Профессор
**Игорь Н. Бекман**

# КОМПЬЮТЕРЫ В ИНФОРМАТИКЕ

Курс лекций

## Лекция 4. КОДИРОВАНИЕ ИНФОРМАЦИИ

Содержание

**1. СИСТЕМЫ СЧИСЛЕНИЯ** 1
***1.1 Позиционные системы счисления*** 2
***1.2 Арифметические операции в P-ичных системах счисления*** 4
**2. ТЕОРИЯ КОДИРОВАНИЯ** 6
***2.1 Определение понятий*** 6
***2.2 Информация и алфавит*** 7
***2.3 Кодирование сигнала*** 8
***2.4 Задача кодирования*** 8
***2.5 Первая теорема Шеннона*** 9
***2.6 Способы кодирования/декодирования информации*** 10
**3. КОНКРЕТНЫЕ МЕТОДЫ КОДИРОВАНИЯ** 15
***3.1 Алфавитное неравномерное двоичное кодирование*** 15
***3.2 Равномерное алфавитное двоичное кодирование*** 19
***3.3 Блочное двоичное кодирование*** 20
***3.4 Методы сжатия цифровой информации*** 21
3.4.1 Алгоритмы обратимых методов сжатия информации 22
3.4.2 Алгоритмы с регулируемой потерей информации 23
***3.5 Коды обнаружения и исправления ошибок*** 25

Информация редко используется человеком в чистом виде, она всегда как-то представлена – формализована или закодирована. Одна и та же информация может быть представлена в разных формах, а одни и те же символы нести разную смысловую нагрузку.

В данной лекции мы рассмотрим различные типы счисления (в первую очередь - двоичную) и теоретические основы кодирования. Затем мы остановимся на сравнительном анализе различных методов кодирования, в том числе - методы сжатия цифровой информации и коды исправления ошибок.

## 1. СИСТЕМЫ СЧИСЛЕНИЯ

Для машинной обработки информации её необходимо представить в какой-либо системе счисления.

**Система счисления** – совокупность приёмов наименования и записи чисел с помощью цифр.

Как мы обсудили в первой лекции, счётные устройства 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 счётные машины (суммирующая машина Паскаля (1642), счётная машина Лейбница (1673), аналитическая машина Бэббиджа (1948)), были разработаны на основе десятичной системы счисления. Эти механические системы были громоздки. Так, если бы проект машины Бэббиджа был реализован, то по размерам машина сравнялась бы с локомотивом, и чтобы привести в движение её устройства, понадобился бы паровой двигатель. Причинами этого были механический принцип построения устройств и использование десятичной системы счисления, затрудняющей создание простой элементной базы.

Через 63 года после смерти Бэббиджа немецкий студент Конрад Цузе начал работу по созданию машины, основанной на принципах действия Аналитической машины Бэббиджа. В 1937 машина Z1 была готова. Работала она на основе двоичной системы счисления и была чисто механической. Но использование двоичной системы сильно уменьшило размеры устройства – машина занимала всего два квадратных метра.

В современных компьютерах вся информация также хранится в виде последовательностей нулей и единиц. Однако двоичная система счисления в чистом виде обладает рядом принципиальных недостатков, тормозящих развитие 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 её передаче). Практическая потребность в решении этих вопросов вызывает сегодня повышенный интерес к способам представления информации в компьютере и новым компьютерным арифметикам. Так, например, 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.

Ниже мы рассмотрим основные системы счисления, используемые в компьютере.

## *1.1 Позиционные системы счисления*

**Систе́ма счисле́ния** или **нумерация** – способ записи (обозначения) чисел.

**Систе́ма счисле́ния** - символический метод записи чисел, представление чисел с помощью письменных знаков. Система счисления: даёт представления множества чисел (целых или вещественных); даёт каждому числу уникальное представление (или, по крайней мере, стандартное представление); отражает алгебраическую и арифметическую структуру чисел.

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

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

**Позиционная система счисления** – система счисления, в которой один и тот же числовой знак (цифра) в записи числа имеет различные значения в зависимости от того места (разряда), где он расположен.

**Число́** - абстракция, используемая для количественной характеристики объектов. Возникнув ещё в первобытном обществе из потребностей счёта, понятие числа изменялось и обогащалось и превратилось в важнейшее математическое понятие.

**Ци́фры** - знаки для записи чисел (числовые знаки). Слово «цифра» без уточнения обычно означает один из следующих знаков: 0 1 2 3 4 5 6 7 8 9 (т. н. «арабские цифры»). Существуют также много других вариантов: римские цифры (I V X L C D M), шестнадцатеричные цифры (0 1 2 3 4 5 6 7 8 9 A B C D E F), в некоторых языках, например, в древнегреческом, в иврите, в церковнославянском, существует система записи чисел буквами и др.

Виды систем счисления: позиционные (значение цифры зависит от её позиции в изображении числа): двоичная, троичная,…, десятичная,…, шестнадцатиричная,…, шестидесятиричная и т.д.; непозиционные (значение цифры не зависит от места, занимаемого в изображении числа: единичная, римская, алфавитные).

**Пример 1:** Римская система счисления:

| V | X | L | C | D | M |
|---|---|---|---|---|---|
| 5 | 10 | 50 | 100 | 500 | 1000 |

XXVIII=10+10+5+1+1+1=28; XCIX= -10+100-1+10=99

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

Последовательность чисел, каждое из которых задаёт «вес» соответствующего разряда, называется базисом позиционной системы счисления.

***Основное достоинство позиционной системы – возможность записи произвольного числа при помощи ограниченного количества символов.***

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

Позиционная система: число $x$ может быть представлено в системе с основанием p, как $x=a^n \cdot p^n + a^{n-1} \cdot p^{n-1} + \ldots + a^1 \cdot p^1 + a^0 \cdot p^0$, где $a^n, \ldots, a^0$ – цифры в представлении данного числа, в основание системы счисления.

**Пример 2.** $103510 = 1 \cdot 10^3 + 0 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$

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:

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

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

| Десятичная | Двоичная | Восьмеричная | Шестнадцатеричная |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **1** | 1 | 1 | 1 |
| **2** | 10 | 2 | 2 |
| ... | ... | ... | ... |
| **10** | 1010 | 12 | A |
| **11** | 1011 | 13 | B |
| **12** | 1100 | 14 | C |
| **13** | 1101 | 15 | D |
| **14** | 1110 | 16 | E |
| **15** | 1111 | 17 | F |
| **16** | 10000 | 20 | 10 |
| ... | ... | ... | ... |
| **1000** | 1111101000 | 1750 | 3E8 |

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 её базисом – последовательность чисел Фибоначчи 1, 2, 3, 5, 8, 13, 21, 34, 55, 89… Базисом факториальной системы счисления является последовательность 1!, 2!,…,n!,… . Здесь количество цифр, используемых в разряде, увеличивается с ростом номера разряда.

Любое десятичное число можно представить в любой позиционной системе счисления, а для целых чисел в большинстве систем это можно сделать единственным образом.

Любое натуральное число можно записать в какой угодно $P$-ичной системе счисления, причём единственным образом.

Рассмотрим запись чисел в разных системах счисления. Двоичная (используются цифры 0, 1); восьмиричная (используются цифры 0, 1, …,7); шестнадцатиричная (для первых целых чисел от нуля до девяти используются цифры 0, 1,…,9, а для следующих чисел – от десяти до пятнадцати – в качестве цифр используются символы A, B, C, D, E, F).

Существуют два основных формата представления чисел в памяти компьютера: для кодирования целых чисел и для кодирования действительных чисел

$$A_q = \pm\Sigma a_i q^i \qquad (2)$$

$$A_q = \pm\left(a_{n-1}q^{n-1} + a_{n-2}q^{n-2} + \ldots + a_0 q^0 + a_{-1}q^2 + \ldots + a_{-m}q^{-m}\right) \qquad (3)$$

Здесь $A$ – само число, $q$ – основание системы счисления, $a_i$ - цифры, принадлежащие алфавиту данной системы счисления, $n$ – число целых разрядов числа, $m$ – число дробных разрядов числа.

$M=a_{n-1}\,a_{n-2}\,a_{n-3}\ldots a_3 a_2 a_1 a_0$ – запись числа $a_{n-1}q^{n-1}+a_{n-2}q^{n-2}+\ldots+a_1q_1+a_0q_0+a_{-1}q^{-1}+\ldots+a_{-m}q^{-m}$, где $a_i$ - цифры системы счисления, $n$ и $m$ – число целых и дробных разрядов, соответственно.

**Пример 3.** $1011{,}12=1*2^3+0*2^2+1*2^1+1*2^0+1*2^{-1}$;

$276{,}528=2*8^2+7*8^1+6*8^0+5*8^{-1}+2*8^{-2}$;

M=342 q=10 $M = 3*10^2+4*10^1+2*10^0$

q=5 $M=3*5^2+4*5^1+2*5^0$ (9710)

q =8 $M=3*8^2+4*8^1+2*9^0$ (22610)

$1 = 1_2$;
$2 = 10_2$
$3 = 11_2$
$4 = 100_2$
$5 = 101_2$
$6 = 110_2$
$7 = 111_2$
$8 = 1000_2$
$9 = 1001_2$
$10 = 1010_2$
$11 = 1011_2$
$12 = 1100_2$

$13 = 1101_2$
$14 = 1110_2$
$15 = 1111_2$
$16 = 10000_2$

**Рис. 1.** Первые 16 чисел, представленные в двоичной системе.

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.

**Пример 4.** Известно, что правильную десятичную дробь 1/3 нельзя записать в виде конечной десятичной дроби. Однако в троичной и 9-ричной системах счисления эта дробь будет записана в виде конечное $P$-ичной дроби. В 3-ичной системе счисления: $1/3=0{,}1_3$; в 9-ричной: $1/3=0{,}3_9$.

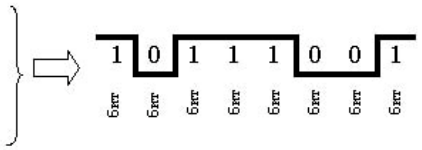
Десятичная обыкновенная дробь будет точно представима конечной *P*-ичной дробью, если существует такое натуральное число *m*, при умножении на которое знаменателя дроби можно получить некоторую натуральную степень числа *P*. Если же такого числа не существует, то в *P*-ичной системе счисления дробь окажется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

**Пример 5.** Запишем 5/16 в двоичной системе. В знаменателе уже стоит четвёртая степень двойки. Переведём числитель в двоичную систему ($5=101_2$) и дополним получившееся число до четырёх цифр: 0101. В результате получим: $5/16=0{,}0101_2$.

**Пример 6.** Запишем $0{,}011_2=\frac{11_2}{1000_2}$ в десятичной системе. Для того, чтобы знаменатель, равный сейчас 23, оказался степенью десяти, его нужно домножить на $5^3$,тогда

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

| Виды информации | Двоичный код |
|---|---|
| Числовая | |
| Текстовая | |
| Графическая | |
| Звуковая | |
| Видео | |




**Двоичная система счисления** - позиционная система счисления с основанием 2. В этой системе счисления натуральные числа записываются с помощью всего лишь двух символов (в роли которых обычно выступают цифры 0 и 1).

В этой лекции нас в основном будет интересовать двоичная система счисления, точнее - двоичный (цифровой) код, основанный на двоичной системе счисления, использующий для представления буквенно-цифровых и других символов наборы комбинаций цифр 1 и 0. В двоичной системе счисления всего две цифры, называемые двоичными. Название разряда двоичного числа – бит. Веса разрядов в двоичной системе изменяются по степеням двойки. Поскольку вес каждого разряда уменьшается либо на 0, либо на 1, то в результате значение числа определяется как сумма соответствующих значений степени двойки. $10102=1\cdot 23+0\cdot 22+1\cdot 21+0\cdot 20=10$.

Известны следующие преимущества использования двоичной системы в вычислительной технике: простота технической реализации; надёжность и помехоустойчивость; возможность применения аппарата булевой алгебры для выполнения логических операций; простота правил двоичной арифметики. Двоичная система используется в цифровых устройствах, т.к. является наиболее простой и соответствует требованиям: 1) Чем меньше значений существует в системе, тем проще изготовить отдельные элементы, оперирующие этими значениями. В частности, две цифры двоичной системы счисления могут быть легко представлены многими физическими явлениями: есть ток — нет тока, индукция магнитного поля больше пороговой величины или нет и т. д. 2) Чем меньше количество состояний у элемента, тем выше помехоустойчивость и тем быстрее он может работать. Например, чтобы закодировать три состояния через величину индукции магнитного поля, потребуется ввести два пороговых значения, что не будет способствовать помехоустойчивости и надёжности хранения информации. 3) Двоичная арифметика является довольно простой. Простыми являются таблицы сложения и умножения - основных действий над числами. 4) Возможно применение аппарата алгебры логики для выполнения побитовых операций над числами.

## *1.2 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.

**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. Наиболее простыми являются таблицы сложения в двоичной и троичной системах счисления (индексы 2 и 3 опущены).

| + | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 10 |

| + | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 1 | 1 | 2 | 10 |
| 2 | 2 | 10 | 11 |

**Рис. 2.** Таблицы сложения двоичной и троичной системы счисления.

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 имеем: $(P-1)+(P-1)=2\cdot P-2=1[P-2]_P$. Например, в четвертичной 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 меньшим, чем 2*P* (максимум $2P\text{-}1=1[P\text{-}1]_P$. Т.е. результат сложения двух положительных *P*-чисел либо имеет столько же значений, что и максимальное из двух слагаемых, либо на одну цифру больше, но этой цифрой может быть только единица.

**Пример 7.** Сложение столбиком в двоичной, троичной и шестнадцатеричной системах счисления.

$$\begin{array}{r} 101{,}01_2 \\ +\ 1{,}11_2 \\ \hline 111{,}00_2 \end{array} \qquad \begin{array}{r} 21_3 \\ +\ 2{,}1_3 \\ \hline 100{,}1_3 \end{array} \qquad \begin{array}{r} F2A_{16} \\ +\ E9_{16} \\ \hline 1013_{16} \end{array}$$

Арифметические действия, выполняемые в двоичной системе, подчиняются тем же правилам, что и в десятичной системе, но в двоичной системе старший разряд возникает чаще, чем в десятичной. Вот как выглядит таблица сложения в двоичной системе: 0+0=0; 0+1=1; 1+0=1; 1+1=0 (перенос в старший разряд)

**Вычитание.** Вычитание из большого 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 придём к необходимости выполнения следующего действия: $10_P+a\text{-}b=1a_P\text{-}b$. Для этого в столбце *b* таблицы сложения мы уже ищем число 1aP, левая цифра в соответствующей строке является результатом вычитания.

**Пример 8.** Вычитание в двоичной, троичной и шестнадцатеричной системах счисления.

$$\begin{array}{r} 101_2 \\ -\ 10{,}1_2 \\ \hline 10{,}1_2 \end{array} \qquad \begin{array}{r} 210_3 \\ -\ 102_3 \\ \hline 101_3 \end{array} \qquad \begin{array}{r} A10_{16} \\ -\ 102_{16} \\ \hline 90E_{16} \end{array}$$

**Умножение**. Для выполнения умножения двух многозначных чисел в *P*-ичной системе счисления надо иметь таблицы умножения и сложения в этой системе. Таблица умножения для двоичных чисел проста: 0·0=0; 0·1=0; 1·0=0; 1·1=1

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

| × | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 2 |
| 2 | 0 | 2 | 11 |

**Рис. 3.** Таблицы умножения двоичной и троичной систем счисления

Приведём примеры выполнения умножения в двоичной и троичной 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 9.** 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. При сложении столбиком трёх и более слагаемых, действия сложения целесообразно производить последовательно, т.к. сложные вычисления в непривычной системе счисления могут породить ошибки. 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).

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

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.

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

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

**Пример 10.** Наиболее просто деление организовать в двоичной системе, т.к. в ней необходимо лишь сравнивать два числа между собой и вычитать из большого числа меньшее. 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 её период.

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

**Пример 11.** Так как

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

Деление в троичной системе счисления. результат последнего вычитания совпал с

предыдущим, то все оставшиеся цифры дробной части результата совпадут с последней найденной цифрой. Повторяющаяся цифра образует период троичной дроби.
Деление в двоичной системе счисления.
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 предыдущем, встречающимся ранее при подсчёте 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.

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

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

Символы (цифры) выбранные для представления чисел называются базисными.

# 2. ТЕОРИЯ КОДИРОВАНИЯ

## *2.1 Определение понятий*

В процессе кодирования источник информации представляет сообщение в алфавите, который называется **первичным**, далее это сообщение попадает в устройство, преобразующее и представляющее его во **вторичном алфавите**.

**Код** – правило, описывающее соответствие знаков (или их сочетаний) первичного алфавита знаком (их сочетаниями) вторичного алфавита. Код – (1) правило, описывающее соответствие знаков или их сочетаний одного алфавита знакам или их сочетаниям другого алфавита; - (2) знаки вторичного алфавита, используемые для представления знаков или их сочетаний первичного алфавита.

**Кодирование** – операция отожествления символов или групп символов одного кода с символами или группами символов другого кода.

**Кодирование информации** – процесс формирования определенного представления информации. В более узком смысле под термином «кодирование» понимают переход от одной формы представления информации к другой, более удобной для хранения, передачи или обработки.

**Декодирование** - операция, обратная кодированию, т.е. восстановление информации из закодированного вида (восстановление в первичном алфавите по полученной последовательности кодов).

**Кодер** – устройство, обеспечивающее выполнение операции кодирования.

**Кодер** - программист, специализирующийся на кодировании - написании исходного кода по заданным спецификациям.

**Кодер** - одна из двух компонент кодека (пары кодер – декодер).

**Декодер** - некоторое звено, которое преобразует информацию из внешнего вида в вид, применяемый внутри узла. В программном обеспечении: модуль программы или самостоятельное приложение, которое преобразует файл или информационный поток из внешнего вида в вид, который поддерживает другое программное обеспечение.

**Декодер** – устройство, производящее декодирование.

**Шифрование** – разновидность кодирования.

**Шифр** – код, значение и правила использования которого известно ограниченному кругу лиц.

Операции кодирования и декодирования называются обратимыми, если их последовательное применение обеспечивает возврат к исходной информации без каких-либо её потерь.

Примером обратимого кодирования является представление знаков в телеграфном коде и их восстановление после передачи. Примером кодирования необратимого может служить перевод с одного естественного языка на другой – обратный перевод, вообще говоря, не восстанавливает исходного текста. Безусловно, для практических задач, связанных со знаковым представлением информации, возможность восстановления информации по ее коду является необходимым условием применения кода, поэтому в дальнейшем изложении ограничим себя рассмотрением только обратимого кодирования.

Таким образом, кодирование предшествует передаче и хранению информации. При этом хранение связано с фиксацией некоторого состояния носителя информации, а передача – с изменением состояния с течением времени (т.е. процессом). Эти состояния или сигналы будем называть ***элементарными сигналами*** – именно их совокупность и составляет вторичный алфавит.

Любой код должен обеспечивать однозначное чтение сообщения (надежность), так и, желательно, быть экономным (использовать в среднем поменьше символов на сообщение).
Возможность восстановить текст означает, что в языке имеется определенная избыточность, за счет которой мы восстанавливаем отсутствующие элементы по оставшимся. Ясно, что избыточность находится в вероятностях букв и их комбинациях, их знание позволяет подобрать наиболее вероятный ответ.

Компьютер может обрабатывать только информацию, представленную в числовой форме. Вся другая информация (звуки, изображения, показания приборов и т. д.) для обработки на компьютере должна быть преобразована в числовую форму. С помощью компьютерных программ можно преобразовывать полученную информацию, в том числе - текстовую. При вводе в компьютер каждая буква кодируется определенным числом, а при выводе на внешние устройства (экран или печать) для восприятия человеком по этим числам строятся изображения букв. Соответствие между набором букв и числами называется кодировкой символов. Как правило, все числа в компьютере представляются с помощью нулей и единиц, т.е. словами, компьютеры работают в двоичной системе счисления, поскольку при этом устройства для их обработки получаются значительно более простыми.
В процессе преобразования информации из одной формы представления (знаковой системы) в другую осуществляется кодирование. Средством кодирования служит таблица соответствия, которая устанавливает взаимно однозначное соответствие между знаками или группами знаков двух различных знаковых систем. В процессе обмена информацией часто приходится производить операции кодирования и декодирования информации. При вводе знака алфавита в компьютер путем нажатия соответствующей клавиши на клавиатуре выполняется его кодирование, т. е. преобразование в компьютерный код. При выводе знака на экран монитора или принтер происходит обратный процесс - декодирование, когда из компьютерного кода знак преобразуется в графическое изображение.

Кодирование информации распадается на этапы:

1) Определение объёма информации, подлежащей кодированию.
2) Классификация и систематизация информации.
3) Выбор системы кодирования и разработка кодовых обозначений.
4) Непосредственное кодирование.

## *2.2 Информация и алфавит*

Хотя естественной для органов чувств человека является аналоговая форма, универсальной все же следует считать дискретную форму представления информации с помощью некоторого набора знаков. В частности, именно таким образом представленная информация обрабатывается компьютером, передаётся по компьютерным и некоторым иным линиям связи. Сообщение - последовательность знаков алфавита. При их передаче возникает проблема распознавания знака: каким образом прочитать сообщение, т.е. по полученным сигналам установить исходную последовательность знаков первичного алфавита. В устной речи это достигается использованием различных фонем (основных звуков разного звучания), по которым мы и отличает знаки речи. В письменности это достигается различным начертанием букв и дальнейшим анализом написанного. Можно реализовать некоторую процедуру, посредством которой выделить из сообщения тот или иной знак. Но появление конкретного знака (буквы) в конкретном месте сообщения – событие случайное. Следовательно, узнавание (отождествление) знака требует получения некоторой порции информации. Можно связать эту информацию с самим знаком и считать, что знак несет в себе некоторое количество информации.

**Алфавит** [*alphabetos*, от названий первых двух букв греческого А. - альфа и бета; аналогично: азбука - от аз и буки], совокупность графических знаков — букв (например, латинский, русский А.) или слоговых знаков (например, индийский А. деванагари), расположенных в традиционно установленном порядке.

**Знак** - соглашение (явное или неявное) о приписывании чему-либо (означающему) какого-либо определённого смысла (означаемого). Знак — это материально выраженная замена предметов, явлений, понятий в процессе обмена информацией в коллективе. Знаком также называют конкретный случай использования такого соглашения для передачи информации. Знак может быть составным, то есть состоять из нескольких других знаков. Буквы и слова человеческого языка являются знаками. Цифры и числа являются знаками. Наука о знаковых системах называется семиотикой.

**Символы** (в компьютере) - цифры, буквы, иероглифы и т.п.

Для представления дискретной информации используется некоторый алфавит. Однако однозначное соответствие между информацией и алфавитом отсутствует. Другими словами, одна и та же информация может быть представлена посредством различных алфавитов. В связи с такой возможностью возникает

проблема перехода от одного алфавита к другому, причём, такое преобразование не должно приводить к потере информации.

Алфавит, с помощью которого представляется информация до преобразования называется первичным; алфавит конечного представления – вторичным.

В рамках математической постановки задачи кодирования введём обозначения: *A* – первичный алфавит, состоящий из *N* знаков со средней информацией на знак $I^A$; *B* – вторичный из *M* знаков со средней информацией на знак $I^B$. Сообщение в первичном алфавите содержит n знаков, а закодированное - *m* знаков. $I_s(A)$ – информация в исходном сообщении; $I_f(B)$ – информация в закодированном сообщении.

## 2.3 Кодирование сигнала

Кодирование сигнала – это его представление в определенной форме, удобной для последующего использования сигнала, т.е. это правило, описывающее отображение одного набора знаков в другой набор знаков. Тогда отображаемый набор знаков называется исходным алфавитом, а набор знаков, который используется для отображения, - кодовым алфавитом, или алфавитом для кодирования. При этом кодированию подлежат как отдельные символы исходного алфавита, так и их комбинации. Аналогично для построения кода используются как отдельные символы кодового алфавита, так и их комбинации. Например, дана таблица соответствия между натуральными числами трёх систем счисления. Эту таблицу можно рассматривать как некоторое правило, описывающее отображение набора знаков десятичной системы счисления в двоичную и шестнадцатеричную. Тогда исходный алфавит - десятичные цифры от 0 до 9, а кодовые алфавиты - это 0 и 1 для двоичной системы; цифры от 0 до 9 и символы {*A, B, C, D, E, F*} - для шестнадцатеричной.

Кодовой комбинацией (кодом) называется совокупность символов кодового алфавита, применяемых для кодирования одного символа (или одной комбинации символов) исходного алфавита. При этом кодовая комбинация может содержать один символ кодового алфавита. Исходным символом называется символ (или комбинация символов) исходного алфавита, которому соответствует кодовая комбинация. Например, поскольку $8 = 1000_2$ и 8 является исходным символом, 1000 - это кодовая комбинация, или код, для числа 8. В то же время 8 - это исходный символ. Совокупность кодовых комбинаций называется кодом. Взаимосвязь символов (или комбинаций символов, если кодируются не отдельные символы) исходного алфавита с их кодовыми комбинациями составляет таблицу соответствия (таблицу кодов). Обратная процедура получения исходных символов по кодам символов называется декодированием. Очевидно, для выполнения правильного декодирования код должен быть однозначным, т.е. одному исходному символу должен соответствовать точно один код и наоборот.

В зависимости от целей кодирования, различают следующие его виды:

- кодирование по образцу - используется всякий раз при вводе информации в компьютер для её внутреннего представления;
- криптографическое кодирование, или шифрование, – используется, когда нужно защитить информацию от несанкционированного доступа;
- эффективное, или оптимальное, кодирование – используется для устранения избыточности информации, т.е. снижения ее объема, например, в архиваторах;
- помехозащитное, или помехоустойчивое, кодирование – используется для обеспечения заданной достоверности в случае, когда на сигнал накладывается помеха, например, при передаче информации по каналам связи.

## 2.4 Задача кодирования

Пусть первичный алфавит *A* содержит *N* знаков со средней информацией на знак, определенной с учетом вероятностей их появления, $I_1^{(A)}$ (нижний индекс отражает то обстоятельство, что рассматривается первое приближение, а верхний индекс в скобках указывает алфавит). Вторичный алфавит *B* пусть содержит *M* знаков со средней информационной емкостью $I_1^{(A)}$. Пусть также исходное сообщение, представленное в первичном алфавите, содержит *n* знаков, а закодированное сообщение – *m* знаков. Если исходное сообщение содержит $I^{(A)}$ информации, а закодированное – $I^{(B)}$, то условие обратимости кодирования, т.е. неисчезновения информации при кодировании, очевидно, может быть записано следующим образом:

$$I^{(A)} \le I^{(B)}, \qquad (4)$$

смысл которого в том, что операция обратимого кодирования может увеличить количество формальной информации в сообщении, но не может его уменьшить. Однако каждая из величин в данном неравенстве может быть заменена произведением числа знаков на среднюю информационную емкость знака, т.е.:

$$n \cdot l_1^{(A)} \le m \cdot l_1^{(B)} \qquad (5а)$$

или

$$l_1^{(A)} \le \frac{m}{n} \cdot l_1^{(B)} \qquad (5б)$$

Отношение *m/n*, очевидно, характеризует среднее число знаков вторичного алфавита, которое приходится использовать для кодирования одного знака первичного алфавита – будем называть его *длиной кода* или *длиной кодовой цепочки* и обозначим $K^{(B)}$ (верхний индекс указывает алфавит кодов).

В частном случае, когда появление любых знаков вторичного алфавита равновероятно, согласно формуле Хартли $I_1^{(B)}=\log_2 M$, и тогда

$$\frac{l_1^{(A)}}{\log_2 M} \le K^{(B)} \qquad (6)$$

Введём относительную избыточность кода ($Q$):

$$Q = 1 - \frac{l^{(A)}}{l^{(B)}} = 1 - \frac{l_1^{(A)}}{K^{(B)} \cdot l_1^{(B)}} \qquad (7)$$

Данная величина показывает, насколько операция кодирования увеличила длину исходного сообщения. Очевидно, чем меньше $Q$ (т.е. чем ближе она к 0 или, что то же, $I^{(B)}$ ближе к $I^{(A)}$), тем более выгодным оказывается код и более эффективной операция кодирования. Выгодность кодирования при передаче и хранении – это экономический фактор, поскольку более эффективный код позволяет затратить на передачу сообщения меньше энергии, а также времени и, соответственно, меньше занимать линию связи; при хранении используется меньше площади поверхности (объема) носителя. При этом следует сознавать, что выгодность кода не идентична временно́й выгодности всей цепочки кодирование – передача – декодирование; возможна ситуация, когда за использование эффективного кода при передаче придется расплачиваться тем, что операции кодирования и декодирования будут занимать больше времени и иных ресурсов (например, места в памяти технического устройства, если эти операции производятся с его помощью).

Выражение (6) пока следует воспринимать как соотношение оценочного характера, из которого неясно, в какой степени при кодировании возможно приближение к равенству его правой и левой частей. По этой причине для теории связи важнейшее значение имеют две теоремы, доказанные Шенноном. Первая – ее мы сейчас рассмотрим – затрагивает ситуацию с кодированием при передаче сообщения по линии связи, в которой отсутствуют помехи, искажающие информацию. Вторая теорема относится к реальным линиям связи с помехами.

## *2.5 Первая теорема Шеннона*

Первая теорема Шеннона о передаче информации, которая называется также основной теоремой о кодировании при отсутствии помех, формулируется следующим образом:

***При отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором среднее число знаков кода, приходящихся на один знак кодируемого алфавита, будет сколь угодно близко к отношению средних информаций на знак первичного и вторичного алфавитов.***

Используя понятие избыточности кода, можно дать более короткую формулировку теоремы:

***При отсутствии помех передачи всегда возможен такой вариант кодирования сообщения, при котором избыточность кода будет сколь угодно близкой к нулю.***

Эта теорема открывает принципиальную возможность оптимального кодирования. Но из Удобство двоичных кодов ($M$=2) и в том, что при равных длительностях и вероятностях каждый элементарный сигнал (0 или 1) несёт в себе 1 бит информации ($\log_2 M$=1); тогда из теоремы Шеннона:

$$I_1^{(A)} \le K^{(2)} \qquad (8)$$

и первая теорема Шеннона получает следующую интерпретацию:

***При отсутствии помех передачи средняя длина двоичного кода может быть сколь угодно близкой к средней информации, приходящейся на знак первичного алфавита.***

В двоичной системе кодирования:

$$Q = 1 - \frac{I_1^{(A)}}{K^{(2)}} \qquad (9)$$

Определение количества переданной информации при двоичном кодировании сводится к простому подсчету числа импульсов (единиц) и пауз (нулей). При этом возникает проблема выделения из потока сигналов (последовательности импульсов и пауз) отдельных кодов. Приемное устройство фиксирует интенсивность и длительность сигналов. Элементарные сигналы (0 и 1) могут иметь одинаковые или разные длительности. Их количество в коде (длина кодовой цепочки), который ставится в соответствие знаку первичного алфавита, также может быть одинаковым (в этом случае код называется равномерным) или разным (неравномерный код). Наконец, коды могут строиться для каждого знака исходного алфавита (алфавитное кодирование) или для их комбинаций (кодирование блоков, слов). В результате при кодировании (алфавитном и словесном) возможны следующие варианты сочетаний:

**Табл**. **1**. Варианты сочетаний

| Длительности элементарных сигналов | Кодировка первичных символов (слов) | Ситуация |
|---|---|---|
| Одинаковые | Равномерная | (1) |
| Одинаковые | Неравномерная | (2) |
| Разные | Равномерная | (3) |
| Разные | Неравномерная | (4) |

В случае использования неравномерного кодирования или сигналов разной длительности (ситуации (2), (3) и (4)) для отделения кода одного знака от другого между ними необходимо передавать специальный сигнал – временной разделитель (признак конца знака) или применять такие коды, которые оказываются уникальными, т.е. несовпадающими с частями других кодов. При равномерном кодировании одинаковыми по длительности сигналами (ситуация (1)) передачи специального разделителя не требуется, поскольку отделение одного кода от другого производится по общей длительности, которая для всех кодов оказывается одинаковой (или одинаковому числу бит при хранении).

Длительность двоичного элементарного импульса (τ) показывает, сколько времени требуется для передачи 1 бит информации. Очевидно, для передачи информации, в среднем приходящейся на знак первичного алфавита, необходимо время $K^{(r)}\tau$. Таким образом, можно построить такую систему кодирования, чтобы суммарная длительность кодов при передаче (или суммарное число кодов при хранении) данного сообщения была бы наименьшей.

## *2.6 Способы кодирования/декодирования информации*

**Коды по образцу.** Данный вид кодирования применяется для представления дискретного сигнала на том или ином машинном носителе. Большинство кодов, используемых в информатике для кодирования по образцу, имеют одинаковую длину и используют двоичную систему для представления кода (и, возможно, шестнадцатеричную как средство промежуточного представления).

**Криптографические коды.** Криптографические коды используются для защиты сообщений от несанкционированного доступа, потому называются также шифрованными. В качестве символов кодирования могут использоваться как символы произвольного алфавита, так и двоичные коды. Существуют различные методы, рассмотрим два из них: метод простой подстановки и метод Вижинера.

**Эффективное кодирование.** Этот вид кодирования используется для уменьшения объемов информации на носителе - сигнале. Для кодирования символов исходного алфавита используют двоичные коды переменной длины: чем больше частота символа, тем короче его код.

**Эффективность кода** определяется средним числом двоичных разрядов для кодирования одного символа – $I_{cp}$ по формуле

$$I_{\tilde{n}\eth} = \sum_{i=1}^{k} f_i n_1, \qquad (10)$$

где k – число символов исходного алфавита; $n_s$ – число двоичных разрядов для кодирования символа s; $f_s$ – частота символа $s$; причем $\sum_{i=1}^{\ddot{e}} f_i = 1.$

При эффективном кодировании существует предел сжатия, ниже которого не «спускается» ни один метод эффективного кодирования - иначе будет потеряна информация. Этот параметр определяется **предельным значением двоичных разрядов** возможного эффективного кода – $I_{np}$:

$$I_{i\eth} = -\sum_{i=1}^{n} f_i \log_2 f_i, \qquad (11)$$

где $n$ – мощность кодируемого алфавита, $f_i$ – частота i-го символа кодируемого алфавита.

Существуют два классических метода эффективного кодирования: метод Шеннона-Фано и метод Хаффмена. Входными данными для обоих методов является заданное множество исходных символов для кодирования с их частотами; результат - эффективные коды.

***Метод Шеннона-Фано.*** Этот метод требует упорядочения исходного множества символов по не возрастанию их частот. Затем выполняются следующие шаги:

а) список символов делится на две части (назовем их первой и второй частями) так, чтобы суммы частот обеих частей (назовем их $\Sigma_1$ и $\Sigma_2$) были точно или примерно равны. В случае, когда точного равенства достичь не удается, разница между суммами должна быть минимальна;

б) кодовым комбинациям первой части дописывается 1, кодовым комбинациям второй части дописывается 0;

в) анализируют первую часть: если она содержит только один символ, работа с ней заканчивается, – считается, что код для ее символов построен, и выполняется переход к шагу г) для построения кода второй части. Если символов больше одного, переходят к шагу а) и процедура повторяется с первой частью как с самостоятельным упорядоченным списком;

г) анализируют вторую часть: если она содержит только один символ, работа с ней заканчивается и выполняется обращение к оставшемуся списку (шаг д). Если символов больше одного, переходят к шагу а) и процедура повторяется со второй частью как с самостоятельным списком;

д) анализируется оставшийся список: если он пуст – код построен, работа заканчивается. Если нет, – выполняется шаг а).

**Пример 12**. Даны символы *a, b, c, d* с частотами $f_a = 0{,}5$; $f_b = 0{,}25$; $f_c = 0{,}125$; $f_d = 0{,}125$. Построить эффективный код методом Шеннона-Фано. Сведем исходные данные в таблицу, упорядочив их по невозрастанию частот:

| Исходные символы | Частоты символов |
|---|---|
| *a* | 0,5 |
| *b* | 0,25 |
| *c* | 0,125 |
| *d* | 0,125 |

Первая линия деления проходит под символом *a*: соответствующие суммы $\Sigma_1$ и $\Sigma_2$ равны между собой и равны 0,5. Тогда формируемым кодовым комбинациям дописывается 1 для верхней (первой) части и 0 для нижней (второй) части. Поскольку это первый шаг формирования кода, двоичные цифры не дописываются, а только начинают формировать код:

| Исходные символы | Частоты символов | Формируемый код |
|---|---|---|
| *a* | 0,5 | 1 |
| *b* | 0,25 | 0 |
| *c* | 0,125 | 0 |
| *d* | 0,125 | 0 |

В силу того, что верхняя часть списка содержит только один элемент (символ *a*), работа с ней заканчивается, а эффективный код для этого символа считается сформированным (в таблице, приведенной выше, эта часть списка частот символов выделена заливкой). Второе деление выполняется под символом *b*: суммы частот $\Sigma_1$ и $\Sigma_2$ вновь равны между собой и равны 0,25. Тогда кодовой комбинации символов верхней части дописывается 1, а нижней части – 0. Таким образом, к полученным на первом шаге фрагментам кода, равным 0, добавляются новые символы:

| Исходные символы | Частоты символов | Формируемый код |
|---|---|---|
| *a* | 0,5 | 1 |
| *b* | 0,25 | 01 |
| *c* | 0,125 | 00 |
| *d* | 0,125 | 00 |

Поскольку верхняя часть нового списка содержит только один символ (*b*), формирование кода для него закончено (соответствующая строка таблицы вновь выделена заливкой). Третье деление проходит между символами *c* и *d*: к кодовой комбинации символа *c* приписывается 1, коду символа *d* приписывается 0:

| Исходные символы | Частоты символов | Формируемый код |
|---|---|---|
| *a* | 0,5 | 1 |
| *b* | 0,25 | 01 |
| *c* | 0,125 | 001 |
| *d* | 0,125 | 000 |

Поскольку обе оставшиеся половины исходного списка содержат по одному элементу, работа со списком в целом заканчивается.

Таким образом, получили коды:

*a* - 1, *b* - 01, *c* - 001, *d* - 000.

Определим эффективность построенного кода по формуле:

$$I_{cp} = 0,5*1 + 0,25*01 + 0,125*3 + 0,125*3 = 1,75.$$

Поскольку при кодировании четырех символов кодом постоянной длины требуется два двоичных разряда, сэкономлено 0,25 двоичного разряда в среднем на один символ.

***Метод Хаффмена.*** Этот метод имеет два преимущества по сравнению с методом Шеннона-Фано: он устраняет неоднозначность кодирования, возникающую из-за примерного равенства сумм частот при разделении списка на две части (линия деления проводится неоднозначно), и имеет, в общем случае, большую эффективность кода. Исходное множество символов упорядочивается по не возрастанию частоты и выполняются следующие шаги:

1) объединение частот: две последние частоты списка складываются, а соответствующие символы исключаются из списка; оставшийся после исключения символов список пополняется суммой частот и вновь упорядочивается; предыдущие шаги повторяются до тех пор, пока ни получится единица в результате суммирования и список ни уменьшится до одного символа;

2) построение кодового дерева: строится двоичное кодовое дерево: корнем его является вершина, полученная в результате объединения частот, равная 1; листьями – исходные вершины; остальные вершины соответствуют либо суммарным, либо исходным частотам, причем для каждой вершины левая подчиненная вершина соответствует большему слагаемому, а правая – меньшему; ребра дерева связывают вершины-суммы с вершинами-слагаемыми. Структура дерева показывает, как происходило объединение частот; ребра дерева кодируются: каждое левое кодируется единицей, каждое правое – нулём;

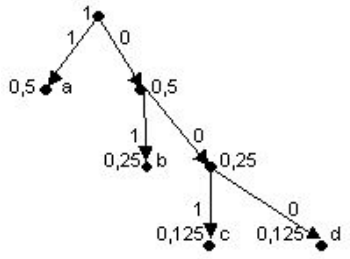
3) формирование кода: для получения кодов листьев (исходных кодируемых символов) продвигаются от корня к нужной вершине и «собирают» веса проходимых рёбер.

**Пример 13**. Даны символы *a, b, c, d* с частотами $f_a = 0,5$; $f_b = 0,25$; $f_c = 0,125$; $f_d = 0,125$. Построить эффективный код методом Хаффмена.

1) объединение частот (результат объединения двух последних частот в списке выделен в правом соседнем столбце заливкой):

| Исходные символы | Частоты $f_s$ | Этапы объединения | | |
|---|---|---|---|---|
| | | первый | Второй | третий |
| a | 0,5 | 0,5 | 0,5 | 1 |
| b | 0,25 | 0,25 | 0,5 | |
| c | 0,125 | 0,25 | | |
| d | 0,125 | | | |

2) построение кодового дерева:




3) формирование кода:

*a* - 1; *b* - 01; *c* - 001;*d* -000.

Как видно, полученные коды совпадают с теми, что были сформированы методом Шеннона-Фано, следовательно, они имеют одинаковую эффективность.

***Повышение эффективности кодирования***

Повысить эффективность кодирования можно, строя код не для символа, а для блоков из n символов, причем частота блока рассчитывается как произведение частот символов, входящих в блок. Рассмотрим этот тезис на примере.

**Пример 14**. Даны символы a и b с частотами, соответственно, 0,9 и 0,1. Построить эффективный код методом Шеннона-Фано для блоков из двух символов ($n$ = 2). Сформируем список возможных блоков и их частот. При этом частоту блока будем рассчитывать как произведение частот символов, входящих в блок. Тогда имеем:

***Блоки исходных символов*** ***Частоты блоков***

| | |
|---|---|
| aa | 0,81 |
| ab | 0,09 |
| ba | 0,09 |
| bb | 0,01 |

Построение кода сведём в таблицу:

| Блоки исходных символов | Частоты блоков | Этапы построения кода | | |
|---|---|---|---|---|
| | | первый | Второй | третий |
| *Aa* | 0,81 | 1 | код построен | |
| *Ab* | 0,09 | 0 | 1 | код построен |
| *Ba* | 0,09 | 0 | 0 | 1 |
| *Bb* | 0,01 | 0 | 0 | 0 |

Таким образом, получены коды:

*aa* – 1; *ab* - 01; *ba* - 001; *bb* - 000.

Определим эффективность построенного кода. Для этого рассчитаем сначала показатель эффективности для блока символов: $I_{ср}^{\text{блока}} = 0{,}81 \cdot 1 + 0{,}09 \cdot 2 + 0{,}09 \cdot 3 + 0{,}01 \cdot 3 = 1{,}28$. Поскольку в блоке 2 символа ($n$=2), для одного символа $I_{ср}=I_{ср}^{блока}/2 = 1{,}28/2 = 0{,}64$. При посимвольном кодировании для эффективного кода потребуется по одному двоичному разряду. В самом деле, применение метода Шеннона-Фано даёт результат, представленный в таблице:

| Исходные символы | Частоты символов | Построение кода |
|---|---|---|
| *a* | 0,9 | 1 |
| *b* | 0,1 | 0 |

Таким образом, при блочном кодировании выигрыш составил 1 - 0,64 = 0,36 двоичных разрядов на один кодируемый символ в среднем.

Эффективность блочного кодирования тем выше, чем больше символов включается в блок

***Декодирование эффективных кодов***

Особенностью эффективных кодов является переменное число двоичных разрядов в получаемых кодовых комбинациях. Это затрудняет процесс декодирования.

Рассмотрим вначале, как происходит декодирование сообщения, если использовались коды постоянной длины.

Пусть кодовая таблица имеет вид:

| Исходные символы | Двоичные коды |
|---|---|
| *a* | 00 |
| *b* | 01 |
| *c* | 10 |
| *d* | 11 |

а закодированное сообщение - 001000011101.

Поскольку длина кода равна двум символам, в этом сообщении слева направо выделяются по два двоичных символа и сопоставляются с кодовой таблицей.

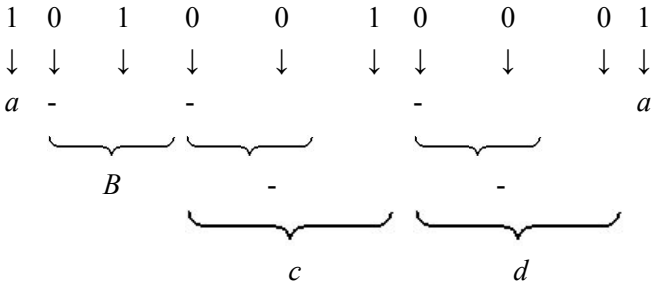
Тогда имеем:

| | | | | | |
|---|---|---|---|---|---|
| 00 | 10 | 00 | 10 | 11 | 01 |
| *a* | *c* | *a* | *b* | *d* | *b* |

Таким образом, в исходном сообщении содержится текст *acabdb.* Декодирование выполнено. Для декодирования кодов переменной длины рассмотренный подход не годится. Но закодированные сообщения могут декодироваться благодаря свойству **префиксности** эффективных кодов: ни одна более короткая кодовая комбинация не является началом более длинной кодовой комбинации. Для раскрытия данного тезиса воспользуемся построенными ранее эффективными кодами:
*a* -1; *b* - 01; *c* - 001; *d* - 000.
Здесь самым коротким кодом является код для символа *a* со значением 1. Как видно, ни один другой код (более длинный) не имеет в начале символ 1. Второй по длине код для символа *b* имеет значение 01 и, как показывает анализ, не является началом ни для кода 001, ни для кода 000. Таким образом, данный код является префиксным. Свойство префиксности позволяет декодировать сообщения, закодированные эффективными кодами. Пусть получено сообщение 1010010001, составленное из кодов
*a* - 1; *b* - 01; *c* - 001; *d* - 000.

Выполним его декодирование.

В сообщении слева направо выделяется по одному двоичному символу и делается попытка декодирования в соответствии с заданной таблицей кодов. Если попытка успешна, двоичный символ (или символы) исключается из исходной цепочки и заменяется соответствующим исходным символом. Если попытка не удается, во входной цепочке выделяется следующий двоичный символ и уже с двумя двоичными символами делается попытка их декодирования по таблице кодов. Если попытка и тогда неудачна, выделяют следующий третий и т.д.
Итак, имеем (направление просмотра цепочки слева направо):




Здесь знак «-» означает, что попытка декодирования не удалась. Таким образом, при декодировании получили строку *abcda.*
Отметим, что методы Шеннона-Фано и Хаффмена строят префиксные коды.

Помимо рассмотренных универсальных методов эффективного кодирования на практике часто применяются методы, ориентированные на конкретные виды сообщений. В зависимости от типа исходного сообщения они делятся на методы эффективного кодирования (сжатия) числовых последовательностей, словарей, естественно-языковых текстов.

***Помехозащитное кодирование.*** Этот вид кодирования применяется для обнаружения и/или исправления ошибок, которые могут возникнуть в дискретном сигнале при его передаче по каналам связи. В качестве базового кода, который подвергается помехозащитному кодированию, используется двоичный код постоянной длины. Такой исходный (базовый) код называется **первичным**, поскольку подвергается модификации. Для понимания сущности вопроса рассмотрим, как происходит искажение кодовых комбинаций при наличии помех в каналах связи.

**Пример 15.** Рассмотри таблицу:

| Буква | Код |
|---|---|
| А | 0 |
| М | 1 |

Напиши в «Блокноте» слово, используя код: МАМА
*Ответ к примеру 15*. 1010
**Пример 16.** Мартышка написала записку. Что здесь написано? Запиши.

| 4 | 2 | 3 | 0 | 9 | 4 | 9 | 1 | 3 | 5 | 9 | 6 | 3 | 1 | 8 | 6 | 4 | 10 | 7 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |

| | | |
|---|---|---|
| С | 4+2-1 | |
| И | 10-1-1 | |
| Р | 2+2-2 | |
| Н | 5+6-10 | |
| Ы | 3+3+1 | |

| | | |
|---|---|---|
| У | 9+1-6 | |
| Л | 4+7-1 | |
| К | 4+3-1 | |
| | 1+5+3 | |
| А | 8-3-2 | |
| ! | 10-7-3 | |

*Ответы к заданию 16*. Ура! У нас каникулы!

Рассмотри таблицу:

| А | Б | В | Г | Д | Е | Ё | Ж | З | И | Й | К | Л | М | Н | О | П |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |

| Р | С | Т | У | Ф | Х | Ц | Ч | Ш | Щ | Ь | Ы | Ъ | Э | Ю | Я |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |

**Пример 16.** Составь слова по номерам букв в алфавите и запиши в "Блокнот"

| 14 | 29 | 26 | 12 | 1 |
|---|---|---|---|---|
| | | | | |

| 14 | 16 | 15 | 10 | 20 | 16 | 18 |
|---|---|---|---|---|---|---|
| | | | | | | |

*Ответы к примеру 16.* Мышка Монитор

**Пример 17.** Продолжи:

| К | Л | А | В | И | А | Т | У | Р | А |
|---|---|---|---|---|---|---|---|---|---|
| 12 | | | | | | | | | |

| П | А | М | Я | Т | Ь |
|---|---|---|---|---|---|
| 17 | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 15 | | | | | | | | 33 |

| | | |
|---|---|---|
| | | Д |
| 12 | | |

*Ответы к примеру 17*. Клавиатура - 12 13 1 3 10 1 20 21 18 1. Память - 17 1 14 33 20 28. Информация - 10 15 22 16 18 14 1 24 10 33. Код - 12 16 5

# 3. КОНКРЕТНЫЕ МЕТОДЫ КОДИРОВАНИЯ

## *3.1 Алфавитное неравномерное двоичное кодирование*

Как следует из названия, символы некоторого первичного алфавита (например, русского) кодируются комбинациями символов двоичного алфавита (т.е. 0 и 1), причем, длина кодов и, соответственно, длительность передачи отдельного кода, могут различаться. Длительности элементарных сигналов при этом одинаковы ($\tau_0 = \tau_1 = \tau$). За счет чего можно оптимизировать кодирование в этом случае? Очевидно, суммарная длительность сообщения будет меньше, если применить следующий подход: тем буквам первичного алфавита, которые встречаются чаще, присвоить более короткие по длительности коды, а тем, относительная частота которых меньше – коды более длинные. Но длительность кода – величина дискретная, она кратна длительности сигнала $\tau$ передающего один символ двоичного алфавита. Следовательно, коды букв, вероятность появления которых в сообщении выше, следует строить из возможно меньшего числа элементарных сигналов. Построим кодовую таблицу для букв русского алфавита, основываясь на приведенных ранее вероятностях появления отдельных букв.

Очевидно, возможны различные варианты двоичного кодирования, однако, не все они будут пригодны для практического использования – важно, чтобы закодированное сообщение могло быть однозначно декодировано, т.е. чтобы в последовательности 0 и 1, которая представляет собой многобуквенное кодированное сообщение, всегда можно было бы различить обозначения отдельных букв. Проще всего этого достичь, если коды будут разграничены разделителем – некоторой постоянной комбинацией двоичных знаков. Условимся, что разделителем отдельных кодов букв будет

последовательность 00 (признак конца знака), а разделителем слов – 000 (признак конца слова – пробел). Довольно очевидными оказываются следующие правила построения кодов:
- код признака конца знака может быть включен в код буквы, поскольку не существует отдельно (т.е. кода всех букв будут заканчиваться 00);
- коды букв не должны содержать двух и более нулей подряд в середине (иначе они будут восприниматься как конец знака);
- код буквы (кроме пробела) всегда должен начинаться с 1;
- разделителю слов (000) всегда предшествует признак конца знака; при этом реализуется последовательность 00000 (т.е. если в конце кода встречается комбинация …000 или …0000, они не воспринимаются как разделитель слов); следовательно, коды букв могут оканчиваться на 0 или 00 (до признака конца знака).

Длительность передачи каждого отдельного кода $t_i$, очевидно, может быть найдена следующим образом: $t_i=k_i \cdot \tau$, где $k_i$ – количество элементарных сигналов (бит) в коде символа $i$. В соответствии с приведенными выше правилами получаем следующую таблицу кодов:

**Табл. 2**. Таблица кодов

| Буква | Код | $p_i \cdot 103$ | $k_i$ | Буква | Код | $p_i \cdot 10^3$ | $k_i$ |
|---|---|---|---|---|---|---|---|
| пробел | 000 | 174 | 3 | я | 1011000 | 18 | 7 |
| о | 100 | 90 | 3 | ы | 1011100 | 16 | 7 |
| е | 1000 | 72 | 4 | з | 1101000 | 16 | 7 |
| а | 1100 | 62 | 4 | ь,ъ | 1101100 | 14 | 7 |
| и | 10000 | 62 | 5 | б | 1110000 | 14 | 7 |
| т | 10100 | 53 | 5 | г | 1110100 | 13 | 7 |
| н | 11000 | 53 | 5 | ч | 1111000 | 12 | 7 |
| с | 11100 | 45 | 5 | й | 1111100 | 10 | 7 |
| р | 101000 | 40 | 6 | х | 10101000 | 9 | 8 |
| в | 101100 | 38 | 6 | ж | 10101100 | 7 | 8 |
| л | 110000 | 35 | 6 | ю | 10110000 | 6 | 8 |
| к | 110100 | 28 | 6 | ш | 10110100 | 6 | 8 |
| м | 111000 | 26 | 6 | ц | 10111000 | 4 | 8 |
| д | 111100 | 25 | 6 | щ | 10111100 | 3 | 8 |
| п | 1010000 | 23 | 7 | э | 11010000 | 3 | 8 |
| у | 1010100 | 21 | 7 | ф | 11010100 | 2 | 8 |

Теперь по формуле можно найти среднюю длину кода $K^{(2)}$ для данного способа кодирования:

$$K^{(2)} = \sum_{i=1}^{32} p_i \cdot k_i = 4{,}964 \quad (12)$$

Поскольку для русского языка, $I_1(r)$=4,356 бит, избыточность данного кода, согласно (2), составляет:

$$Q(r) = 1 - 4{,}356/4{,}964 \approx 0{,}122; \quad (13)$$

это означает, что при данном способе кодирования будет передаваться приблизительно на 12% больше информации, чем содержит исходное сообщение. Аналогичные вычисления для английского языка дают значение $K(2) = 4{,}716$, что при $I_1(e) = 4{,}036$ бит приводят к избыточности кода $Q(e) = 0{,}144$.

Рассмотрев один из вариантов двоичного неравномерного кодирования, попробуем найти ответы на следующие вопросы: возможно ли такое кодирование без использования разделителя знаков? Существует ли наиболее оптимальный способ неравномерного двоичного кодирования?

Суть первой проблемы состоит в нахождении такого варианта кодирования сообщения, при котором последующее выделение из него каждого отдельного знака (т.е. декодирование) оказывается однозначным без специальных указателей разделения знаков. Наиболее простыми и употребимыми кодами такого типа являются так называемые префиксные коды, которые удовлетворяют следующему условию (Фано): Неравномерный код может быть однозначно декодирован, если никакой из кодов не совпадает с началом

(префиксом) какого-либо иного более длинного кода. Например, если имеется код 110, то уже не могут использоваться коды 1, 11, 1101, 110101 и пр. Если условие Фано выполняется, то при прочтении (расшифровке) закодированного сообщения путем сопоставления со списком кодов всегда можно точно указать, где заканчивается один код и начинается другой.

**Пример 18**. Пусть имеется следующая таблица префиксных кодов:

**Табл. 17**. Таблица кодов

| а | л | м | р | у | ы |
|---|---|---|---|---|---|
| 10 | 010 | 00 | 11 | 0110 | 0111 |

Требуется декодировать сообщение: 00100010000111010101110000110. Декодирование производится циклическим повторением следующих действий. Отрезать от текущего сообщения крайний левый символ, присоединить к рабочему кодовому слову. Сравнить рабочее кодовое слово с кодовой таблицей; если совпадения нет, перейти к (1). Декодировать рабочее кодовое слово, очистить его. Проверить, имеются ли еще знаки в сообщении; если "да", перейти к (1).

Применение данного алгоритма даёт:

| Шаг | Рабочее слово | Текущее сообщение | Распознанный знак | Декодированное сообщение |
|---|---|---|---|---|
| 0 | пусто | 00100010000111010101110000110 | – | – |
| 1 | 0 | 01000100001110101011100001 10 | нет | – |
| 2 | 00 | 10001000011101010111000011 0 | м | м |
| 3 | 1 | 00010000111010101110000110 | нет | м |
| 4 | 10 | 00100001110101011100001 10 | а | ма |
| 5 | 0 | 01000011101010111000011 0 | нет | ма |
| 6 | 00 | 10000111010101110000110 | м | мам |
| ... | | | | |

Доведя процедуру до конца, получим сообщение: «мама мыла раму».

Таким образом, использование префиксного кодирования позволяет делать сообщение более коротким, поскольку нет необходимости передавать разделители знаков. Однако условие Фано не устанавливает способа формирования префиксного кода и, в частности, наилучшего из возможных.

Способ оптимального префиксного двоичного кодирования был предложен Д.Хаффманом. Построение кодов Хаффмана мы рассмотрим на следующем примере: пусть имеется первичный алфавит А, состоящий из шести знаков $a_1, ..., a_6$ с вероятностями появления в сообщении, соответственно, 0,3; 0,2; 0,2; 0,15; 0,1; 0,05. Создадим новый вспомогательный алфавит $A_1$, объединив два знака с наименьшими вероятностями ($a_5$ и $a_6$) и заменив их одним знаком (например, $a_1$); вероятность нового знака будет равна сумме вероятностей тех, что в него вошли, т.е. 0,15; остальные знаки исходного алфавита включим в новый без изменений; общее число знаков в новом алфавите, очевидно, будет на 1 меньше, чем в исходном. Аналогичным образом продолжим создавать новые алфавиты, пока в последнем не останется два знака; ясно, что число таких шагов будет равно $N - 2$, где $N$ – число знаков исходного алфавита (в нашем случае $N = 6$, следовательно, необходимо построить 4 вспомогательных алфавита). В промежуточных алфавитах каждый раз будем переупорядочивать знаки по убыванию вероятностей. Всю процедуру построения представим в виде таблицы:

| № знака | Вероятности | | | | |
|---|---|---|---|---|---|
| | Исходный алфавит | Промежуточные алфавиты | | | |
| | $A^{(0)}$ | $A^{(1)}$ | $A^{(2)}$ | $A^{(3)}$ | $A^{(4)}$ |
| 1 | 0,3 | 0,3 | 0,3 | 0,4 | 0,6 |
| 2 | 0,2 | 0,2 | 0,3 | 0,3 | 0,4 |
| 3 | 0,2 | 0,2 | 0,2 | 0,3 | |
| 4 | 0,15 | 0,15 | 0,2 | | |
| 5 | 0,1 | 0,15 | | | |
| 6 | 0,05 | | | | |

Теперь в обратном направлении поведем процедуру кодирования. Двум знакам последнего алфавита присвоим коды 0 и 1 (которому какой – роли не играет; условимся, что верхний знак будет иметь код 0, а нижний – 1). В нашем примере знак $a_1(4)$ алфавита $A(4)$, имеющий вероятность 0,6 , получит код 0, а $a_2(4)$ с вероятностью 0,4 – код 1. В алфавите $A(3)$ знак $a_1(3)$ с вероятностью 0,4 сохранит свой код (1); коды знаков $a_2(3)$ и $a_3(3)$, объединенных знаком $a_1(4)$ с вероятностью 0,6 , будут уже двузначным: их первой цифрой станет код связанного с ними знака (т.е. 0), а вторая цифра – как условились – у верхнего 0, у нижнего – 1; таким образом, $a_2(3)$ будет иметь код 00, а $a_3(3)$ – код 01. Полностью процедура кодирования представлена в следующей таблице:


Из самой процедуры построения кодов легко видеть, что они удовлетворяют условию Фано и, следовательно, не требуют разделителя. Средняя длина кода при этом оказывается:

$$K^{(2)} = 0{,}3\cdot2+0{,}2\cdot2+0{,}2\cdot2+0{,}15\cdot3+0{,}1\cdot4+0{,}05\cdot4 = 2{,}45. \qquad (14)$$

Для сравнения можно найти $I_1^{(A)}$ – она оказывается равной 2,409, что соответствует избыточности кода $Q$ = 0,0169, т.е. менее 2%.

Код Хаффмана важен в теоретическом отношении, поскольку можно доказать, что он является самым экономичным из всех возможных, т.е. ни для какого метода алфавитного кодирования длина кода не может оказаться меньше, чем код Хаффмана.

Применение описанного метода для букв русского алфавита дает следующие коды:

**Табл. 4**. Таблица кодов русских букв

| Буква | Код | $p_i\cdot10^3$ | $k_i$ | Буква | Код | $p_i\cdot10^3$ | $k_i$ |
|---|---|---|---|---|---|---|---|
| пробел | 000 | 174 | 3 | я | 0011001 | 18 | 6 |
| о | 111 | 90 | 3 | ы | 0101100 | 16 | 6 |
| е | 0100 | 72 | 4 | з | 010111 | 16 | 6 |
| а | 0110 | 62 | 4 | ь,ъ | 100001 | 14 | 6 |
| и | 0111 | 62 | 4 | б | 101100 | 14 | 6 |
| т | 1001 | 53 | 4 | г | 101101 | 13 | 6 |
| н | 1010 | 53 | 5 | ч | 110011 | 12 | 6 |
| с | 1101 | 45 | 4 | й | 0011001 | 10 | 7 |
| р | 00101 | 40 | 5 | х | 1000000 | 9 | 7 |
| в | 00111 | 38 | 5 | ж | 1000001 | 7 | 7 |
| л | 01010 | 35 | 5 | ю | 1100101 | 6 | 7 |
| к | 10001 | 28 | 5 | ш | 00110000 | 6 | 8 |
| м | 10111 | 26 | 5 | ц | 11001000 | 4 | 8 |
| д | 11000 | 25 | 5 | щ | 11001001 | 3 | 8 |
| п | 001000 | 23 | 6 | э | 001100010 | 3 | 9 |
| у | 001001 | 21 | 6 | ф | 001100011 | 2 | 9 |

Средняя длина кода оказывается равной $K(2)$ = 4,395; избыточность кода $Q(r)$ = 0,00887, т.е. менее 1%.

Таким образом, можно заключить, что существует метод построения оптимального неравномерного алфавитного кода. Не следует думать, что он представляет число теоретический интерес. Метод Хаффмана и его модификация – метод адаптивного кодирования (динамическое кодирование Хаффмана) – нашли широчайшее применение в программах-архиваторах, программах резервного копирования файлов и дисков, в системах сжатия информации в модемах и факсах.

**Кодирование энтропии** - кодирования словами (кодами) переменной длины, при которой длина кода символа имеет обратную зависимость от вероятности появления символа в передаваемом сообщении. Обычно энтропийные

кодировщики используют для сжатия данных коды, длины которых пропорциональны отрицательному логарифму вероятности символа. Таким образом, наиболее вероятные символы используют наиболее короткие коды.

## 3.2 Равномерное алфавитное двоичное кодирование

В этом случае двоичный код первичного алфавита строится цепочками равной длины, т.е. со всеми знаками связано одинаковое количество информации равное $I_0$. Передавать признак конца знака не требуется, поэтому для определения длины кодовой цепочки можно воспользоваться формулой: $K^{(2)} \geq \log_2 N$. Приемное устройство просто отсчитывает оговоренное заранее количество элементарных сигналов и интерпретирует цепочку (устанавливает, какому знаку она соответствует). Правда, при этом недопустимы сбои, например, пропуск (непрочтение) одного элементарного сигнала приведет к сдвигу всей кодовой последовательности и неправильной ее интерпретации; решается проблема путем синхронизации передачи или иными способами. С другой стороны, применение равномерного кода оказывается одним из средств контроля правильности передачи, поскольку факт поступления лишнего элементарного сигнала или, наоборот, поступление неполного кода сразу интерпретируется как ошибка.

Примером равномерного алфавитного кодирования является телеграфный код Бодо, пришедший на смену азбуке Морзе. Исходный алфавит должен содержать не более 32-х символов; тогда $K^{(2)} = \log_2 32 = 5$, т.е. каждый знак содержит 5 бит информации. Условие $N \leq 32$, очевидно, выполняется для языков, основанных на латинском алфавите ($N = 27 = 26$+"пробел"), однако в русском алфавите 34 буквы (с пробелом) – именно по этой причине пришлось "сжать" алфавит (как в коде Хаффмана) и объединить в один знак "е" и "ё", а также "ь" и "ъ". После такого сжатия $N = 32$, однако, не остается свободных кодов для знаков препинания, поэтому в телеграммах они отсутствуют или заменяются буквенными аббревиатурами; это не является заметным ограничением, поскольку, как указывалось выше, избыточность языка позволяет легко восстановить информационное содержание сообщения. Избыточность кода Бодо для русского языка $Q^{(r)} = 0{,}129$, для английского $Q^{(e)} = 0{,}193$.

Другим важным для нас примером использования равномерного алфавитного кодирования является представление символьной информации в компьютере. Чтобы определить длину кода, необходимо начать с установления количество знаков в первичном алфавите. Компьютерный алфавит должен включать:

26×2=52 букв латинского алфавита (с учетом прописных и строчных);

33×2=66 букв русского алфавита;

цифры 0...9 – всего 10;

знаки математических операций, знаки препинания, спецсимволы ≈20.

Получаем, что общее число символов $N$=148. Теперь можно оценить длину кодовой цепочки: $K^{(2)} \geq \log_2 148 \geq 7{,}21$. Поскольку $K^{(2)}$ должно быть целым, очевидно, $K^{(2)} = 8$. Именно такой способ кодирования принят в компьютерных системах: любому символу ставится в соответствие цепочка из 8 двоичных разрядов (8 бит). Такая цепочка получила название *байт*, а представление таким образом символов – *байтовым кодированием*.

Байт наряду с битом может использоваться как единица измерения количества информации в сообщении. Один байт соответствует количеству информации в одном символе алфавита при их равновероятном распределении. Этот способ измерения количества информации называется также объёмным. Пусть имеется некоторое сообщение (последовательность знаков); оценка количества содержащейся в нём информации согласно рассмотренному ранее вероятностному подходу (с помощью формулы Шеннона) даёт $I_{вер}$, а объемная мера пусть равна $I_{об}$; соотношение между этими величинами: $I_{вер} \leq I_{об}$

Именно байт принят в качестве единицы измерения количества информации в международной системе единиц СИ. 1 байт = 8 бит. Использование 8-битных цепочек позволяет закодировать $2^8$=256 символов, что превышает оцененное выше $N$ и, следовательно, дает возможность употребить оставшуюся часть кодовой таблицы для представления дополнительных символов.

Однако недостаточно только условиться об определенной длине кода. Ясно, что способов кодирования, т.е. вариантов сопоставления знакам первичного алфавита восьмибитных цепочек, очень много. По этой причине для совместимости технических устройств и обеспечения возможности обмена информацией между многими потребителями требуется согласование кодов. Подобное согласование осуществляется в форме стандартизации кодовых таблиц. Первым таким международным стандартом, который применялся на и телекоммуникационных системах применяется международный байтовый код больших вычислительных машинах, был *EBCDIC> (Extended Binary Coded Decimal Interchange Code*) – «***расширенная двоичная*** *кодировка десятичного кода обмена»*. В персональных компьютерах *ASCII*

*(American Standard Code for Information Interchange* – «***американский стандартный код обмена информацией***»). Он регламентирует коды первой половины кодовой таблицы (номера кодов от 0 до 127, т.е. первый бит всех кодов 0). В эту часть попадают коды прописных и строчных английских букв, цифры, знаки препинания и математических операций, а также некоторые управляющие коды (номера от 0 до 31). Ниже приведены некоторые *ASCII*-коды:

**Табл. 5**. Некоторые *ASCII*-коды

| Знак, клавиша | Код двоичный | Код десятичный |
|---|---|---|
| пробел | 00100000 | 32 |
| *A* (лат) | 01000001 | 65 |
| *B* (лат) | 01000010 | 66 |
| *Z* | 01011010 | 90 |
| 0 | 00110000 | 48 |
| 1 | 00110001 | 49 |
| 9 | 00111001 | 57 |
| Клавиша *ESC* | 00011011 | 27 |
| Клавиша *Enter* | 00001101 | 13 |

Вторая часть кодовой таблицы – она считается расширением основной – охватывает коды в интервале от 128 до 255 (первый бит всех кодов 1). Она используется для представления символов национальных алфавитов (например, русского или греческого), а также символов псевдографики. Для этой части также имеются стандарты, например, для символов русского языка это *КОИ–8, КОИ–7* и др.

Как в основной таблице, так и в её расширении коды букв и цифр соответствуют их лексикографическому порядку (т.е. порядку следования в алфавите) – это обеспечивает возможность автоматизации обработки текстов и ускоряет ее.

В настоящее время появился и находит все более широкое применение ещё один международный стандарт кодировки – *Unicode*. Его особенность в том, что в нем использовано 16-битное кодирование, т.е. для представления каждого символа отводится 2 байта. Такая длина кода обеспечивает включения в первичный алфавит 65536 знаков. Это, в свою очередь, позволяет создать и использовать единую для всех распространенных алфавитов кодовую таблицу.

## *3.3 Блочное двоичное кодирование*

Вернёмся к проблеме оптимального кодирования. Пока что наилучший результат (наименьшая избыточность) был получен при кодировании по методу Хаффмана – для русского алфавита избыточность оказалась менее 1%. При этом указывалось, что код Хаффмана улучшить невозможно. На первый взгляд это противоречит первой теореме Шеннона, утверждающей, что всегда можно предложить способ кодирования, при котором избыточность будет сколь угодно малой величиной. На самом деле это противоречие возникло из-за того, что до сих пор мы ограничивали себя алфавитным кодированием. При алфавитном кодировании передаваемое сообщение представляет собой последовательность кодов отдельных знаков первичного алфавита. Однако возможны варианты кодирования, при которых кодовый знак относится сразу к нескольким буквам первичного алфавита (будем называть такую комбинацию блоком) или даже к целому слову первичного языка. Кодирование блоков понижает избыточность. В этом легко убедиться на простом примере.

Пусть имеется словарь некоторого языка, содержащий $n$ = 16000 слов (это, безусловно, более чем солидный словарный запас!). Поставим в соответствие каждому слову равномерный двоичный код. Очевидно, длина кода может быть найдена из соотношения $K^{(2)} \geq \log_2 n \geq 13{,}97 = 14$. Следовательно, каждому слову будет поставлена в соответствие комбинация из 14 нулей и единиц – получатся своего рода двоичные иероглифы. Например, пусть слову "ИНФОРМАТИКА" соответствует код 10101011100110, слову "НАУКА" – 00000000000001, а слову "ИНТЕРЕСНАЯ" – 00100000000010; тогда последовательность:
00000000000011010101110011000000000000000001,
очевидно, будет означать *"ИНФОРМАТИКА ИНТЕРЕСНАЯ НАУКА"*.

Легко оценить, что при средней длине русского слова $K^{(r)}$ = 6,3 буквы (5,3 буквы + пробел между словами) средняя информация на знак первичного алфавита оказывается равной $I^{(2)}=K^{(2)}/K^{(r)}$ = 14/6,3 = 2,222 бит, что почти в 2 раза меньше, чем 4,395 бит при алфавитном кодировании. Для английского языка такой метод кодирования дает 2,545 бит на знак. Таким образом, кодирование слов оказывается более выгодным, чем алфавитное.

Ещё более эффективным окажется кодирование в том случае, если сначала установить относительную частоту появления различных слов в текстах и затем использовать код Хаффмана. По относительным частотам 8727 наиболее употребительных в английском языке слов Шеннон, что средняя информация на знак первичного алфавита оказывается равной 2,15 бит. Вместо слов можно кодировать сочетания букв – блоки. В принципе блоки можно считать словами равной длины, не имеющими, однако, смыслового содержания. Удлиняя блоки и применяя код Хаффмана можно добиться того, что средняя информация на знак кода будет сколь угодно приближаться к $I_\infty$. Однако, применение блочного и словесного метода кодирования имеет свои недостатки. 1) Необходимо хранить огромную кодовую таблицу и постоянно к ней обращаться при кодировании и декодировании, что замедлит работу и потребует значительных ресурсов памяти. 2) Помимо основных слов разговорный язык содержит много производных от них, например, падежи существительных в русском языке или глагольные формы в английском; в данном способе кодирования им всем нужно присвоить свои коды, что приведет к увеличению кодовой таблицы еще в несколько раз. 3) Возникает проблема согласования (стандартизации) этих громадных таблиц, что непросто. 4) Алфавитное кодирование имеет то преимущество, что буквами можно закодировать любое слово, а при кодировании слов – использовать только имеющийся словарный запас. По указанным причинам блочное и словесное кодирование представляет лишь теоретический интерес, на практике же применяется кодирование алфавитное.

## *3.4 Методы сжатия цифровой информации*

Характерной особенностью большинства «классических» типов информации, с которыми работают люди, является их избыточность.

**Пример 19.** 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 не сможет его понять. Мы, носители русского языка, можем с лёгкостью восстановить окончания, пропущенные буквы в слогах, подобрать подходящие слова. Для носителя языка обычный связный текст на его родном языке содержит избыточную информацию – её можно удалить, но смысл текста для него сохранится.

**Пример 20.** Одним из примеров проявления избыточности информации и её 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:

$$2 + 4 + 6 + 8 + 10 + 12 + 14 + 16 + 18 + 20 + 22 + \\ + 24 + 26 + 28 + 30 + 32 + 34 + 36 + 38 + 40.$$

Эта сумма составлена из всех чётных чисел от 2 до 40. Используя математическую нотацию, 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, то вторая понятна только тем, кто знает, как следует интерпретировать эту запись.

Пример с суммой позволяет продемонстрировать ещё одну особенность методов сжатия – степень сжатия входных данных 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 удаётся 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.

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 надёжность её восприятия, особенно в неблагоприятных условиях (просмотр телепередач при наличии помех, чтение тестов в условиях недостаточной освещённости, разговор в вагоне метро и т.п.).

Хранение и передача информации требуют определённой 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, придётся заплатить меньше денег за пользование Интернетом.

## 3.4.1 Алгоритмы обратимых методов сжатия информации

Все методы сжатия можно поделить на два больших класса. Одни алгоритмы только изменяют способ представления входных данных, приводя их форме, которая более компактно кодируется. Такие алгоритмы принято называть обратимыми, поскольку для них существуют обратные алгоритмы, способные 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 сжатию. Такие алгоритмы принято называть алгоритмы с регулируемой потерей информации.

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:

- *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** заключается в уменьшении количества бит, отводимых для кодирования символов, если в сжимаемом массиве данных присутствует только небольшая часть используемого алфавита.

**Пример 21.** Допустим, входной текст состоит из десятичной записи целых чисел и знаков «минус», разделённых пробела (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 четырёх 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 кодируются 4-битовыми кодами, называется *BCD* – форматом (*Binary Coded Decimal*, или двоично-десятичная запись). *BCD*- формат нередко используется для хранения целых чисел, например в базах данных.

**Пример 22.** 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 тексте 18 символов, так что потребуется 18·3=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≈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 байт, 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 даёт хорошие результаты только, если множество используемых символов невелико. Например, если в тесте используются только прописные русские буквы и знаки препинания, то текст может быть сжат всего на 25%: 33 русские буквы плюс пробел и знаки препинания – 40 символов. Для их кодирования достаточно 6 бит. При упаковке текст уменьшается до 6/8=3/4 от первоначального объёма.

Слабое место метода упаковки заключается в том, что символы кодируются битовыми последовательностями одинаковой длины. Например, любой текст, состоящий только из двух букв A и B, сжимается методом упаковки в восемь раз. Однако если к такому тексту добавить всего лишь одну букву, например, C, то степень сжатия сразу уменьшится вдвое, причём независимо от длины текста и количества добавленных символов C.

Улучшения степени сжатия можно достичь, кодируя часто встречающиеся символы короткими кодами, редко встречающиеся – более длинными. Именно такова идея метода Д. Хаффмана (1952). Алгоритм Хаффмана оказался простым, быстрым и оптимальным: среди алгоритмов, кодирующих каждый символ по отдельности и целым количеством бит, он обеспечивал наилучшее сжатие.

**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 ещё раз и при этом генерируется выходной массив данных.

**Пример 23.** Для текста КОЛ_ОКОЛО_КОЛОКОЛА код текста будет занимать 39 бит или 5 байт. Коэффициент сжатия равен 18/5=3,6.

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 передаваемого символа. Получателю пересылают только дерево Хаффмана в компактном виде, а затем входная последовательность кодов символов декодируется им самостоятельно без какой-либо дополнительной информации.

В основу **алгоритмов RLE** (*Run-Length Encoding* – кодирование путём учёта числа повторений) положен принцип выявления повторяющихся последовательностей данных и заменой их простой структурой. Алгоритмы Лемпеля-Зива тем лучше сжимают текст, чем больше размер входного массива. Характерной особенностью обратных алгоритмов *LZ*77 и *LZ*78 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 древо Хаффмана, иначе распаковка будет невозможна. В настоящее время существует полсотни модификаций этого алгоритма. Все они называются методами сжатия со словарём. Эти алгоритмы оказались настолько быстры и эффективны, что сейчас занимают лидирующее место среди используемых на практике алгоритмов сжатия.

### 3.4.2 Алгоритмы 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 не обязательно. Некоторые особенности человеческого восприятия позволяют существенно увеличить степень сжатия звуковой, графической и видеоинформации. Например, глаз человека наиболее чувствителен к зелёному цвету, чувствительность к красному ниже в 4 раза, а к синему – 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 идёт о воспроизведении видеоинформации для человека.

К середине 1990-х 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 алгоритм сжатия называют алгоритмами с регулируемой потерей информации. За счёт удаления части информации удаётся 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.

**Алгоритм *JPEG*** используется для сжатия статических изображений. Помимо сжимаемого изображения, алгоритму передаётся 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 (соответствующей ему цветовой модели *YCdCr* цвет представляется компонентами «яркость» *Y*, «цветоразность зелёный – красный» *Cr* и «цветоразность зелёный-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 объём данных вдвое. Оставшиеся данные подвергаются специальной процедуре «сглаживания», при котором объём данных не изменяется, но потенциальная степень их сжимаемости резко увеличивается. На этом этапе учитывается желаемый коэффициент сжатия. Затем данные сжимаются алгоритмом Хаффмана.

**Алгоритм *MP*3** является частью стандарта *MPEG* и описывает сжатие аудиоинформации. Помимо сжимаемого звукового фрагмента алгоритму передаётся также желаемый битрейт (*bitrate*) – количество бит, используемых для кодирования одной секунды звука. Этот параметр регулирует долю информации, которая будет удалена при сжатии.

Сжатие *MP*3 также осуществляется в несколько этапов: звуковой фрагмент разбивается на небольшие участки – фреймы (*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 MP№ обеспечивает четырёхкратное сжатие аудиоинформации по сравнению м форматом Audi XD, при таком же субъективном качестве звука. Формат MP3 стал стандартом де-факто для распространения музыкальных файлов через Интернет.

***MPEG*** – целое семейство методов сжатия видеоданных. В них используется очень большое количество приёмов сжатия. Они опираются на несколько базовых идей, а различаются конкретной реализацией алгоритмов. Одна из основных идей сжатия видео – метод «опорного кадра» - сохраняет не целиком видеокадры, 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 е кадров от первого, можно получить очень большую степень сжатия. Ещё один способ уменьшения кодируемой информации заключается в том, чтобы быстро меняемые 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 потоков данных. Так, в основном потоке можно сохранить фильм, в другом – логотип, в третьем – субтитры, и т.д. Потоки данных накладывают друг на друга только при воспроизведении. Такой способ позволяет, например, сохранить 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 – использовался 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.

## *3.5 Коды обнаружения и исправления ошибок*

Обнаружение ошибок - действие, направленное на контроль целостности данных при записи/воспроизведении информации. Исправление ошибок - процедура восстановления информации после чтения её из устройства хранения. Для обнаружения ошибок используют коды обнаружения ошибок, для исправления - корректирующие коды. В процессе хранения данных и передачи информации по сетям связи неизбежно возникают ошибки. Контроль целостности данных и исправление ошибок - важные задачи на многих уровнях работы с информацией.

Существует несколько стратегий борьбы с ошибками: 1) обнаружение ошибок в блоках данных и *автоматический запрос повторной передачи* повреждённых блоков - этот подход применяется в основном на канальном и транспортном уровнях; 2) обнаружение ошибок в блоках данных и отбрасывание повреждённых блоков - такой подход иногда применяется в системах потокового мультимедиа, где важна задержка передачи и нет времени на повторную передачу; 3) *исправление ошибок* применяется на физическом уровне.

***Корректирующие коды*** - коды, служащие для обнаружения или исправления ошибок, возникающих при передаче информации под влиянием помех, а также при её хранении.

Для этого при записи (передаче) в полезные данные добавляют специальным образом структурированную *избыточную* информацию (контрольное число), а при чтении (приёме) её используют для обнаружения или исправления ошибки. Число ошибок, которое можно исправить, ограничено и зависит от конкретного применяемого кода. С *кодами, исправляющими ошибки*, тесно связаны *коды обнаружения ошибок*. По способу работы с данными коды, исправляющие ошибки делятся на *блоковые*, делящие информацию на фрагменты постоянной длины и обрабатывающие каждый из них в отдельности, и *свёрточные*, работающие с данными как с непрерывным потоком. Хороший код должен удовлетворять, как минимум, следующим критериям: 1) способность исправлять как можно большее число ошибок, 2) как можно меньшая избыточность, 3) простота кодирования и декодирования.

Практически все используемые коды являются линейными. Это связано с тем, что нелинейные коды значительно сложнее исследовать, и для них трудно обеспечить приемлемую лёгкость кодирования и декодирования.

**Линейный блоковый код** - такой код, что множество его *кодовых слов* образует *k*-мерное линейное подпространство в *n*-мерном линейном пространстве, изоморфное пространству *k*-битных векторов. Это значит, что операция кодирования соответствует умножению исходного *k*-битного вектора на невырожденную матрицу, называемую *порождающей матрицей*.

**Коды Хемминга** - простейшие линейные коды с минимальным расстоянием 3, то есть способные исправить одну ошибку.

При передаче информации по каналу связи вероятность ошибки зависит от отношения сигнал/шум на входе демодулятора, поэтому при постоянном уровне шума решающее значение имеет мощность передатчика. В системах спутниковой или мобильной связи остро стоит вопрос экономии энергии, а в телефонной связи неограниченно повышать мощность сигнала не дают технические ограничения. Поскольку помехоустойчивое кодирование позволяет исправлять ошибки, при его применении мощность передатчика можно снизить, оставляя скорость передачи информации неизменной. Энергетический выигрыш определяется как разница отношений сигнал/шум при наличии и отсутствии кодирования.

Коды, исправляющие ошибки, применяются: 1) в системах цифровой связи, в том числе: спутниковой, радиорелейной, сотовой, передаче данных по телефонным каналам. 2) в системах хранения информации, в том числе магнитных и оптических. Коды, обнаруживающие ошибки, используются в сетевых протоколах различных уровней.