

```
      1001001
    *   10101
    -------
    1001001
+ 1001001
1001001
------------
10111111101
```

## 1.13. Сложность арифметических алгоритмов

Затраты памяти на хранение чисел и времени на выполнение операций с ними зависят, очевидно, от длины записи числа в цифрах рабочей системы счисления. Для заданной $b$-с. с. следующие величины: $k_N$ — длина записи (натурального) числа $N$, $N_k$ — максимальное натуральное число, записываемое $k$ цифрами, связаны соотношениями:

$$\begin{array}{l} k_N = [log_b N] + 1, \quad \text{где } [x] \ - \text{наибольшее целое, не превышающее } x; \\ N_k = b^k - 1. \end{array} \tag{9}$$

Верхние оценки для размера результата арифметической операции над парой целых чисел $N_1$ и $N_2$ (пусть $N_1 \geq N_2$) таковы: для сложения и вычитания — $k_{N_1} + 1$, для умножения — $k_{N_1} + k_{N_2}$, для деления — $k_{N_1}$ (так как $N_2 \geq 1$).

Для определения времени исполнения алгоритма требуется точное знание времени исполнения каждой элементарной операции. Однако сейчас нас интересует только оценка порядка роста этого времени, как функция $T(k)$, для чего достаточно считать длительность каждой операции или фрагмента алгоритма, не зависящих от $k$, некоторой известной константой. Например, время выполнения элементарных поразрядных операций сложения, умножения и пр. при выборе по таблицам явно не зависит от длины исходных чисел.

Алгоритмы сложения содержат один проход по всем разрядам числа, причем каждый разряд обрабатывается не более одного раза. Поэтому время работы алгоритма сложения линейно по $k$: $T_{\text{слож}}(k) \sim k$. Алгоритмы умножения и деления выполняют сложение и вычитание несколько раз (не более, чем $k$), со сдвигом на одну позицию. Так как время сложения линейно, время умножения и деления квадратично по $k$: $T_{\text{умн}} \sim k^2$, $T_{\text{дел}}(k) \sim k^2$.

В системах команд компьютеров есть команды типа сложения и умножения, которые работают не с отдельными двоичными цифрами (битами), а с группами цифр сразу (байтами, словами); они обычно рассматриваются как элементарные. Проведенные выше оценки сохраняют свою силу, если заменить базовую с. с. кратной ей (со степенью кратности равной длине слова). Обычно, если речь не идет о реализации специальных алгоритмов неэлементарных вычислений или многоразрядной арифметики (для чисел, занимающих несколько машинных слов), то длину чисел и время выполнения над ними традиционных арифметических действий считают постоянными.

# 2. Представление числовой информации в ЭВМ

До сих пор в рассматриваемых представлениях, формулах и алгоритмах на длину записи числа не накладывалось никаких ограничений: количество цифр слева и справа от точки могло быть сколь угодно большим (но конечным, иначе формулы и алгоритмы были бы невычислимыми). При вычислениях «на бумажке» не возникает проблем с увеличением разрядности чисел: нужные цифры всегда можно дописать. В машинной памяти на запись числа накладываются ограничения по длине, из-за чего при вычислениях могут возникать такие ситуации, как переполнение и потеря точности. Особенности *машинной арифметики*, рассматриваемой в этом разделе, связаны со свойствами различных форматов кодирования чисел в разрядной сетке и требуют правильной организации процесса вычислений с учетом возможных ошибок и минимизации погрешности.

Как и прежде, если не оговорено иное, все результаты верны для произвольной с. с. (без уточнения основания), хотя в первую очередь подразумеваются двоичная с. с., кратные ей и десятичная, представляющие практический интерес.

## 2.1. Ограничение представления чисел разрядной сеткой

Память компьютера, отводимую для хранения числа или другого элемента данных в числовом коде, удобно описать моделью *разрядной сетки* — массива (линейной таблицы) $b$-ичных разрядов, имеющего постоянные *размер* $k$ и *формат*, определяющий смысл отдельных разрядов. Все позиции (разряды) разрядной сетки должны быть заполнены какими-то допустимыми цифрами, т. е. память не может пустовать.

Размер разрядной сетки обычно равен или кратен размеру минимальной адресуемой ячейки памяти (машинного слова) и размеру минимальной порции данных, обрабатываемой одной командой процессора. Эти константы определяются архитектурой процессора и количеством разрядов, которые могут передаваться одновременно от процессора к памяти или обратно по системной магистрали (шине) данных. В большинстве компьютеров эти единицы кратны 1 байту (8 двоичным разрядам).

Формат разрядной сетки и правила выполнения арифметических операций определяются архитектурой процессора. Современные процессоры умеют работать с разрядными сетками, различающимися по размеру и формату (например, для действительных и целых чисел, со знаком и без, с одинарной и двойной точностью). В языках программирования манипулирование числами разного формата реализуется через различные *числовые типы данных*.

## 2.2. Представление целых без знака: арифметика «по модулю»

Модель *беззнаковых целых (unsigned integer)* описывает представление неотрицательного целочисленного диапазона $[0...N_{max}]$ в сетке разрядности $k$. В записи числа допускаются только $k$ разрядов целой части; цифры, возникающие в результате операций в прочих разрядах, отсекаются. Более короткие числа дописываются нулями слева.

Разряды числа нумеруются справа налево: самый правый, 0-й разряд, называется *младшим*, самый левый, $k-1$-й, — *старшим*.

**Примеры**

5 $k = 6$ 0

| 1 | 0 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|

101101

4 $k = 5$ 0

| 0 | 0 | 0 | 1 | 1 |
|---|---|---|---|---|

11

6 $k = 7$ 0

| 1 | 1 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|

~~1101111~~01111011 (усечение)

Как следует из формулы (9), максимальное представимое в $b$-ичной $k$-разрядной сетке число $N_{max}(k, b)$ равно $b^k - 1$ и записывается $k$ цифрами «$b-1$». Прибавление к этому числу единицы приводит к обнулению всех значащих разрядов и переносу 1 в $k$-й разряд, который усекается. Поэтому «следующим» числом за максимальным будет снова 0.

Вспомним, что выборка $k$ младших цифр $b$-ичного числа $N$ равносильна взятию остатка $Q$ от деления $N$ на $b^k$. Говорят, что $N$ *сравнимо* с $Q$ *по модулю* $b^k$, что записывается как $N = Q(mod\, b^k)$ (термин «модуль» здесь никак не связан с одноименной операцией отбрасывания знака). Множество чисел, сравнимых с $Q$ по модулю $b^k$, — это все числа вида

$Q+n\cdot b^k$, где $n$ — произвольное целое. Все они имеют в $k$-разрядной сетке одинаковое представление. Поэтому взятие остатка по модулю равносильно приведению исходного числа в интервал $[0...b^k-1]$ прибавлением или вычитанием величины модуля некоторое число раз. Отрицательные числа, которых нет в беззнаковой арифметике, приводятся этой операцией к положительным!

**Примеры**

- $153 = 1(mod\ 8)$
- $-7 = 9(mod\ 16)$
- $7 = 7(mod\ 16)$
- $7 + 3 = 2(mod\ 8)$
- $5 \cdot 5 = 9(mod\ 16)$
- $N_{max}(8, 2) = 2^8 - 1 = 255(«11111111_{(2)}»)$
- $N_{max}(4, 16) = 16^4 - 1 = 65535(«FFFF_{(16)}»)$

Беззнаковое представление используется для кодирования адресов ячеек памяти, счетчиков, индексов массивов, элементов перечисления и другой информации, кодируемой числами натурального ряда.

## 2.3. Представление целых со знаком: дополнительный код

Модель *знаковых целых (signed integer)* описывает представление целочисленного диапазона $[N_{min}...N_{max}]$ в сетке разрядности $k$. Старший (левый) разряд кодирует знак числа (0 — плюс, 1 — минус). В остальных $k-1$ разрядах хранятся младшие значащие цифры числа (разряды с $k-2$ до 0), остальные отсекаются, как и при беззнаковом кодировании. Очевидно, что при таком кодировании $N_{min} = -N_{max}$.

Хранение числа в виде знака и модуля на практике оказывается неудобным: ноль имеет двойное представление (со знаком и без), из-за чего представимый диапазон сужается на единицу, кроме того, по-разному реализуются операции сложения и вычитания. Поэтому на практике применяют другой способ кодирования — *дополнительный код*.

Дополнительный код позволяет кодировать $b^k$ различными наборами цифр целочисленный интервал $[-b^{k-1}... + b^{k-1} - 1]$. При этом числа из положительной половины этого интервала имеют те же коды, как

при беззнаковом кодировании (в старшем разряде 0 и абсолютная величина в остальных разрядах), а отрицательная половина отображается на вторую половину беззнакового интервала $[b^{k-1}...b^k - 1]$ по правилу $-N \to b^k - N$ (такие пары чисел сравнимы по модулю $b^k$).

**Примеры**

Зн 4 $k = 6$ 0

| 0 | 0 | 1 | 1 | 0 | 1 |
|---|---|---|---|---|---|

$+1101$

Зн 3 $k = 5$ 0

| 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|

$-1 = 100000 - 1$

Зн 5 $k = 7$ 0

| 1 | 1 | 1 | 0 | 1 | 1 | 0 |
|---|---|---|---|---|---|---|

$-1010 = 10000000 - 1010$

Следующий алгоритм вычисляет представление $-N$ в дополнительном коде в $b$-ичной разрядной сетке размера $k$.

**Алгоритм А11:**

• заменить в записи числа $N$ каждую цифру ее дополнением до $b-1$, максимальной цифры с. с. (для двоичной системы это равносильно инверсии разрядов: замене нулей единицами и наоборот);

• к полученному числу прибавить единицу;

• взять младшие $k$ знаков результата; при необходимости дополнить нулями слева.

Действия алгоритма $A11$ равносильны вычислению

$$((b^k - 1) - N) + 1 = b^k - N = -N(mod\ b^k).$$

Свойства дополнительного кода (примеры см. ниже):

• алгоритм $A11$ дает правильное представление обратных по знаку чисел — как положительных, так и отрицательных. В частности, его повторное применение к своему результату восстанавливает исходное число $N$;

• для двоичной с. с.: если приписать к числу его знак старшей цифрой и выполнить алгоритм $A11$ над расширенной записью, то правильный знак обратного числа получается автоматически;

• ноль имеет единственное представление: алгоритм $A11$ для ноля выдает снова ноль;

• то же свойство имеет минимальное отрицательное число $-b^{k-1}$, не имеющее положительного «антипода»: его дополнение равно ему самому.

**Примеры**

Пусть b = 2, k = 8: Для числа со знаком $N$ приведены его представление в дополнительном коде и беззнаковая интерпретация того же представления; аналогичные данные приведены для $-N$.

| $N_{\text{зн}}$ | $s(N)$ | $N_{\text{б/зн}}$ | $-N_{\text{б/з}}$ | $s(-N)$ | $-N$ |
|---|---|---|---|---|---|
| • 1 | 0:0000001 | 1 | 255 = 256-1 | 1:1111111 | -1 |
| • -2 | 1:1111110 | 254 | 2 = 256-254 | 0:0000010 | 2 |
| • 25 | 0:0011001 | 25 | 231 = 256-25 | 1:1100111 | -25 |
| • -47 | 1:1010001 | 209 | 47 = 256-209 | 0:0101111 | 47 |
| • 127 | 0:1111111 | 127 | 129 = 256-127 | 1:0000001 | -127 |
| • -128 | 1:0000000 | 128 | 128 = 256-128 | 1:0000000 | -128 |
| • 0 | 0:0000000 | 0 | $0 = 256 (\mathrm{mod}\ 256)$ | 0:0000000 | 0 |

## 2.4. Выполнение арифметических операций в разрядной сетке

Дополнительный код удобен тем, что правильное формирование знакового разряда результата арифметических операций в поразрядных алгоритмах можно осуществить регулярным образом: как если бы он был просто дополнительным старшим разрядом числа.

Во-первых, это позволяет избавиться (как в знаковой, так и в беззнаковой арифметике) от реализации вычитания отдельным алгоритмом: действительно, так как $a - b = a + (-b)$, достаточно перевести вычитаемое в дополнительный код и выполнить сложение.

Во-вторых, оказывается, поразрядное сложение чисел со знаком и без знака можно осуществить единым алгоритмом (например, $A10$): манипуляции с цифрами одинаковы, различается лишь способ интерпретации конечного результата — как знакового или беззнакового числа. (Для умножения и деления это не проходит без дополнительных приведений.)

Следовательно, для реализации операций типа сложения для обеих моделей достаточно одного алгоритма.

При выполнении операций над числами разной разрядности производится так называемое *выравнивание*: расширение формата более короткого числа до более длинного. Для этого его разрядная сетка дополняется слева нужным числом старших разрядов, которые заполняются нулями для беззнаковых и положительных знаковых чисел и единицами для отрицательных знаковых (эта операция также называется *знаковым расширением числа*). Выравнивание чисел обычно выполняется во вспомогательной памяти нужного размера.

**Пример**

Вычислить разность 25-3 для 8-разрядной двоичной сетки.

```
   а). перевод +25 в 2-с.с. со знаком:                    011001
   б). выравнивание: знаковое расширение до 8 разр.:    00011001
   в). перевод +3 в 2-с.с. со знаком:                      011
   г). перевод +3 в дополнит. код (так как это вычитаемое):  101
   д). выравнивание: знаковое расширение -3 до 8 разр.: 11111101
   е). 8-разрядное сложение с переносом(без знака):   1)00010110
   ж). приведение: отсечение старш. разрядов вне сетки: 00010110
результат - положительное знаковое число 22
```

Замечание: перестановка операций перевода в дополнительный код и знакового расширения на шагах `г` и `д` приводит к тому же результату. Однако при этом расширяется всегда положительное число.

Этот пример иллюстрирует способ вычитания беззнаковых чисел 25 и 3, а также сложение беззнаковых чисел 25 и 253 с результатом $22 = 278(mod\ 2^8)$. Таким образом, алгоритм корректен и для беззнаковых чисел.

## 2.5. Перенос и переполнение

При выполнении арифметических операций результат может оказаться вне границ допустимого интервала. Приведение по модулю (путем усечения значащих цифр в старших разрядах) вызывает отклонение полученного результата от истинного. Эта ситуация называется *переполнением разрядной сетки*; она возможна в обоих рассмотренных представлениях и должна контролироваться во избежание ошибок в вычислениях.

*Беззнаковое переполнение (арифметический перенос)*, т. е. выход за границы интервала $[0...b^k - 1]$, при сложении равносильно переносу 1

из старшего $k-1$-го разряда в несуществующий $k$-й (сумма больше $b^k-1$, приведенная сумма оказывается меньше обоих слагаемых, что ненормально). При вычитании (большего числа из меньшего) переполнение равносильно заему 1 из несуществующего $k$-го разряда в старший $k-1$-й (разность отрицательна, приведенная разность оказывается больше уменьшаемого, что тоже ненормально). Процессор контролирует эти ситуации установкой *флажка переноса* (carry flag) в регистре состояния, который можно проверить, чтобы выполнить обработку переноса.

Арифметический перенос не всегда означает ошибку: это нормальное явление при реализации арифметики с многоразрядными числами, хранимыми в нескольких машинных словах. Их сложение выполняется последовательно от младшего слова к старшему, и сигнал переноса свидетельствует лишь о необходимости прибавления дополнительной единицы к следующему слову или вычитания из него. Для этого в системе команд процессора обычно имеются команды установки состояния флажка переноса и команды сложения и вычитания с переносом (заемом).

*Знаковое переполнение*, т. е. выход из интервала $[-b^{k-1}...b^{k-1}-1]$, возникает при сложении положительных чисел в случае переноса 1 в знаковый $k-1$-й разряд результата (приведенная сумма становится отрицательна) или при сложении отрицательных с заемом из знакового разряда (приведенная сумма становится положительна); аналогично — при вычитании. Процессор контролирует знаковое переполнение установкой *флажка переполнения* (overflow flag) в регистре состояния.

Так как операции типа сложения для знаковых и беззнаковых чисел реализуются одной командой, флажки переноса и переполнения формируются одновременно, но для знаковой арифметики несущественно состояние переноса, а для беззнаковой — переполнения.

**Примеры**

а) 1 : 1111100 + 1 : 1110001 = 1)1:1101101, перенос = 1, переполнение = 0

— числа со знаком: (-4) + (-15) = (-19) – перенос не имеет смысла, ошибки нет;

— числа без знака: 252 + 241=(256+)237 – перенос указывает на беззнаковое переполнение.

б) 0 : 1111111 + 0 : 00000010 = 1 : 0000001, перенос = 0, переполнение = 1

— числа со знаком: 127 + 2 = (256) - 127 – знаковое переполнение;

— числа без знака: 127 + 2 = 129 – переполнение не имеет смысла, ошибки нет.

## 2.6. Конечное представление действительных чисел: арифметика погрешностей

Любой действительный интервал содержит бесконечное множество чисел, так как существуют числа с произвольно длинной и даже бесконечной дробной частью. Для представления в конечной памяти применяется дискретизация интервала: разбиение его на достаточно малые подынтервалы, в каждом из которых все числа сводятся к одному и тому же *числу-представителю*, становясь, таким образом, неразличимыми. Ширина сводимого интервала определяет максимальную *погрешность представления* — отклонение значения истинного числа от значения его представления.

Простейший способ разбиения и выбора представлений — усечение дробной части числа $R$ *после* некоторого разряда $-k$. Получаемый представитель $r$ имеет не более $k$ значащих цифр в дробной части и представляет все числа из интервала $[r...r+\varepsilon[$, где $\varepsilon = b^{-k}$ — максимальная абсолютная погрешность, равная расстоянию между представителями, равномерная вдоль всей числовой оси.

Другой способ разбиения: усечение после разряда $-k$ с округлением по $-k-1$-му — приводит к представлению числом $r$ симметричного интервала $[r-\varepsilon...r+\varepsilon[$, где $\varepsilon = b^{-k}/2$, и дает вдвое меньшую оценку максимальной абсолютной погрешности при том же расстоянии между представителями.

Третий способ разбиения: усечение после $k$ *старших значащих* разрядов числа с округлением — приводит к неравномерным по ширине сводимым интервалам, сгущающимся вокруг нуля: число $r$ представляет интервал $[r-\varepsilon_r...r+\varepsilon_r[$, где $\varepsilon_r = b^{-k+p}/2$, $p$ – порядок нормализованного вещественного числа $r$. Погрешность представления здесь *относительная*, так как зависит от абсолютной величины числа.

Погрешности, возникающие при представлении действительных чисел, растут при любых арифметических операциях над ними, как видно из следующих формул:

пусть $|x-x'| \le \varepsilon_1$, $|y-y'| \le \varepsilon_1$, тогда

$$|(x \pm y) - (x' \pm y')| \le \varepsilon_1 + \varepsilon_2,$$

$$|x \cdot y - x' \cdot y'| \le min(|x|,|x'|) \cdot \varepsilon_2 + min(|y|,|y'|) \cdot \varepsilon_1 + \varepsilon_1 \cdot \varepsilon_2.$$

Растущая недостоверность вычислений с действительными числами приводит к необходимости предварительного анализа расчетных формул с оценкой максимальной погрешности, а также специальной орга-

низации вычислений, минимизирующей ее. Например, вычисление выражения $(a + b) \cdot (c + d)$ дает меньшую погрешность при раскрытии скобок, а выражения $(a/b) \cdot (c/d)$ — при перегруппировке в одну дробь.

## 2.7. Представление действительных чисел: формат с фиксированной точкой

Модель действительных чисел с фиксированной точкой аналогична представлению целых чисел со знаком: под хранение целой и дробной части отводится известное число битов, т. е. привязка номеров разрядов числа к разрядам сетки фиксирована и позиция разделяющей двоичной точки известна. Разряды отсутствующих цифр заполняются незначащими нулями слева в целой части и справа в дробной. Отрицательные числа могут храниться в дополнительном коде.

**Пример**

15 $k_{\text{ц}} = 7$ 8 7 $k_{\text{д}} = 8$ 0

| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

$+101100\ .011101111\sout{001111}$ (усечение)

Погрешность представления в данной модели равномерна: $\varepsilon = b^{-k_{\text{д}}}/2$. Представимый диапазон — $[-b^{k_{\text{ц}}} + \varepsilon\ ...\ b^{k_{\text{ц}}} - \varepsilon[$.

Арифметические операции над числами с фиксированной точкой можно выполнять поразрядно, рассматривая их без точки, как знаковые целые. После умножения точка ставится перед $2k_{\text{д}}$-й цифрой результата справа, младшие $k_{\text{д}}$ разрядов отбрасываются, причем по старшему из них выполняется округление младшей цифры результата. Для деления ненулевое делимое сдвигается влево на $p_1$ позиций до появления первой единицы в старшем разряде сетки, а делитель — влево на $p_2$ позиций, до получения максимального числа, которое меньше делимого. Число $p = p_2 - p_1 + 1$ определяет количество значащих цифр перед точкой в результате (или количество ведущих нулей в дробной части, если $p < 0$). Если $p > k_{\text{ц}}$, деление заведомо даст слишком большое частное (переполнение); если $p < -k_{\text{д}} - 1$, результатом будет машинный ноль (антипереполнение). Деление можно выполнять «столбиком» (вычитанием делителя со сдвигом остатка влево) до получения $p + k_{\text{д}} + 1$ цифр результата (последняя цифра используется для округления). Заметим, что при выполнении округления также возможно переполнение.

## 2.8. Представление действительных чисел: формат с плавающей точкой

Модель действительных чисел с плавающей точкой служит для представления чисел с максимальной *относительной* точностью, измеряемой количеством хранимых значащих цифр действительного числа. Для этого используется запись числа в *нормализованной форме*. Для перевода числа $R$ в эту форму точка в его $b$-ичной записи сдвигается в позицию после старшей значащей цифры: получившееся число $m$ называется *мантиссой*, а число позиций $p$, на которое точка сдвинулась влево, — *$b$-ичным порядком* ($p$ отрицательно, если фактически точка переместилась вправо). При этом выполняется равенство

$$R = \pm m \cdot b^p, \quad \text{где} \quad 1 = b^0 \le m < b^1, \quad p - \text{целое}, \tag{10}$$

правая часть которого и является нормализованным представлением $R$.

**Примеры**

$123_{(10)} = 1.23 \cdot 10^2$, $0.00123_{(10)} = 1.23 \cdot 10^{-3}$, $01001.11_{(2)} = 1.00111 \cdot 2^3$.

Так как позиция точки в мантиссе фиксирована (после первой цифры), $m$ однозначно соответствует целому числу $m'$, составленному из тех же цифр. Таким образом, число $R$ может быть представлено парой целых чисел $(m', p)$, как это и делается в модели с плавающей точкой. Для хранения цифр мантиссы и порядка отводится фиксированное число разрядов сетки ($k_m$ и $k_p$), в которых они хранятся, как обычные целые, при этом порядок обычно хранится как знаковое целое в дополнительном коде, а мантисса — как беззнаковое целое. Знак вещественного числа хранится в отдельном разряде.

**Пример**

зн 14 $k_p = 5$ 10 9 $k_m = 10$ 0

| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

$1010.1011011011 = 1.0101011011011 \cdot 2^3$

$m' = 10101011011\,\cancel{01}1$ (усечение), $p = 11_{(2)}$

**Замечание:** в двоичной с. с. старшая значащая цифра мантиссы всегда единица, которую иногда не хранят (как в этом примере), чтобы сэкономить разряд для дополнительной значащей цифры, а добавляют при извлечении значения перед выполнением операций.

Представимый этой моделью интервал чисел имеет верхнюю границу $R_{max} = +m_{max} \cdot b^{p_{max}}$ и нижнюю $R_{min} = -R_{max}$. Минимально

представимое число, отличное от 0: $\varepsilon_1 = m_{min} \cdot b^{p_{min}}$, и относительная погрешность представления чисел в модели: $\varepsilon_2 = b^{-k_m}/2$, — являются важными характеристиками машины, используемыми при анализе вычислительных алгоритмов. Для приведенной двоичной модели $R_{max} = (2 - \varepsilon_2) \cdot 2^{2^{k_p - 1}}$, $\varepsilon_1 = 1 \cdot 2^{-2^{k_p - 1}}$.

Арифметические операции типа сложения с плавающей точкой требуют предварительной *денормализации* операндов (приведения к одному порядку путем сдвига одного из аргументов, обычно с меньшим порядком) и последующей нормализации результата. При мультипликативных операциях отдельно выполняется умножение или деление нормализованных мантисс, отдельно — сложение или вычитание порядков; последующая нормализация результата действий с мантиссами может скорректировать результат действий с порядками на 1. В данном пособии мы не приводим алгоритмы, реализующие арифметические операции с плавающей точкой из-за их громоздкости. В настоящее время почти все компьютерные архитектуры поддерживают модель вещественной арифметики, описанную международным стандартом IEEE 754, что обеспечивает переносимость вычислительных алгоритмов.

# 3. Элементы теории информации и кодирования

В данном пособии мы коснемся самых простых понятий из теории вероятностей, которые нам потребуются для изучения некоторых способов кодирования. Теория вероятностей изучает способы вычисления вероятностей случайных событий.

## 3.1. Определение вероятности и основные правила

Событие, которое может произойти или нет, называют *случайным*. Примерами таких событий являются попадание стрелка в мишень, извлечение дамы пик из колоды карт, выигрыш билета в розыгрыше лотереи и т. д. На основании отдельно взятого случайного события нельзя научно предсказать, например, какие билеты окажутся выигрышными. Но если провести достаточно большую последовательность испытаний, то можно выявить определенные закономерности, позволяющие делать количественные предсказания.

Для математического описания эксперимента со случайными событиями нам потребуется понятие *пространства элементарных событий (исходов)* как множества всех различных событий, возможных при проведении эксперимента; обозначим его $\Omega$. Элементарность исходов понимается в том смысле, что ни один из них не рассматривается как сочетание других событий.

Рассмотрим следующий эксперимент. Будем бросать монету до тех пор, пока не выпадет герб. После этого эксперимент закончим. «Элементарный исход» этого эксперимента можно представить в виде последовательности р, р, р, ..., р, г (где р — решка, г — герб). Таких последовательностей бесконечно много. Следовательно, в данном случае множество $\Omega$ бесконечно.

Рассмотрим другой пример: однократное бросание игральной кости. Будем считать, что возможен только один из 6 исходов, соответствующих падению кости гранями с $1,\ 2, ..., 6$ очками вверх. Все другие исходы, например падение кости на ребро, считаются невозможными. Каждый возможный исход удобно обозначать числом выпавших очков. Тогда пространство элементарных событий $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Формула $\omega \in \Omega$ означает, что элементарное событие $\omega$ является элементом пространства $\Omega$. Многие события естественно описывать множествами, составленными из элементарных исходов. Например, событие, состоящее в появлении четного числа очков, описывается множеством $S = \{2, 4, 6\}$. Формула $S \subseteq \Omega$ означает, что событие $S$ является подмножеством пространства $\Omega$.

Можно проследить соответствие введенных понятий со следующими известными в математике понятиями:

| | | |
|---|---|---|
| Случайная величина | $\longrightarrow$ | переменная |
| Элементарный исход | $\longrightarrow$ | значение переменной |
| Пространство элементарных исходов | $\longrightarrow$ | область значений |
| Событие | $\longrightarrow$ | подмножество области значений |

Определим формально *меру события* $\mu$ как отображение из пространства $\Omega$ в $N$ (множество натуральных чисел), обладающее следующими свойствами:

1) $\mu(\emptyset) = 0$, где $\emptyset$ — пустое множество, т. е. множество, не содержащее ни одного элемента;

2) $S_1 \subseteq S_2 \ \Rightarrow \mu(S_1) \leq \mu(S_2)$, где $S_1 \subseteq \Omega, S_2 \subseteq \Omega$;

3) $\mu(S_1 \cup S_2) = \mu(S_1) + \mu(S_2) - \mu(S_1 \cap S_2)$.

Введем функцию $p(S)$ *вероятности события* как численного выражения возможности события $S$ на заданном пространстве элементарных исходов $\Omega$ следующим образом:

$$p(S) = \frac{\mu(S)}{\mu(\Omega)} = \frac{\text{число желательных исходов}}{\text{число всех возможных исходов}}. \tag{11}$$

Под «желательными» исходами здесь понимаются элементарные исходы, образующие событие $S$. Из определения ясно, что $0 \leq p(S) \leq 1$ для любого $S \subseteq \Omega$, $p(\emptyset) = 0, p(\Omega) = 1$. Событие с вероятностью 1, очевидно, содержит все элементарные исходы и, следовательно, происходит наверняка. Событие с вероятностью 0 не содержит ни одного исхода, следовательно, не происходит никогда.

Вероятность того, что при бросании кости выпадет единица, равна

$$\frac{\mu(\{1\})}{\mu(\{1,2,3,4,5,6\})} = \frac{1}{6}.$$

Вероятность появления четного числа очков равна

$$\frac{\mu(\{2,4,6\})}{\mu(\{1,2,3,4,5,6\})} = \frac{3}{6} = \frac{1}{2}.$$

Паскаль в письмах к Ферма в 1654 г. писал: «Как велика вероятность, что когда я проснусь ночью и посмотрю на часы, то большая стрелка будет стоять между 15 и 20 минутами?» И в этом же письме приводит рассуждения о том, что вероятность того, что стрелка часов будет находиться в этом промежутке, равна $\frac{5}{60} = \frac{1}{12}$.

Теорема о сложении вероятностей.

**(Т3) Если пересечение событий $A$ и $B$ непусто, то справедливо равенство**

$$p(A \cup B) = p(A) + p(B) - p(A \cap B).$$

Эта формула для вероятности следует из аксиомы 3 для меры.

**Пример.** Найдем вероятность того, что вытащенная из полной колоды карта окажется пикой или картинкой (валетом, дамой, королем, тузом). Пусть событию $A$ соответствует извлечение из колоды карт пики, событию $B$ — картинки. Для каждой карты из колоды вероятность вытащить ее равна $\frac{1}{52}$. Число пик в полной колоде равно 13. Следовательно, вероятность события $A$ равна $\frac{13}{52} = \frac{1}{4}$. Число картинок равно 16, следовательно, вероятность события равна $\frac{16}{52} = \frac{4}{13}$. События $A$ и

$B$ имеют непустое пересечение, так как вытащенная карта может оказаться одновременно и пикой и картинкой. Множество $A \cap B$ состоит из четырех элементов, следовательно, $p(A \cap B) = \frac{4}{52} = \frac{1}{13}$. Отсюда

$$p(A \cup B) = p(A) + p(B) - p(A \cap B) = \frac{1}{4} + \frac{4}{13} - \frac{1}{13} = \frac{25}{52}.$$

**Следствие.** Если событие $S$ разделено на несколько несовместных (взаимоисключающих) событий, то его вероятность равна сумме вероятностей этих событий. Например, если $S = A \cup B, \quad$ где $A \cap B = \emptyset$, то

$$p(S) = p(A) + p(B).$$

**Пример.** Рассмотрим опыт. Найдем вероятность того, что вытащенная из полной колоды карта окажется пикой или червой. Пусть событию $A$ соответствует извлечение из колоды карт пики, событию $B$ — червы. Вероятность вытащить любую карту из колоды равна $\frac{1}{52}$. Число пик в полной колоде равно 13. Следовательно, вероятность события $A$ равна $\frac{13}{52} = \frac{1}{4}$. Вероятность события $B$ также равна $\frac{13}{52} = \frac{1}{4}$. Так как события $A$ и $B$ *несовместны*, т. е. никакая карта не может быть и пикой и червой одновременно, то искомая вероятность равна $\frac{1}{4} + \frac{1}{4} = \frac{1}{2}$.

Рассмотрим теперь серию экспериментов, в которой некоторая случайная величина наблюдается последовательно несколько раз. Последовательные события называются *независимыми*, если наступление каждого из них не связано ни с каким из других. Например, исходы при бросании кости являются независимыми событиями, а последовательные вытягивания карт из одной и той же колоды без возврата — нет (вероятность вытянуть даму пик зависит, например, от того, была ли она уже вытянута в предыдущей серии).

Теорема об умножении вероятностей.

**(Т4) Вероятность того, что независимые события $S_1$, $S_2$ произойдут в одной серии испытаний, равна произведению вероятностей событий $S_1$ и $S_2$.**

**Пример.** Будем подбрасывать две монеты одновременно. Какова вероятность того, что обе монеты упадут гербом вверх? По теореме об умножении эта вероятность равна $\frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

## 3.2. Информационная модель Шеннона

К. Шеннон в 1936 г. предложил к рассмотрению следующую модель системы передачи информации. Имеются *источник* (*кодер*) и *приемник*

(*декодер*). Источник и приемник связаны между собой *каналом*, по которому передаются сообщения. Канал не искажает и не теряет сообщений.

Пусть в области источника происходит наблюдение за некоторой случайной величиной. Приемник может иметь некоторое *априорное* представление о множестве $S_{\text{до}}$ возможных исходов этой величины до того, как произошло наблюдение. Например, в худшем случае, когда ничего не известно заранее, $S_{\text{до}}$ принимается за все пространство возможных исходов $\Omega$. Пусть теперь источник передает приемнику сообщение о произошедшем наблюдении, после получения которого множество предположительных исходов у приемника, вообще говоря, сужается до $S_{\text{после}}$ (в лучшем случае до единственного наблюдаемого исхода). Это представление будем называть *апостериорным*. (Будем разумно предполагать, что сообщение не может вносить хаос в имеющиеся знания приемника, т. е. расширять объем существующих предположений.) Будем говорить, что источник передал приемнику некоторую *информацию* о происшедшем событии, на основании которой изменилось представление приемника о множестве возможных исходов наблюдаемой величины. Нам бы хотелось теперь ввести количественную меру информации для построения математической теории.

Определим *количество информации*, содержащейся в сообщении $m$, изменяющем представление приемника о событии с $S_{\text{до}}$ до $S_{\text{после}}$ (предполагая, что $S_{\text{до}} \supseteq S_{\text{после}}$) по формуле

$$I(m) = -log_2 \frac{p(S_{\text{до}})}{p(S_{\text{после}})}. \tag{12}$$

Единицей количества информации является *бит*.

**Пример 1.** В семье должен родиться ребенок. Пространство элементарных исходов данной случайной величины — {*мальчик, девочка*}, — состоит из двух исходов. Отсутствие априорной информации у приемника (родителей) о поле малыша означает, что $S_{\text{до}}$ совпадает с этим пространством. Сообщение источника (врача) «у вас родился мальчик» сужает это множество предположений до множества $S_{\text{после}}$ из единственного исхода *мальчик*. По формуле (12) количество полученной информации определяется как

$$I(m) = -log_2 \frac{p(S_{\text{до}})}{p(S_{\text{после}})} = -log_2 \frac{\mu(S_{\text{до}})}{\mu(S_{\text{после}})} = -log_2 \frac{1}{2} = 1(\text{бит}).$$

**Пример 2.** Из колоды вытягивается карта. Пространство элементарных исходов — все 52 карты полной колоды. В отсутствие изначальной информации пространство предположений $S_{\text{до}\_1}$ также совпадает

со всем пространством. Первое сообщение от источника «выпала трефа» сужает его до $S_{\text{после\_1}}$ из 13 возможных исходов. Второе сообщение «выпала картинка» сужает $S_{\text{до\_2}} = S_{\text{после\_1}}$ до $S_{\text{после\_2}}$, состоящего из 4 исходов. Третье сообщение «выпала дама треф» сужает $S_{\text{до\_3}} = S_{\text{после\_2}}$ до $S_{\text{после\_3}}$, состоящего из единственного исхода. Количество информации, содержащееся в первом сообщении, очевидно, равно $-log_2(13/52) = 2$ битам, во втором — $-log_2(4/13) = 1.5$, в третьем — $-log_2(1/4) = 2$ битам. Нетрудно проверить, что (в силу аддитивности функции логарифм) суммарное количество полученной информации — 5.5 бит, совпадает с количеством информации, которое несло бы сообщение «выпала дама треф» $= -log_2(1/52) = 5.5$ бит.

Этот пример иллюстрирует следующую теорему об аддитивности информации:

**(Т5) Количество информации, переносимое сообщением о событии, равно сумме количеств информации, переносимых сообщениями, последовательно уточняющими это событие.**

Подытожим существенные свойства информации:

— количество полученной приемником информации зависит от его предварительного знания о событии;

— количество информации зависит не от события, а от сообщения о нем.

## 3.3. Формулы Шеннона, Хартли

Предположим теперь, что источник является генератором символов из некоторого множества $\{x_1, x_2, ..., x_n\}$ (назовем его *алфавитом* источника). Эти символы могут служить для обозначения каких-то элементарных событий, происходящих в области источника, но, абстрагируясь от них, в дальнейшем будем считать, что рассматриваемым событием является поступление в канал самих символов. Если $p(x_i)$ — вероятность поступления в канал символа $x_i$, то очевидно, что

$$\sum_{i=1}^{n} p(x_i) = 1.$$

Рассмотрим теперь модель, в которой элементарным исходом является текстовое *сообщение*, т. е. некоторая цепочка символов из алфавита источника. Таким образом, $\Omega$ — это множество всех цепочек символов произвольной длины. По поступившему сообщению $m$ можно посчитать

экспериментальную *частоту* встречаемости в нем каждого символа где $N$ — общая длина сообщения, а $n_i$ — число повторений в нем символа $x_i$.

$$\nu_m(x_i) = \frac{n_i}{N}, \tag{13}$$

Понятно, что анализируя различные сообщения, мы будем получать различные экспериментальные частоты символов, но для источников, характеризующихся закономерностью выдачи символов (их называют эргодическими), оказывается, что в достаточно длинных сообщениях все частоты символов сходятся к некоторым устойчивым величинам

$$p(x_i) = \lim_{N\to\infty} \frac{n_i}{N}, \tag{14}$$

которые можно рассматривать как *распределение вероятностей* выдачи символов данным источником.

Рассмотрим (достаточно длинное) сообщение $m$, состоящее из $n_1$ символов $x_1$, $n_2$ символов $x_2$ и т. д. в произвольном порядке, как серию элементарных событий, состоящих в выдаче одиночных символов. Тогда вероятность появления на выходе источника сообщения $m$ равна

$$p(m) = (\frac{n_1}{N})^{n_1} \cdot ... \cdot (\frac{n_n}{N})^{n_n} = \frac{1}{N^N} \cdot ({n_1}^{n_1} \cdot ... \cdot {n_n}^{n_n}).$$

Количество информации, переносимой сообщением $m$ длины $N$, определяется как

$$I(m) = -log_2 \frac{p(m)}{1} = -log_2((\frac{n_1}{N})^{n_1} \cdot ... \cdot (\frac{n_n}{N})^{n_n}) = -\sum_{i=1}^{N} n_i \cdot log_2(\frac{n_i}{N}).$$

Количество информации, приходящейся в среднем на каждый символ в сообщении $m$, есть

$$I_0(m) = \frac{1}{N} \cdot I(m),$$

где $N$ — длина сообщения $m$.

Перейдем к пределу по длине всевозможных сообщений ($N \to \infty$):

$$I_0(A) = \lim_{N\to\infty} I_0(m) = \lim_{N\to\infty} \frac{1}{N} \cdot (-\sum_{i=1}^{N} n_i \cdot log_2(\frac{n_i}{N})) =$$

$$= (-\sum_{i=1}^{N} \lim_{N\to\infty}(\frac{n_i}{N}) \cdot log_2 \lim_{N\to\infty}(\frac{n_i}{N})).$$

По формуле (14), вспоминая, что в достаточно большом сообщении $p(x_i) = \lim_{N\to\infty} \frac{n_i}{N}$, получаем

$$I_0(A) = -\sum_{i=1}^{N} p(x_i) \cdot log_2 p(x_i). \tag{15}$$

Формула (15) называется формулой **Шеннона**. Это ключевая формула теории информации.

Величина $I_0(A)$ характеризует среднее количество информации на один символ из алфавита $A$ с заданным (или экспериментально определенным) распределением вероятностей

$$p(x_1), p(x_2), ..., p(x_n).$$

Рассмотрим случай, когда все символы в алфавите равновероятны, т. е.

$$p(x_1) = p(x_2) = \ \ldots \ = p(x_n) = \frac{1}{N}.$$

Среднее количество информации, приходящееся на каждый символ такого алфавита, по формуле Шеннона определяется следующим образом:

$$I_0(A) = -\sum_{i=1}^{N} \frac{1}{N} \cdot log_2 \frac{1}{N} = -N \cdot \frac{1}{N} \cdot log_2 \frac{1}{N} = log_2 N. \tag{16}$$

Формула (16) есть формула **Хартли**.

## 3.4. Понятие кода

Введем теперь понятие кода для сообщений. *Функцией кодирования* называется отображение $K$ : Алф1 $\rightarrow$ Алф2, которое некоторой цепочке символов из алфавита Алф1 сопоставляет некоторую цепочку символов из алфавита Алф2. Последняя называется *кодом* исходной цепочки.

Можно выделить четыре типа кодирования:

1. Символ $\rightarrow$ символ, т. е. код $K$ : Алф1 $\rightarrow$ Алф2, причем |Алф1| = |Алф2|. $|S|$ обозначает количество элементов в множестве $S$, |Алф1| — количество символов в алфавите Алф1. Код $K$ в этом случае заменяет одиночные символы алфавита Алф1 одиночными символами алфавита Алф2.

2. Символ $\to$ слово, т. е. код $K$ : Алф1 $\to$ Алф2$^*$, причем обычно |Алф1| $>$ |Алф2|. Алф2$^*$ обозначает цепочку (последовательность) символов из алфавита Алф2. Здесь одиночные символы Алф1 кодируются цепочками из Алф2 длины $\geq 1$.

3. Слово $\to$ символ, т. е. код $K$ : Алф1$^*$ $\to$ Алф2, причем |Алф1| $<$ |Алф2|.

4. Слово $\to$ слово, т. е. код $K$ : Алф1$^*$ $\to$ Алф2$^*$.

**Пример.** Пусть Алф1 $= \{a, b, c, d\}$, Алф2 $= \{0, 1\}$. Тогда примером кода типа 2 может являться код, который символ $a$ переводит в символ 0 (будем писать $a \to 0$), $b \to 01$ ($b$ – в цепочку символов 01), $c \to 10$, $d \to 1$.

Допустим, что слово $s = 01101010$ было получено в результате кодирования по данному правилу. Нам хотелось бы узнать, какое слово из алфавита Алф1 было переведено в слово $s$. Для этого нужно выполнить действие, называемое *декодированием*, или *дешифровкой*. Это действие состоит в том, чтобы найти обратное преобразование $K^{-1}$ : Алф2 $\longrightarrow$ Алф1. В рассматриваемом примере кода возможны два варианта дешифровки кода. Слово *addbba* кодируется словом $s$, и слово *bccc* также кодируется словом $s$. Мы столкнулись с проблемой неоднозначности декодирования, когда две разные исходные цепочки в Алф1 имеют один и тот же код в Алф2. Для успешной дешифровки необходимо, чтобы коды допускали однозначное декодирование.

Код называется *префиксным*, если никакое кодовое слово не является началом никакого другого кодового слова. В примере выше слово, состоящее из одного символа 0, является началом другого слова 01. Следовательно, рассмотренный код не является префиксным кодом.

Утверждение. **Префиксный код всегда может быть однозначно декодирован.**

*Пояснение.* Если кодовая цепочка дешифруема, то минимальная по длине начальная цепочка, являющаяся кодом некоторого символа, позволяет восстановить первый символ исходной цепочки. В силу префиксности никакая более длинная начальная цепочка не может быть кодом другого символа. Часть кода, оставшаяся после отделения кода найденного символа, дешифруется также.

**Пример.** Пусть Алф1 $= \{a, b, c, d\}$, Алф2 $= \{0, 1\}$. Рассмотрим код $K$ такой, что

$a \to 0$,

$b \to 101$,

$c \to 110$,

$d \to 1110$.

Рассмотрим слово $s = 01101010$, состоящее из символов алфавита Алф2. Код $K$ является префиксным, поэтому слово $s$ однозначно декодируется, а именно: *acba*.

Критерии качества кодирования:

— минимальная длина кода;

— однозначное декодирование.

При преобразовании информации из одного вида в другой количество информации не возрастает.

## 3.5. Связь между информационной емкостью и средней длиной кода. Избыточность кодирования

Рассмотрим пример двоичного кодирования типа «символ $\longrightarrow$ слово». Пусть в области источника с равной вероятностью возникают два элементарных события, которые кодируются в канале символами 0 и 1. По формуле Хартли количество информации, переносимое каждым символом кодового слова, есть $log_2 2 = 1$ бит. Если используется двоичное кодирование типа «символ $\longrightarrow$ слово», то для передачи одного символа исходного алфавита используется различное число двоичных цифр. Поделив длину достаточно большого кодового сообщения на длину исходного сообщения, мы получим усредненную длину кодового слова для данного способа кодирования, т. е. среднее число кодовых символов, использующихся для кодирования одного исходного символа (как мы знаем, эта величина будет зависеть от частоты встречаемости исходных символов и длины кодовых слов). Нас интересует построение такого кода, при котором эта средняя длина кода символа была бы как можно меньше.

Оказывается, что величина $I_0(A)$, введенная выше и названная средней информационной емкостью символа алфавита A, определяет предел сжимаемости кода: никакой двоичный код не может иметь среднюю длину меньшую, чем $I_0$, в противном случае можно было бы передать некоторое количество информации меньшим числом битов, что невозможно. Таким образом, любой код может быть лишь в большей или меньшей степени *избыточным*, т. е. использовать для передачи $I_0$ битов информации на один исходный символ (в среднем) то же или большее количество символов кодового алфавита (в среднем).

Относительная избыточность кода характеризуется как отношение числа «избыточных» битов в коде к общей длине кода, т. е. если сообщение из $N$ символов алфавита $A$ с информационной емкостью $I_0(A)$ закодировано кодом длины $L$ битов, то избыточное число битов есть

$$L - N \cdot I_0(A),$$

а удельная избыточность каждого символа кода

$$\frac{L - N * I_0(A)}{L} = 1 - \frac{N}{L} * I_0(A). \tag{17}$$

Заметив, что $\lim_{N\to\infty} \frac{L}{N}$ есть средняя длина кодового слова $K_0(A)$, получим независимое от сообщения соотношение для избыточности кода:

$$Z(K) = 1 - \frac{I_0(A)}{K_0(A)}. \tag{18}$$

Из формулы (18) следует, что оптимальным кодом с нулевой избыточностью является код со средней длиной кодового слова $K_0 = I_0(A)$ битов или наиболее близкий к нему. Построением такого кода мы и займемся.

**Резюме.** $I_0(A)$ показывает, какое в среднем количество двоичных символов нужно для записи всех кодовых слов алфавита A при произвольном кодировании «символ $\longrightarrow$ слово».

Для алфавитов с равновероятными символами формула Хартли определяет минимальную необходимую длину кодового слова, например для алфавита $ASCII$

$$I_0(ASCII) = log_2 256 = 8 \text{ бит}.$$

Таким образом, любой 8-битный код для $ASCII$ будет оптимальным.

## 3.6. Метод кодирования со сжатием по Хаффману

Метод кодирования по Хаффману состоит в построении для заданной входной строки из произвольных символов двоичного кода по следующему алгоритму.

**Алгоритм А12:**

1. Определить набор входных символов $A =< a_1, a_2, ..., a_n >$ и распределение их экспериментальных частот $P =< p_1, p_2, ..., p_n >$.

2. Для данных алфавита $A$ и распределения $P$ построить так называемое *двоичное дерево Хаффмана.*

3. По построенному дереву составить функцию двоичного кодирования по Хаффману типа «символ → слово».

4. Используя функцию кодирования, закодировать входную строку.

Первый шаг алгоритма, очевидно, реализуется однократным просмотром входной строки, накоплением встречающихся символов и подсчетом их количества: как по отдельности, так и в целом. Распределение частот строится в конце просмотра по накопленным счетчикам по формуле (13).

На втором шаге алгоритма строится дерево Хаффмана, узлы которого помечены парами $<a, p>$, где $a$ – некоторый исходный символ или вспомогательный псевдосимвол, а $p$ – его частота. Этот шаг детализируется следующим алгоритмом.

**Алгоритм А13:**

**Процедура Шаг_построения:**

**вход:** набор свободных узлов дерева;

•$W_{\text{вх}} = \{< a_1, p_1 >, < a_2, p_2 >, ..., < a_k, p_k >\}$;

• найти в $W_{\text{вх}}$ два узла $W_i =< a_i, p_i >$ и $W_j =< a_j, p_j >$ с минимальными частотами $p_i$ и $p_j$;

• создать новый узел $W_{i\&j} =< a_{i\&j}, p_{i\&j} = p_i + p_j >$, где $a_{i\&j}$ – новый псевдосимвол, представляющий "$a_i$ или $a_j$"и отличный от всех $a_k$, $p_{i\&j}$ – частота, приписываемая этому новому символу;

• присоединить узлы $W_i$ и $W_j$ к $W_{i\&j}$ как сыновей;

• исключить вновь связанные узлы и добавить новый свободный узел: $W_{\text{вых}} := (W_{\text{вх}} \bigcup \{W_{i\&j}\}) \backslash \{W_i, W_j\}$;

**выход:** обновленный набор свободных узлов $W_{\text{вых}}$, содержащий $k - 1$ узел.

**конец процедуры;**

**вход:** $A$ – исходный набор символов $< a_1, ..., a_N >$,

$P =< p_1, p_2, ..., p_N >$ – распределение их частот;

•$W_0 := \{< a_1, p_1 >, ..., < a_N, p_N >\}$ (начальный набор свободных узлов соответствует встречающимся символам);

• цикл по $i$ от 1 до $N$

• $W_i :=$ Шаг_построения$(W_{i-1})$;

• конец цикла;

**выход:** дерево Хаффмана, построенное в цикле с корневым узлом, содержащимся в $W_N$.

Проиллюстрируем этап построения на примере строки «кол около ко-

локола». Набор встречающихся символов и их экспериментальные частоты таковы:

к — $\frac{4}{18}$,

л — $\frac{4}{18}$,

о — $\frac{7}{18}$,

пробел — $\frac{2}{18}$,

а — $\frac{1}{18}$.

Нарисуем ряд узлов (рис. 1), образующих начальный список $W_0$ (дерево будет нагляднее, если выписывать узлы в порядке убывания частот):

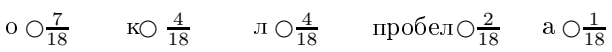


*Рис. 1.* Начальный список узлов дерева Хаффмана

На первой итерации цикла построения дерева мы находим два узла с минимальными частотами (это крайние справа узлы «пробел» и «*a*») и соединяем их с новым узлом $< b_1, p >$, $b_1$ – это псевдосимвол, представляющий любой из символов «пробел» или «*a*», а частота его, очевидно, равна сумме частот $\frac{2}{18} + \frac{1}{18} = \frac{3}{18}$. Новый список свободных узлов $W_1$ состоит из узлов «о», «к», «л» и $b_1$ (рис. 2) и короче первоначального на 1 узел.

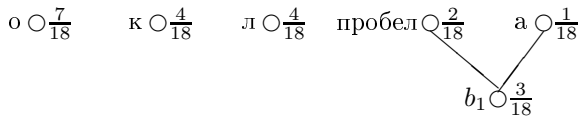


*Рис. 2.* Дерево Хаффмана после 1-й итерации

На второй итерации цикла вновь находятся узлы с минимальными частотами: это $< b_1, \frac{3}{18} >$ и один из узлов <«к»,$\frac{4}{18}$ > и <«л»,$\frac{4}{18}$ >. Здесь мы вправе выбрать любой: пусть это будет «л». Вновь происходит слияние двух свободных узлов в один $< b_2, \frac{7}{18} >$, $b_2$ представляет уже три исходных символа (рис. 3).

Продолжая, мы будем получать все более короткие списки $W_i$, а в дерево будем добавлять каждый раз новый узел. В итоге для нашего примера мы построим дерево, изображенное на рис. 4.

Теперь переходим к 3-му шагу алгоритма А13 – построению функции кодирования. Пометим дуги, исходящие из каждой вершины дерева к

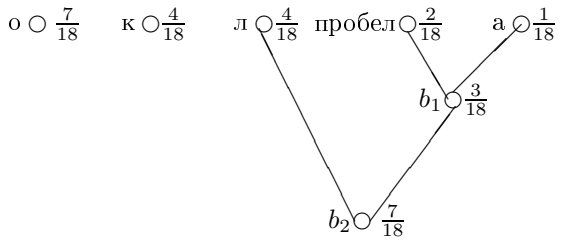


*Рис. 3.* Дерево Хаффмана после 2-й итерации

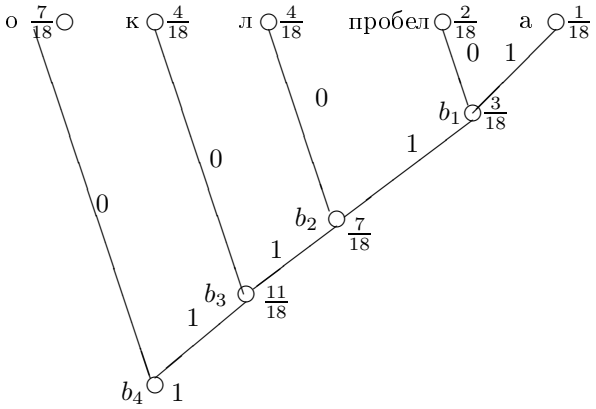


*Рис. 4.* Дерево Хаффмана

ее сыновьям, единицей и нулем. Порядок пометки не имеет значения; на рис. 4 мы пометили нулями дуги, исходящие налево, а единицей — направо.

А теперь, проходя путь из корня дерева в вершину, имеющую пометкой символ алфавита $A$, и выписывая все пометки дуг на этом пути, мы получим кодовое слово для этого символа.

В нашем примере коды будут такими:

| | | |
|---|---|---|
| символ | — | код |
| о | — | 0, |
| к | — | 10, |
| л | — | 110, |
| пробел | — | 1110, |
| а | — | 1111. |

Наконец, четвертый шаг очевиден: код исходного сообщения «кол около колокола», составленный с использованием построенной кодовой таблицы, есть

$$100110111001001100111010010010011011111.$$

Длина кодовой строки $L = 39$.

По построению кода Хаффмана легко увидеть, что он является префиксным: ни один путь от корня к вершине не проходит через другую вершину.

Заметим, что неоднозначность выбора одного из подходящих узлов при построении дерева Хаффмана, а также свобода разметки дуг при построении функции кодирования означают, что кодирование алфавита этим способом не является однозначно определенной процедурой. Для разобранного выше примера можно построить и другое дерево, изображенное на рис. 5.

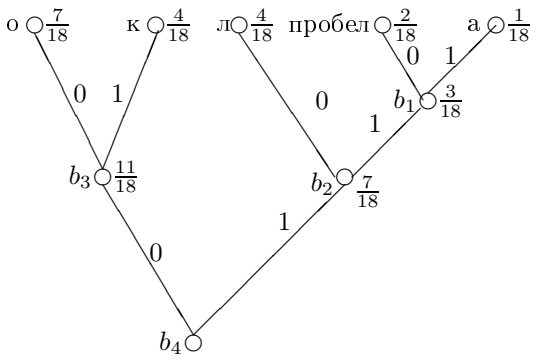


*Рис. 5.* Дерево Хаффмана

В этом случае код, соответствующий дереву, изображенному на рис. 5, будет следующим:

| | | |
|---|---|---|
| символ | — | код |
| о | — | 00, |
| к | — | 01, |
| л | — | 10, |
| пробел | — | 110, |
| а | — | 111. |

что приводит к другому коду исходного сообщения:

$$010010110000100100011001001000010010111.$$

Его длина тоже 39 — это, вообще говоря, совпадение, но отклонение может быть лишь незначительным.

Посчитаем информационную емкость кода, соответствующего дереву на рис. 4: длина исходного сообщения $N = 18$, длина кода $L = 39$ битов. По формуле (15) удельная информационная емкость алфавита $A$ с распределением $P$ есть

$$I_0(A) = \frac{8}{18} * log_2\frac{18}{4} + \frac{1}{18} * log_2\frac{18}{1} + \frac{7}{18} * log_2\frac{18}{7} + \frac{2}{18} * log_2\frac{18}{2} = 2.1 \text{ бита.}$$

Итак, теория информации предсказывает, что в среднем (с учетом вероятности появления) длина каждого кодового слова при любом способе кодирования этого текста не может быть меньше 2.1 двоичных цифр.

Используя формулу (18), посчитаем избыточность кода в нашем примере

$$Z = 1 - \frac{N}{L} * I_0(A) = 1 - \frac{18}{39} * 2.1 = 0.03,$$

т. е. построенный код избыточен всего на 3 %. (Проверьте для нескольких других наугад составленных кодов, что их избыточность будет больше.)

Приведенные расчеты иллюстрируют раннее утверждение, что код Хаффмана является оптимальным в данном классе методов кодирования, т. е. его избыточность наиболее близка к нулю. Интуитивно это можно объяснить тем, что чем чаще встречаются символы, тем более короткие коды для них строятся (так как они включаются в дерево Хаффмана на все более поздних стадиях) за счет того, что более редким символам возможно приписать более длинные коды. Строгое доказательсво этого факта выходит за рамки нашего курса.

Еще один вопрос, имеющий практическое значение при реализации этого метода: насколько длинным может оказаться кодовое слово? (Это важно для оценки размеров отводимой памяти.) Ответ на этот вопрос, на первый взгляд, неожиданный.

**(Т6) Длина кодового слова в методе Хаффмана ограничена порядковым номером минимального числа Фибоначчи, превосходящего длину входного текста.**

Так, для кодирования текста размером 1Мб кодовые слова не будут превышать 32 битов или 1 машинного слова. Этот результат напрямую следует из доказуемого факта, что при заданной длине текста $N$ наибольшая высота дерева Хаффмана возникает тогда, когда числители

частот в экспериментальном распределении образуют начало последовательности Фибоначчи.

Метод Хаффмана наряду с другими методами кодирования используется в современных системах сжатия информации. Цель этого описания — не систематическое изложение теории по вероятности и кодированию, а необходимый минимум знаний, нужный для написания учебно-демонстрационных программ по архивации.