

Министерство образования и науки Российской Федерации  
Федеральное агентство по образованию  
Ярославский государственный университет им. П.Г. Демидова  
Кафедра компьютерных сетей

# Теория кодирования

*Методические указания*

*Рекомендовано*  
*Научно-методическим советом университета*  
*для студентов, обучающихся по специальности*  
*Прикладная математика и информатика*

Ярославль 2006

УДК 007  
ББК 181я73  
К 78

*Рекомендовано  
Редакционно-издательским советом университета  
в качестве учебного издания. План 2006 года*

Рецензент  
кафедра компьютерных сетей Ярославского государственного  
университета им. П.Г. Демидова

Составитель: М.В. Краснов

**Теория кодирования :** метод. указания / сост.  
К 78 М.В. Краснов; Яросл. гос. ун-т. – Ярославль : ЯрГУ, 2006. –  
48 с.

Основное использование вычислительной техники связано с хранением и передачей информации. При хранении информации возникает задача экономного метода записи. При передаче информации возникает задача ее защиты от случайных помех. Описанию некоторых математических понятий и приемов, используемых при решении этих задач, и посвящена данная работа.

Предназначено для студентов, обучающихся по специальности 010501 Прикладная математика и информатика (дисциплина «Теория информации и кодирование», блок ДС), очной формы обучения.

УДК 007  
ББК 181я73

© Ярославский государственный университет, 2006  
© М.В. Краснов, 2006

Под кодированием в широком смысле понимается переход от одного способа задания информации к другому, допускающий восстановление исходной информации. Для уточнения термина «кодирование», используемого в работе, остановимся на двух случаях, которые рассматриваются далее: это передача данных по каналу связи с помехами и передача информации по каналу без помех. В первом случае будем говорить о кодах, исправляющих ошибки, а во втором – о методах сжатия информации.

## 1. Коды, исправляющие ошибки

В этом разделе рассматриваются способы исправления ошибок (помех) при передаче информации. Наиболее действенный способ борьбы с помехами – введение избыточности, то есть, кроме информационных символов, сообщение должно содержать некоторое число контрольных символов, служащих для обнаружения и исправления ошибок.

### 1.1. Основные определения

Пусть имеется сообщение  $i = \{u_0, \dots, u_{k-1}\}$ , состоящее из символов алфавита  $A = \{0, \dots, q-1\}$ , которое должно быть передано по каналу связи. Из-за существования помех передача сообщения в чистом виде исключается. Поэтому сообщение  $i$  кодируется другой последовательностью  $c = \{c_0, \dots, c_{n-1}\}$ , состоящей из символов того же алфавита  $A = \{0, \dots, q-1\}$ , по которой можно восстановить  $i$ . Последовательность  $c$  называется в этом случае кодовым словом, а  $i$  – информационным словом.

*Определение 1.1* Блочным кодом  $\xi$  над алфавитом из  $q$  символов называется множество  $q$ -ичных последовательностей длины  $n$ . Мощность кода  $M$  – число этих последовательностей.

*Определение 1.2.* Блочный код  $\xi$  мощности  $q^k$  называется  $(n, k)$ -кодом.

При кодировании произвольной  $q$ -ичной последовательности  $(n, k)$ -кодом она разбивается на части из  $k$ -символов, и каждая часть кодируется элементом кода  $\xi$ .

О блоковом коде судят по трем параметрам: длине блока  $n$ , информационной длине  $k$  и минимальному расстоянию  $d^*$ . Минимальное расстояние вводится двумя следующими определениями.

*Определение 1.3.* Расстоянием по Хэммингу между двумя  $q$ -ичными последовательностями  $x$  и  $y$  длины  $n$  называется число позиций, в которых они различны. Это расстояние обозначается через  $d(x, y)$ .

*Определение 1.4.* Пусть  $\xi = \{c_i, i = 0, \dots, M-1\}$  – код. Тогда минимальное расстояние кода  $\xi$  равно наименьшему из всех расстояний по Хэммингу между различными парами кодовых слов, то есть  $d^* = \min_{c_i, c_j \in \xi, i \neq j} d(c_i, c_j)$ .

Если  $d^* \geq 2t+1$ , то код может исправлять  $t$  ошибок. Исправление ошибок в этом случае заключается в замене принятого слова на ближайшее кодовое слово.

## 1.2. Некоторые сведения из алгебры

*Определение 1.5.* Бинарной операцией на множестве  $M$  называется произвольная функция  $f: M \times M \rightarrow M$ .

*Определение 1.6.* Множество  $G$  с бинарной операцией  $*$  называется группой, если выполнены следующие аксиомы:

- ассоциативность.  $(a * b) * c = a * (b * c)$  для любых  $a, b, c \in G$ ;
- существует единичный элемент  $e \in G$  такой, что  $a * e = e * a = a$  для любого  $a \in G$ ;
- для любого  $a \in G$  существует обратный элемент  $b \in G$ , такой, что  $a * b = b * a = e$ .

Группа  $G$  называется абелевой группой, если  $a * b = b * a$  для любых  $a, b \in G$ .

Группа  $G$  называется циклической (с образующим элементом  $a$ ), если существует такой элемент  $a \in G$ , что любой элемент  $b \in G$  является некоторой степенью  $a$ .

*Определение 1.7.* Кольцом  $R$  называется множество с двумя определенными на нем бинарными операциями; первая называется сложением (обозначается  $+$ ), вторая называется умножением (обозначается  $*$ ), причем имеют место следующие аксиомы:

- относительно сложения  $(+)$   $R$  является абелевой группой ( $0$  – единичный элемент);
- замкнутость: произведение  $a * b$  принадлежит  $R$  для любых  $a, b \in R$ ;
- $(a * b) * c = a * (b * c)$  для любых  $a, b, c \in R$ ;
- существует элемент  $1 \in R$  такой, что  $a * 1 = 1 * a = a$  для любого элемента  $a \in R$ ;
- $(a + b) * c = (a * c) + (b * c)$  для любых  $a, b, c \in R$ .

*Определение 1.8.* Полем называется множество с двумя определенными на нем операциями – сложением и умножением, причем имеют место следующие аксиомы:

- множество образует абелеву группу по сложению ( $0$  – единичный элемент);
- поле замкнуто относительно умножения, и множество элементов ( $\neq 0$ ) образует абелеву группу по умножению ( $1$  – единичный элемент);
- закон дистрибутивности:  $(a + b) * c = (a * c) + (b * c)$  для любых  $a, b, c$  из поля.

*Определение 1.9.* Пусть  $F$  – некоторое поле. Подмножество  $S \subset F$  называется подполем, если оно само является полем относительно операций поля  $F$ . В этом случае поле  $F$  называется расширением поля  $S$ .

*Определение 1.10.* Множество  $V$  называется векторным пространством над полем  $F$  если

1) для пар элементов из  $V$  (векторов) определена операция векторного сложения;

2) для элемента из  $V$  и скаляра (элемент поля  $F$ ) определена операция умножения на скаляр,

то результат выполнения операций дает элемент из  $V$  и выполняются следующие аксиомы:

–  $V$  является абелевой группой относительно векторного сложения;

– где  $\forall a \in F$  и  $\forall v_1, v_2 \in V$ ;

–  $(a + b)v = av + bv$ , где  $\forall a, b \in F$  и  $\forall v \in V$ ;

–  $(ab)v = a(bv)$ , где  $\forall a, b \in F$  и  $\forall v \in V$ ;

–  $1v = v$ , где  $\forall v \in V$ .

*Определение 1.11.* Множество векторов  $\{v_1, \dots, v_n\}$  из пространства  $V$  называется базисом если

$$\forall u \in V \exists a_1, \dots, a_n \in F : u = a_1 v_1 + \dots + a_n v_n;$$

не существует таких  $a_1, \dots, a_n \in F$ , которые не все равны 0, а  $a_1 v_1 + \dots + a_n v_n = 0$ .

*Определение 1.12.* Векторным подпространством называется непустое подмножество векторного пространства, если оно также является векторным пространством относительно исходных операций.

### 1.3. Поля Галуа

Важнейшие идеи теории кодирования основаны на арифметических системах полей Галуа.

*Определение 1.13.* Конечное поле, состоящее из  $q$  элементов, обозначим через  $GF(q)$  и будем называть полем Галуа.

*Утверждение 1.1.* Группа ненулевых элементов любого поля Галуа  $GF(q)$  по умножению является циклической.

*Определение 1.14.* Примитивным элементом поля  $GF(q)$  называется такой элемент  $\alpha$ , что все элементы поля, за исключением нуля, могут быть представлены в виде степени элемента  $\alpha$ .

Для работы с конечными полями надо иметь таблицу сложения и таблицу умножения. Приведем примеры, где обе таблицы легко определяются.

*Пример 1.1.* Конечные поля, основанные на кольце целых чисел.

Пусть  $q$  – простое число. Тогда можно построить конечное поле с элементами  $\{0, \dots, q-1\}$ , если операции сложения и умножения определить равенствами:  
 $a + b = a + b \pmod q, \quad ab = ab \pmod q. \quad \square$

*Пример 1.2.* Конечные поля, основанные на кольцах многочленов.

Пусть имеется кольцо многочленов  $F[x]$  над полем  $F$ . Выбираем из  $F[x]$  произвольный многочлен  $p(x)$ , будем рассматривать только приведенные многочлены. Через  $F[x]/(p(x))$

обозначим кольцо остатков от деления на многочлен  $p(x)$ . Отметим, что кольцо многочленов по модулю приведенного многочлена  $p(x)$  является полем тогда и только тогда, когда многочлен  $p(x)$  прост<sup>1</sup>.

Для операции сложения каждый элемент поля Галуа  $GF(q^m)$  представляется как многочлен степени  $m$ , а операция их сложения сводится к сложению их коэффициентов – элементов поля  $GF(q)$ .

Для операции умножения каждый элемент поля Галуа  $GF(q^m)$  представляется через примитивный элемент, а операция умножения сводится к сложению показателей степени.  $\square$

*Определение 1.15.* Примитивным многочленом  $p(x)$  над полем Галуа  $GF(q)$  называется простой многочлен такой, что элемент  $x$  поля  $GF(q)[x]/(p(x))$  является примитивным.

Некоторые основные свойства полей Галуа:

- число элементов любого поля Галуа равно степени простого числа;

- для любого простого  $p$  и целого положительного  $m$  существует поле Галуа с  $p^m$  элементами, поле  $GF(p)$  является наименьшим подполем поля  $GF(p^m)$ , число  $p$  называется характеристикой поля  $GF(p^m)$ ;

- каждое поле Галуа  $GF(q)$  содержит хотя бы один примитивный элемент;

- над каждым полем Галуа существует хотя бы один примитивный многочлен любой положительной степени;

- два поля Галуа с одним и тем же числом элементов изоморфны.

Таким образом, для того, чтобы построить конечное поле из  $p^m$  элементов ( $p$ -простое число), достаточно найти неприводимый многочлен  $f(x)$  над полем  $GF(p)$  степени  $m$  и построить

---

<sup>1</sup> Напомним, что простой многочлен является одновременно неприводимым и приведенным. Для построения поля достаточно только неприводимости  $p(x)$ , но мы условились рассматривать только приведенные многочлены.

$GF(p)[x]/(f(x))$ . Многочлен  $f(x)$  будем называть порождающим многочленом поля.

*Пример 1.3.* Построим поле Галуа  $GF(8)$  как расширение поля  $GF(2)$  по примитивному многочлену  $x^3 + x + 1$ .

*Решение.*

Легко заметить, что поле  $GF(8)$  состоит из элементов  $\{0, 1, z, z+1, z^2, z^2+1, z^2+z, z^2+z+1\}$ .

В виде многочлена	В виде степени	В двоичном виде
0		000
1	$\alpha^7$	001
$z$	$\alpha$	010
$z+1$	$\alpha^3$	011
$z^2$	$\alpha^2$	100
$z^2+1$	$\alpha^6$	101
$z^2+z$	$\alpha^4$	110
$z^2+z+1$	$\alpha^5$	111

□

*Пример 1.4.* Построим поле Галуа  $GF(9)$  как расширение поля  $GF(3)$  по примитивному многочлену  $x^2 + x + 2$ .

*Решение.*

Легко заметить, что поле  $GF(9)$  состоит из элементов  $\{0, 1, 2, z, z+1, z+2, 2z, 2z+1, 2z+2\}$ .

В виде многочлена	В виде степени	В виде вектора
0		00
1	$\alpha^8$	01
2	$\alpha^4$	02
$z$	$\alpha$	10
$z+1$	$\alpha^7$	11
$z+2$	$\alpha^6$	12
$2z$	$\alpha^5$	20
$2z+1$	$\alpha^2$	21
$2z+2$	$\alpha^3$	22

□



## Простые многочлены над $GF(2)$ .

сте- пень	простой многочлен $p(x)$	сте- пень	простой многочлен $p(x)$	сте- пень	простой многочлен $p(x)$
2	$x^2 + x + 1$	7	$x^7 + x^3 + 1$	12	$x^{12} + x^6 + x^4 + x + 1$
3	$x^3 + x + 1$	8	$x^8 + x^4 + x^3 + x^2 + 1$	13	$x^{13} + x^4 + x^3 + x + 1$
4	$x^4 + x + 1$	9	$x^9 + x^4 + 1$	14	$x^{14} + x^{10} + x^6 + x + 1$
5	$x^5 + x^2 + 1$	10	$x^{10} + x^3 + 1$	15	$x^{15} + x + 1$
6	$x^6 + x + 1$	11	$x^{11} + x^2 + 1$	16	$x^{16} + x^{12} + x^3 + x + 1$

### 1.4. Минимальные многочлены

*Определение 1.16.* Пусть  $GF(q)$  – некоторое поле, пусть  $GF(q^m)$  – расширение поля  $GF(q)$  и пусть  $\beta \in GF(q^m)$ . Простой многочлен  $f(x)$  наименьшей степени над полем  $GF(q)$ , для которого  $f(\beta) = 0$ , называется минимальным многочленом элемента  $\beta$  над  $GF(q)$ .

*Утверждение 1.2.* Предположим, что  $f(x)$  над  $GF(q)$  является минимальным многочленом элемента  $\beta$  из  $GF(q^m)$ . Тогда  $f(x)$  является также минимальным многочленом элемента  $\beta^q$ . И кроме того  $f(x) = (x - \beta)(x - \beta^q) \dots (x - \beta^{q^{r-1}})$ , где  $r$  является наименьшим целым числом, таким, что  $\beta^{q^r} = \beta$ .

*Пример 1.5.* Найдем все минимальные многочлены в поле  $GF(9)$ , построенном как расширение поля  $GF(3)$  по примитивному многочлену  $x^2 + x + 2$ .

*Решение.*

Легко заметить, что

$$f_1(x) = (x - \alpha)(x - \alpha^3) = x^2 - (\alpha + \alpha^3)x + \alpha^4 = x^2 - 2x + 2 = x^2 + x + 2.$$

$$f_2(x) = (x - \alpha^2)(x - \alpha^6) = x^2 - (\alpha^2 + \alpha^6)x + \alpha^8 = x^2 + 1.$$

$$f_3(x) = (x - \alpha^4) = x - 2 = x + 1.$$

$$f_4(x) = (x - \alpha^5)(x - \alpha^7) = x^2 - (\alpha^5 + \alpha^7)x + \alpha^{12} = x^2 - x + \alpha^{12} = x^2 + 2x + 2.$$

$$f_5(x) = (x - \alpha^8) = x - 1 = x + 2. \square$$

## 1.5. Линейные блочные коды

Напомним, что векторное пространство  $GF^n(q)$  представляет собой множество  $n$ -последовательностей элементов из  $GF(q)$  с покомпонентным сложением и умножением на скаляр.

*Определение 1.17.* Линейный код есть подпространство в  $GF^n(q)$ .

Таким образом, линейный код есть непустое множество  $n$  – последовательностей над  $GF(q)$ , называемых кодовыми словами, такое, что сумма двух кодовых слов является кодовым словом и произведение любого кодового слова на элемент поля  $GF(q)$  также является кодовым словом. В линейном коде нулевое слово – кодовое слово.

*Определение 1.18.* Вес Хэмминга  $w(c)$  кодового слова  $c$  равен числу его ненулевых компонент.

*Утверждение 1.3.* Для линейного кода минимальное расстояние  $d^*$  находится из равенства  $d^* = \min_{c \neq 0} w(c)$ , где минимум берется по всем кодовым словам, кроме нулевого.

Обычно линейные блочные коды задаются с помощью порождающей матрицы  $G$  (размера  $k \times n$ ), которая устанавливает взаимно однозначное соответствие между информационными словами  $i$  и кодовыми словами  $c$ . Матрица  $G$  строится следующим образом: выбрав базис в подпространстве  $\xi$  линейного кода, записываем векторы базиса по строкам.

Наиболее естественный способ кодирования использует отображение  $c = iG$ , где  $i$  – информационное слово, а  $c$  – кодовое слово.

Другой матрицей, характеризующей линейные блочные коды, является проверочная матрица  $H$  (размера  $n - k \times n$ ). Матрица  $H$  называется проверочной потому, что выполняется условие  $cH^T = 0$ , для любого кодового слова  $c \in \xi$ . Матрица  $H$  позволяет проверить, является ли рассматриваемое слово кодовым. Ясно, что выполнимо равенство  $GH^T = 0$ .

Проверочная матрица позволяет определить минимальный вес кода.

*Утверждение 1.4.* Код  $\xi$  содержит ненулевое кодовое слово веса Хэмминга не более  $w$  тогда и только тогда, когда  $H$  содержит множество из  $w$  линейно зависимых столбцов.

*Утверждение 1.5.* Код имеет минимальный вес не менее  $w$  тогда и только тогда, когда каждое множество из  $w-1$  столбцов матрицы  $H$  линейно независимо.

Каждая порождающая матрица  $G$  эквивалентна некоторой матрице канонического ступенчатого вида  $G = [E_k, P]$ , где  $E_k$  – единичная матрица размера  $k$ . Тогда естественным определением проверочной матрицы в систематическом виде является равенство  $G = [-P^T, E_{n-k}]$ .

*Утверждение 1.6.* Минимальное расстояние  $d^*$  любого линейного  $(n, k)$ -кода удовлетворяет неравенству  $d^* \leq 1 + n - k$ .

### 1.5.1. Стандартное расположение

Напомним, что нулевое слово всегда будет кодовым. Пусть  $d^* = 2t + 1$ , тогда внутри сферы радиуса  $t$  с центром в нулевом слове находится множество точек  $S_0 = \{v \mid d(0, v) \leq t\}$ . Эта сфера содержит все принятые слова  $v$ , которые декодируются в нулевое кодовое слово. Аналогично можно описать сферы для каждого кодового слова  $S_c = \{v \mid d(c, v) \leq t\}$  или  $S_c = S_{0+c} = \{v + c \mid v \in S_0\}$ .

Стандартное расположение представляет собой способ описания всех этих сфер. Пусть  $0, c_2, \dots, c_{q^k}$  – кодовые слова  $(n, k)$  – кода. Следующим способом построим таблицу:

- в первой строке выпишем все кодовые слова;
- для заполнения  $j$ -строки выбирается слово  $v_l$ , которое является ближайшим к нулевому кодовому слову и отсутствует в предыдущих строках таблицы. После этого в  $j$ -строке записываются векторы  $0 + v_l, c_2 + v_l, \dots, c_{q^k} + v_l$ ;

- процедура заполнения таблицы заканчивается, когда не остается неиспользованных слов.

Поскольку линейный код  $\xi$  является подгруппой по сложению, то описанная процедура порождает смежные классы по

этой подгруппе. Слова из первого столбца стандартного расположения называются лидерами смежных классов.

Таблицу можно упростить, если помнить только первые строку и столбец, а остальные столбцы находить по мере необходимости.

*Определение 1.19.* Для линейного кода с проверочной матрицей  $H$  синдромом принятого вектора  $v$  называется вектор  $s = vH^T$ .

*Утверждение 1.7.* Все векторы из одного смежного класса имеют одинаковый синдром, присущий только этому смежному классу.

*Пример 1.6.* Построить стандартное расположение для  $(5,2)$ -кода над полем  $GF(2)$ , если он задается порождающей матрицей

$$G = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

*Решение.*

Напомним:

- строки матрицы  $G$  являются кодовыми словами;
- сумма двух кодовых слов – кодовое слово.

Кодовые слова:

$$c_0 = (0,0,0,0,0), c_2 = (1,0,1,1,1), c_3 = (0,1,1,0,1), c_4 = (1,1,0,1,0).$$

Построим стандартное расположение

00000	10111	01101	11010
00001	10110	01100	11011
00010	10101	01111	11000
00100	10011	01001	11110
01000	11111	00101	10010
10000	00111	11101	01010
<hr/>			
00011	10100	01110	11001
00110	10001	01011	11100

Этот код исправляет одну ошибку. Если принятое слово лежит ниже горизонтальной черты, то оно не подлежит декодированию.  $\square$

## 1.5.2. Описание алгоритма декодирования по синдрому

➤ Входные данные алгоритма: проверочная матрица  $H$ , лидер для каждого значения синдрома, принятый вектор  $v$ .

➤ Работа алгоритма:

- Для принятого вектора  $v$  вычисляем синдром  $s = vH^T$ ;

- По синдрому  $s$  находим лидер  $l_s$  соответствующего класса смежности;
- Полагаем кодовый вектор  $c = v - l_s$ .

*Пример 1.7.* С помощью алгоритма декодирования по синдрому найти кодовое слово для примера 1.6.

Пусть заданы:  $H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$  и  $v = 10010$

Таблица соответствия лидеров смежных классов и синдромов.

Лидеры смежных классов	Синдромы	Лидеры смежных классов	Синдромы
00000	000	01000	101
00001	001	10000	111
00010	010	00011	011
00100	100	00110	110

*Решение.*  $s = vH^T = 101$ . Лидер смежного класса равен  $l_s = 01000$ . Следовательно, переданное кодовое слово равно  $c = v - l_s = 10010 - 01000 = 11010$ .  $\square$

### 1.5.3. Коды Хэмминга

Линейный код  $(n, k)$  с данным числом проверочных символов  $m \geq 2$ , исправляющий любую единичную ошибку, называется кодом Хэмминга.

Рассмотрим построение кодов Хэмминга над полем  $GF(q)$ . Пусть  $m \geq 2$  – параметр кода Хэмминга. Положим  $n = (q^m - 1)/(q - 1)$ ,  $k = ((q^m - 1)/(q - 1)) - m$ , а в качестве проверочной матрицы  $H$  возьмем матрицу, любая пара столбцов которой линейно независима. Чтобы обеспечить линейную независимость, выберем в качестве столбцов матрицы  $H$  все  $m$ -последовательности, у которых первая ненулевая компонента равна единице. Отметим, что код Хэмминга строится так, чтобы  $d^* = 3$ .

*Пример 1.8.* Построим код Хэмминга над полем  $GF(3)$  с параметром  $m = 2$ .

*Решение.*

$$n = \frac{q^m - 1}{q - 1} = \frac{3^2 - 1}{3 - 1} = 4, \quad k = \frac{q^m - 1}{q - 1} - m = \frac{3^2 - 1}{3 - 1} - 2 = 2.$$

Строим (4,2)-код. Легко заметить, что матрица  $H = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 \end{bmatrix}$ . Так как  $H = [-P^T, E_{n-k}]$ , а  $G = [E_k, P]$ , то получаем  $G = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \end{bmatrix}$ .  $\square$

Отметим, что над полем  $GF(2)$  код Хэмминга  $(2^m-1, 2^m-1-m)$  получается, если в качестве проверочной матрицы взять матрицу размера  $(m \times 2^m - 1)$ , у которой все столбцы ненулевые и различны.

Код Хэмминга, исправляющий одиночные ошибки, существует для каждого  $q$ , для которого существует поле  $GF(q)$ , и для любого  $m$ .

## 1.6. Циклические коды

*Определение 1.20.* Линейный код  $\xi$  называется циклическим, если из того, что слово  $c = (c_0, \dots, c_{n-1})$  принадлежит  $\xi$ , следует, что слово  $c' = (c_{n-1}, c_0, \dots, c_{n-2})$  также принадлежит  $\xi$ . Кодовое слово  $c'$  получается из кодового слова  $c$  циклическим сдвигом всех компонент на одну позицию вправо. Каждый линейный код над  $GF(q)$  длины  $n$  представляет собой подпространство пространства  $GF^n(q)$ , а циклический код является частным случаем подпространства, так как обладает дополнительным свойством цикличности.

Интерпретируем векторное пространство  $GF^n(q)$  как множество многочленов степени меньше  $n$  над  $GF(q)$ . Это множество многочленов обладает структурой кольца  $GF(q)[x]/(x^n - 1)$  с операцией умножения  $f_1(x) \times f_2(x) = f_1(x)f_2(x) \bmod (x^n - 1)$ .

Циклический сдвиг можно записать через умножение в этом кольце  $x \times f_2(x) = xf_2(x) \bmod (x^n - 1)$ .

Тогда определение циклического кода будет следующим, если  $c(x) \in \xi$  то  $xc(x) \bmod (x^n - 1) \in \xi$ .

*Определение 1.21.* Приведенный ненулевой многочлен наименьшей степени в коде  $\xi$  называется порождающим многочленом кода  $\xi$  и обозначается через  $g(x)$ .

*Утверждение 1.8.* Циклический код состоит из всех произведений порождающего многочлена  $g(x)$  на многочлены степени не выше  $k-1$ , иначе говоря,  $\xi = \{c(x) = a(x)g(x), \forall a(x): \deg a(x) < n - \deg g(x)\}$ .

Кодирование можно осуществить по следующему простому правилу  $c(x) = i(x)g(x)$ . Такой кодер является несистематическим, так как по многочлену  $c(x)$  нельзя сразу установить  $i(x)$ .

*Утверждение 1.9.* Циклический код длины  $n$  с порождающим многочленом  $g(x)$  существует тогда и только тогда, когда  $x^n - 1 = g(x)h(x)$ . Многочлен  $h(x)$  называется проверочным многочленом, потому что для каждого кодового слова  $c(x) \in \xi$  выполняется равенство  $c(x)h(x) = i(x)g(x)h(x) = 0 \bmod (x^n - 1)$ .

Отметим, что возможно матричное описание циклических кодов. Одним из способов сделать это является построение порождающей матрицы непосредственно по порождающему многочлену. Так как кодовые слова записываются в виде  $c(x) = i(x)g(x)$ , то в матричной форме имеем

$$G = \begin{pmatrix} 0 & \dots & 0 & g_{n-k} & g_{n-k-1} & \dots & g_2 & g_1 & g_0 \\ 0 & & g_{n-k} & g_{n-k-1} & g_{n-k-2} & \dots & g_1 & g_0 & 0 \\ 0 & & g_{n-k-1} & g_{n-k-2} & g_{n-k-3} & \dots & g_0 & 0 & 0 \\ \dots & & & \dots & & & \dots & & \dots \\ g_{n-k} & \dots & & & & & \dots & & 0 \end{pmatrix}.$$

Проверочная матрица равна

$$H = \begin{pmatrix} 0 & 0 & 0 & \dots & \dots & h_{k-1} & h_k \\ \dots & & & \dots & \dots & & \dots \\ 0 & h_0 & h_1 & \dots & h_{k-1} & h_k & 0 & \dots & 0 \\ h_0 & h_1 & h_2 & \dots & h_k & 0 & 0 & \dots & 0 \end{pmatrix},$$

где  $h(x)$  – проверочный многочлен циклического кода.

## 1.7. БЧХ-коды

*Определение 1.22.* Код над полем  $GF(q)$ , имеющий длину  $n = q^m - 1$ , называется примитивным.

Коды Боуза-Чоудхури-Хоквингема (БЧХ) представляют собой подкласс циклических кодов и способны исправлять заданное число ошибок.

### 1.7.1 Построение примитивного БЧХ-кода

Код БЧХ над полем  $GF(q)$  длины  $n = q^m - 1$ , исправляющий  $t$  ошибок, строится следующим образом:

- выберем примитивный многочлен степени  $m$  и построим поле  $GF(q^m)$ . Пусть  $\alpha$  – примитивный элемент;
- найдем минимальные многочлены  $f_j(x)$  для  $\alpha^j$ ,  $j = 1, \dots, 2t$ ;
- порождающий многочлен БЧХ-кода находим  $g(x) = \text{НОК}(f_1(x), \dots, f_{2t}(x))$ ;
- число информационных символов  $k = n - \deg g(x)$ .

*Пример 1.9.* Построим все возможные коды БЧХ длины  $n = 15$  над полем  $GF(2)$ . Закодируем информационный многочлен  $i(x) = x^2 + 1$  кодом БЧХ, исправляющим 2 ошибки.

*Решение.*

1. Построим все возможные коды БЧХ длины  $n = 15$  над полем  $GF(2)$ . В качестве примитивного многочлена возьмем  $z^4 + z + 1$ . Построим поле  $GF(2^4)$ .

в виде степени	в виде многочлена	минимальные многочлены
0	0	
$\alpha^0$	1	$x + 1$
$\alpha$	$z$	$x^4 + x + 1$
$\alpha^2$	$z^2$	$x^4 + x + 1$
$\alpha^3$	$z^3$	$x^4 + x^3 + x^2 + x + 1$
$\alpha^4$	$z + 1$	$x^4 + x + 1$
$\alpha^5$	$z^2 + z$	$x^2 + x + 1$
$\alpha^6$	$z^3 + z^2$	$x^4 + x^3 + x^2 + x + 1$



$\alpha^7$	$z^3 + z + 1$	$x^4 + x^3 + 1$
$\alpha^8$	$z^2 + 1$	$x^4 + x + 1$
$\alpha^9$	$z^3 + z$	$x^4 + x^3 + x^2 + x + 1$
$\alpha^{10}$	$z^2 + z + 1$	$x^2 + x + 1$
$\alpha^{11}$	$z^3 + z^2 + z$	$x^4 + x^3 + 1$
$\alpha^{12}$	$z^3 + z^2 + z + 1$	$x^4 + x^3 + x^2 + x + 1$
$\alpha^{13}$	$z^3 + z^2 + 1$	$x^4 + x^3 + 1$
$\alpha^{14}$	$z^3 + 1$	$x^4 + x^3 + 1$

Легко заметить, что все возможные коды БЧХ можно описать так:

$t$ -кол-во ошибок	порождающий многочлен кода $g(x)$	полученный код
1	$x^4 + x + 1$	(15,11)
2	$x^8 + x^7 + x^6 + x^4 + 1$	(15,7)
3	$x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$	(15,5)
4	$x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 +$ $+ x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	(15,1)
5, 6, 7	$x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 +$ $+ x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	(15,1)
>7	код БЧХ не определен, поскольку ненулевых элементов поля $GF(2^4)$ всего 15.	

Отметим, что в данном примере для значений  $t > 3$  порождающий многочлен  $g(x)$  задает (15,1)-код, исправляющий 7 ошибок.

2. Закодируем информационный многочлен  $x^2 + 1$  кодом БЧХ, исправляющим 2 ошибки. Напомним, что  $c(x) = i(x)g(x)$ , следовательно

$$c(x) = (x^2 + 1)(x^8 + x^7 + x^6 + x^4 + 1) = x^{10} + x^9 + x^7 + x^4 + x^2 + 1). \square$$

*Пример 1.10.* Построим все возможные коды БЧХ длины  $n = 8$  над полем  $GF(3)$ .

*Решение.*

1. Построим поле  $GF(3^2)$ . В качестве примитивного многочлена возьмем  $z^2 + 2z + 2$ .

в виде степени	в виде многочлена	минимальные многочлены
0		
$\alpha^0$	1	$x + 2$
$\alpha$	$z$	$x^2 + 2x + 2$
$\alpha^2$	$z + 1$	$x^2 + 1$
$\alpha^3$	$2z + 1$	$x^2 + 2x + 2$
$\alpha^4$	2	$x + 1$
$\alpha^5$	$2z$	$x^2 + x + 2$
$\alpha^6$	$2z + 2$	$x^2 + 1$
$\alpha^7$	$z + 2$	$x^2 + x + 2$

Легко заметить, что все возможные коды БЧХ можно описать так:

$t$ - количество ошибок	порождающий многочлен кода $g(x)$	полученный код
1	$x^2 + x + 2$	(8,4)
2	$x^5 + 2x^3 + 2x^2 + x + 2$	(8,3)
3	$x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$	(8,1)

□

### 1.7.2. Построение общего БЧХ кода максимальной длины

*Определение 1.23.* Пусть заданы  $q, t$  и элемент  $\beta \in GF(q^m)$  порядка  $n = q^m - 1$ . Тогда для любого целого числа  $j_0$  код БЧХ, исправляющий  $t$  ошибок, задается порождающим многочленом

$$g(x) = \text{НОК}(f_{j_0}(x), \dots, f_{j_0+2t-1}(x)),$$

где  $f_j(x)$  - минимальный многочлен элемента  $\beta^j$ .

Выбор  $\beta, j_0$  позволяют уменьшить степень порождающего многочлена кода  $g(x)$ , а следовательно, и увеличить количество информационных символов.

### 1.7.3. Декодер БЧХ-кодов

Предположим, что параметр БЧХ  $j_0 = 1$  и что в основе конструкции кода БЧХ лежит элемент поля  $\alpha$ . Пусть  $v(x) = c(x) + e(x)$  является принятым многочленом, где  $c(x)$  – кодовое слово,  $e(x)$  – многочлен ошибок.

Поскольку мы строим БЧХ код для исправления  $t$  ошибок, то можно предположить, что при передаче произошло  $\gamma$  ошибок,  $0 \leq \gamma \leq t$ . Многочлен ошибок можно записать в виде  $e(x) = Y_1 x^{e_1} + \dots + Y_\gamma x^{e_\gamma}$ , где  $Y_i$  – величина ошибки, а  $l_i$  – положение  $i$  ошибки.

Для нахождения ошибки необходимо вычислить компоненты синдрома

$$S_j = v(\alpha^j) = c(\alpha^j) + e(\alpha^j), \text{ где } j = 1, \dots, 2t.$$

Действительно, так как  $c(x) = i(x)g(x)$ , а по построению кода БЧХ элементы  $\alpha, \dots, \alpha^{2t}$  являются корнями многочлена  $g(x)$ , то  $S_j = v(\alpha^j) = e(\alpha^j)$ .

$$\text{Легко заметить, что } S_1 = v(\alpha) = c(\alpha) + e(\alpha) = Y_1 \alpha^{e_1} + \dots + Y_\gamma \alpha^{e_\gamma}.$$

Положим локаторы ошибок  $X_i = \alpha^{l_i}$  для  $i = 1, \dots, \gamma$ ; тогда  $S_j = Y_1 X_1^j + \dots + Y_\gamma X_\gamma^j$  для  $j = 1, \dots, 2t$ .

Для нахождения многочлена ошибок надо найти неизвестные  $X_j, Y_j$  то есть решить систему

$$\begin{cases} S_1 = Y_1 X_1 + \dots + Y_\gamma X_\gamma \\ S_2 = Y_1 X_1^2 + \dots + Y_\gamma X_\gamma^2 \\ \dots \\ S_{2t} = Y_1 X_1^{2t} + \dots + Y_\gamma X_\gamma^{2t} \end{cases}$$

#### Алгоритм декодировки.

Алгоритм состоит из шести шагов.

Шаг 1. Находим компоненты синдрома  $S_j = v(\alpha^j)$ , где  $j = 1, \dots, 2t$ .

Шаг 2. Определим, сколько ошибок на самом деле произошло, то есть найдем  $\gamma$ . Напомним, что по построению кода БЧХ произошло не более  $t$  ошибок.

Поиск  $\gamma$  будем осуществлять следующим образом:

В качестве пробного значения возьмем  $\gamma = t$  и вычислим определитель матрицы  $M = \begin{bmatrix} S_1 & \dots & S_\gamma \\ S_\gamma & \dots & S_{2\gamma-1} \end{bmatrix}$ . Если определитель не равен нулю, то действительно произошло  $\gamma$  ошибок; в противном случае уменьшаем  $\gamma$  на единицу и повторяем процедуру.

Шаг 3. Рассмотрим вспомогательный многочлен  $\Lambda(x)$  (многочлен локаторов ошибок) – это многочлен, корнями которого являются обратные к локаторам ошибок величины  $X_l^{-1}$  для  $l = 1, \dots, \gamma$

$$\Lambda(x) = \Lambda_\gamma x^\gamma + \Lambda_{\gamma-1} x^{\gamma-1} + \dots + \Lambda_1 x + 1,$$

$$\Lambda(x) = (1 - xX_1)(1 - xX_2) \dots (1 - xX_\gamma).$$

Коэффициенты многочлена  $\Lambda(x)$  находятся при решении системы уравнений

$$M \begin{bmatrix} \Lambda_\gamma \\ \dots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} S_1 & S_2 & \dots & S_\gamma \\ S_2 & S_3 & \dots & S_{\gamma+1} \\ & & \dots & \\ S_\gamma & S_{\gamma+1} & \dots & S_{2\gamma-1} \end{bmatrix} \begin{bmatrix} \Lambda_\gamma \\ \dots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\gamma+1} \\ \dots \\ -S_{2\gamma} \end{bmatrix}$$

Система легко решается, так как определитель матрицы  $M$  не равен нулю.

Шаг 4. Находим корни многочлена  $\Lambda(x) = \Lambda_\gamma x^\gamma + \Lambda_{\gamma-1} x^{\gamma-1} + \dots + \Lambda_1 x + 1$ . Поскольку число элементов поля конечно, то корни многочлена локатора ошибок можно найти перебором всех элементов поля. Пусть  $X_l^{-1}$  для  $l = 1, \dots, \gamma$  - корни этого многочлена.

Шаг 5. Находим величины  $Y_1, \dots, Y_\gamma$ , решая систему линейных уравнений

$$\begin{cases} S_1 = Y_1 X_1 + \dots + Y_\gamma X_\gamma \\ S_2 = Y_1 X_1^2 + \dots + Y_\gamma X_\gamma^2 \\ \dots \\ S_\gamma = Y_1 X_1^\gamma + \dots + Y_\gamma X_\gamma^\gamma \end{cases}$$

В результате шага 5 был получен многочлен ошибок  $e(x)$ .

Шаг 6. Кодовый многочлен  $c(x)$  находится как  $c(x) = v(x) - e(x)$ .

Алгоритм окончен.

Легко заметить, что в случае, если параметр БЧХ  $j_0 \neq 1$ , алгоритм декодировки будет аналогичным. Изменится только Шаг 1, компоненты синдрома вычисляются по формуле  $S_j = v(\alpha^{j+j_0-1})$ , для  $j = 1, \dots, 2t$ .

*Пример 1.11.* Дано. Поле  $GF(16)$ , которое является расширением поля  $GF(2)$ . В качестве примитивного многочлена взят многочлен  $z^4 + z + 1$ . Пришло сообщение  $v(x) = x^{10} + x^9 + x^7 + x^4 + 1$ . Известно, что в основе конструкции кода БЧХ лежит элемент поля  $\alpha$  и  $t = 2$ .

Вопрос. Найдите кодовое слово  $c(x)$ , которое было отправлено.

*Решение.*

1. Найдём компоненты синдрома.

$$S_1 = v(\alpha) = \alpha^{10} + \alpha^9 + \alpha^7 + \alpha^4 + 1 =$$

$$(z^2 + z + 1) + (z^3 + z) + (z^3 + z + 1) + (z + 1) + 1 = z^2 = \alpha^2;$$

$$S_2 = v(\alpha^2) = \alpha^{20} + \alpha^{18} + \alpha^{14} + \alpha^8 + 1 = \alpha^5 + \alpha^3 + \alpha^{14} + \alpha^8 + 1 =$$

$$(z^2 + 1) + z^3 + (z^3 + 1) + (z^2 + 1) + 1 = z + 1 = \alpha^4;$$

$$S_3 = v(\alpha^3) = \alpha^{30} + \alpha^{27} + \alpha^{21} + \alpha^{12} + 1 = 1 + \alpha^{12} + \alpha^6 + \alpha^{12} + 1 =$$

$$1 + (z^3 + z^2 + z + 1) + (z^3 + z^2) + (z^3 + z^2 + z + 1) + 1 = z^3 + z^2 = \alpha^6;$$

$$S_4 = v(\alpha^4) = \alpha^{40} + \alpha^{36} + \alpha^{28} + \alpha^{16} + 1 = \alpha^{10} + \alpha^6 + \alpha^{13} + \alpha + 1 =$$

$$(z^2 + z + 1) + (z^3 + z^2) + (z^3 + z^2 + 1) + z + 1 = z^2 + 1 = \alpha^8.$$

2. Определим, сколько действительно произошло ошибок.

Предположим, что  $\gamma = t = 2$ .  $M = \begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} = \begin{bmatrix} \alpha & \alpha^2 \\ \alpha^2 & \alpha^4 \end{bmatrix}$  Легко

заметить, что определитель матрицы  $M$  равен нулю. Пусть  $\gamma = 1$ , тогда  $M = [S_1] = [\alpha^2]$  Определитель матрицы  $M$  не равен нулю, следовательно, произошла одна ошибка.

3. Найдем многочлен локаторов ошибок.

$\Lambda(x) = \Lambda_1 x + 1$ . Найдем коэффициенты многочлена локаторов ошибок.

$$[\Lambda_1] = M^{-1}[-S_2] = \alpha^{13}[-\alpha^4] = \alpha^{13}[\alpha^4] = \alpha^{17} = \alpha^2.$$

Следовательно, многочлен локаторов ошибок  $\Lambda(x) = \alpha^2 x + 1$ .

4. Найдем корни многочлена локаторов ошибок  $\Lambda(x) = \alpha^2 x + 1$ .

Легко убедиться, что корень многочлена  $x = \alpha^{13}$ . Следовательно ошибка произошла на второй позиции.

5. Поскольку код является двоичным, значения ошибок равны 1 и  $e(x) = x^2$ .

6. Найдем кодовое слово  $c(x)$ .

$$c(x) = v(x) - e(x) = x^{10} + x^9 + x^7 + x^4 + 1 - x^2 = x^{10} + x^9 + x^7 + x^4 + x^2 + 1. \square$$

*Пример 1.12.* Дано. Поле  $GF(9)$ , которое является расширением поля  $GF(3)$ . В качестве примитивного многочлена взят  $z^2 + 2z + 2$ . Пришло сообщение  $v(x) = x^6 + x^5 + x^3$  Известно, что в основе конструкции кода БЧХ лежит элемент поля  $\alpha$  и  $t = 2$ .

Вопрос. Найдите кодовое слово  $c(x)$ , которое было отправлено.

*Решение.*

1. Найдем компоненты синдрома.

$$S_1 = v(\alpha) = \alpha^6 + \alpha^5 + \alpha^3 = (2z + 2) + 2z + (2z + 1) = 0;$$

$$S_2 = v(\alpha^2) = \alpha^{12} + \alpha^{10} + \alpha^6 = \alpha^4 + \alpha^2 + \alpha^6 =$$

$$2 + (z + 1) + (2z + 2) = 2 = \alpha^4;$$

$$S_3 = v(\alpha^3) = \alpha^{18} + \alpha^{15} + \alpha^9 = \alpha^2 + \alpha^7 + \alpha =$$

$$(z + 1) + (z + 2) + z = 0;$$

$$S_4 = v(\alpha^4) = \alpha^{24} + \alpha^{20} + \alpha^{12} = \alpha^0 + \alpha^4 + \alpha^4 = \alpha^4.$$

2. Определим, сколько действительно произошло ошибок.

Предположим, что  $\gamma = t = 2$ .  $M = \begin{bmatrix} 0 & \alpha^4 \\ \alpha^4 & 0 \end{bmatrix}$ . Определитель

матрицы  $M$  не равен нулю, следовательно, произошло две ошибки.

3. Найдем многочлен локаторов ошибок.

$\Lambda(x) = \Lambda_2 x^2 + \Lambda_1 x + 1$ . Найдем коэффициенты многочлена локаторов ошибок.

$$\begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = M^{-1} \begin{bmatrix} -S_3 \\ -S_4 \end{bmatrix} = \begin{bmatrix} 0 & \alpha^4 \\ \alpha^4 & 0 \end{bmatrix} \begin{bmatrix} -0 \\ -\alpha^4 \end{bmatrix} = \begin{bmatrix} 0 & \alpha^4 \\ \alpha^4 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha^4 \\ 0 \end{bmatrix}.$$

Следовательно, многочлен локаторов ошибок  $\Lambda(x) = \alpha^4 x^2 + 1 = 2x^2 + 1$ .

4. Найдем корни многочлена локаторов ошибок  $\Lambda(x) = 2x^2 + 1$ .

Легко убедиться, что корни многочлена  $x = \alpha^8$  и  $x = \alpha^4$ . Следовательно ошибки произошли в нулевой и в четвертой позиции.

5. Найдем величины ошибок, решая систему линейных уравнений

$$\begin{cases} S_1 = Y_1 X_1 + Y_2 X_2 \\ S_2 = Y_1 X_1^2 + Y_2 X_2^2 \end{cases}$$

или

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} \alpha^0 & \alpha^4 \\ \alpha^0 & \alpha^0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \alpha^4 \end{bmatrix} = \begin{bmatrix} \alpha^4 & \alpha^4 \\ \alpha^0 & \alpha^4 \end{bmatrix} \begin{bmatrix} 0 \\ \alpha^4 \end{bmatrix} = \begin{bmatrix} \alpha^0 \\ \alpha^0 \end{bmatrix}.$$

Следовательно,  $e(x) = x^4 + 1$ .

Найдем кодовое слово  $c(x)$ .

$$c(x) = v(x) - e(x) = x^6 + x^5 + x^3 - x^4 - 1 = x^6 + x^5 + 2x^4 + x^3 + 2. \square$$

## Задачи

1. Построить поле Галуа  $GF(27)$ , которое является расширением поля  $GF(3)$  (порождающий многочлен поля  $x^3 + 2x + 1$ ).

2. Построить поле Галуа  $GF(32)$ , которое является расширением поля  $GF(2)$  (примитивный многочлен  $x^5 + x^2 + 1$ ).

3. Найти все минимальные многочлены для расширения  $GF(25)$  поля  $GF(5)$  (порождающий многочлен поля  $x^2 + x + 1$ ).

4. Построить код Хэмминга над полем  $GF(3)$  с параметром  $m = 3$ . Записать матрицы  $G$  и  $H$  в систематическом виде.

5. Построить код Хэмминга над полем  $GF(2)$  с параметром  $m = 5$ . Записать матрицы  $G$  и  $H$  в систематическом виде.

6. Построить стандартное расположение для  $(4,2)$ -кода над полем  $GF(3)$ , если он задается порождающей матрицей

$$G = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \end{bmatrix}.$$

7. Найти все примитивные элементы поля  $GF(32)$ , которое было получено из  $GF(2)$  с помощью порождающего многочлена поля  $x^5 + x + 1$ .

8. Дано. Поле  $GF(16)$ , которое является расширением поля  $GF(2)$ . В качестве примитивного многочлена взят многочлен  $z^4 + z + 1$ . Пришло сообщение  $v(x) = x^{10} + x^7 + x^4 + 1$ . Известно, что в основе конструкции кода БЧХ лежит элемент поля  $\alpha$  и  $t = 2$ . Найдите кодовое слово  $c(x)$ , которое было отправлено.

9. Дано. Поле  $GF(9)$ , которое является расширением поля  $GF(3)$ . В качестве примитивного многочлена взят  $z^2 + 2z + 2$ . Пришло сообщение  $v(x) = x^6 + 2x^5 + x^3 + 2$ . Известно, что в основе конструкции кода БЧХ лежит элемент поля  $\alpha$  и  $t = 2$ . Вопрос. Найдите кодовое слово  $c(x)$ , которое было отправлено.



## 2. Методы сжатия информации

Цель сжатия данных – обеспечить компактное представление данных, вырабатываемых источником, для их более экономного сохранения и передачи по каналам связи.

Все способы сжатия можно разделить на две категории: обратимое и необратимое сжатие.

Под необратимым сжатием, или сжатием с потерями, подразумевают такое преобразование входного потока данных, при котором выходной поток представляет, с некоторой точки зрения, достаточно похожий по внешним характеристикам на входной поток объект, однако отличается от него объемом. Такие подходы и алгоритмы используются для сжатия, например данных растровых графических файлов, где на выходе нужно представить графическую картинку, очень похожую на оригинал, но не обязательно являющуюся его точной копией.

Обратимое сжатие, или сжатие без потерь, всегда приводит к снижению объема выходного потока информации без изменения его информативности, т.е. без потери информационной структуры. Из выходного потока, полученного после выполнения алгоритма обратимого сжатия, при помощи декодирования, получается поток, в точности совпадающий с исходным.

Далее будут рассмотрены некоторые методы сжатия без потерь.

В основе этих методов лежит следующая идея: если представить часто используемые элементы короткими кодами, а редко используемые длинными кодами, то скорее всего для хранения данных потребуется меньший объем памяти, чем потребовался бы в случае представления всех элементов кодами одинаковой длины.

Методы сжатия можно разбить на два класса: статистические и преобразующие. Статистические методы оперируют величинами вероятностей элементов напрямую, а преобразующие используют статистические свойства данных опосредованно.

## 2.1. Статистические методы

*Утверждение 2.1.* Элемент  $s_i$ , вероятность появления которого равняется  $p(s_i)$ , выгоднее всего представлять  $-\log_2 p(s_i)$  битами.

Если распределение вероятностей  $F$  неизменно и вероятности появления элементов независимы, то среднюю длину кодов в битах можно найти как

$$H = -\sum p(s_i) \log_2 p(s_i),$$

это значение также называют энтропией источника в заданный момент времени.

Обычно вероятность появления элемента является условной. Тогда при кодировании очередного элемента  $s_i$  распределение вероятностей  $F$  принимает одно из возможных значений  $F_k$  и соответственно  $H = H_k$ . Среднюю длину кодов в битах можно вычислить по формуле

$$H = -\sum_k P_k H_k = -\sum_{k,i} P_k p_k(s_i) \log p_k(s_i),$$

где  $P_k$  – вероятность того, что  $F$  принимает  $k$ -е значение, которому соответствует набор вероятностей  $p_k(s_i)$  генерации всех возможных элементов  $s_i$ .

### 2.1.1. Алгоритм Хаффмана

Исходными данными алгоритма являются:

- алфавит  $A = \{a_1, \dots, a_n\}$ , состоящий из  $n$  символов;
- $p(a_i)$  – вероятность появления каждого символа алфавита в рассматриваемом сообщении.

**Алгоритм Хаффмана** состоит из нескольких шагов:

Шаг 1. Выстраиваем все символы текущего алфавита в порядке убывания вероятностей.

Шаг 2. Строим новый алфавит  $A_j$ , который получается из предыдущего заменой двух символов  $a_{i_{r-1}}$   $a_{i_r}$  (с наименьшими вероятностями) на псевдосимвол  $a'$ , причем  $p(a') = p(a_{i_{r-1}}) + p(a_{i_r})$ .

Продолжая эту процедуру (шаг 1 – шаг 2), будем приходить ко все более коротким алфавитам; после  $n-2$  – кратного сжатия придем к алфавиту  $A_{n-2}$ , содержащему уже всего две буквы.

Шаг 3. Символам последнего алфавита поставим в соответствие кодовые обозначения 0 и 1.

Шаг 4. Пусть кодовые обозначения уже приписаны всем символам алфавита  $A_j$ . Символам предыдущего алфавита  $A_{j-1}$  (где  $A_0$  – исходный алфавит  $A$ ) кодовые обозначения приписываются:

– если  $a \in A_j$  и  $a \in A_{j-1}$ , то в алфавите  $A_{j-1}$  он будет иметь те же кодовые обозначения, что и в алфавите  $A_j$ ;

– если элементы  $a_{i_{r-1}}$ ,  $a_{i_r}$  принадлежат алфавиту  $A_{j-1}$  и не принадлежат алфавиту  $A_j$  (они были заменены на элемент  $a' \in A_j$ ), то их кодовые обозначения получаются из кодового обозначения элемента  $a'$  добавлением цифр 0 и 1 в конце.

Выполняем шаг 4, пока всем элементам исходного алфавита  $A$  не будет сопоставлено кодовое обозначение. Алгоритм окончен.

Легко заметить, что кодирование некоторого алфавита по методу Хаффмана не является однозначно определенной процедурой. Например, на любом этапе построения кода можно заменить цифру 1 на цифру 0 и наоборот; при этом получатся два различных кода.

*Пример 2.1.* Дан алфавит  $A = \{ "a", "b", "c", "d", "e", "f" \}$ , для каждого символа из алфавита  $A$  задана вероятность его появления в тексте.

символ	$a$	$b$	$c$	$d$	$e$	$f$
вероятность	$\frac{4}{10}$	$\frac{2}{10}$	$\frac{2}{10}$	$\frac{1}{10}$	$\frac{5}{100}$	$\frac{5}{100}$

Закодировать алфавит с помощью алгоритма Хаффмана.

### Решение

Процесс перехода к более коротким алфавитам можно описать следующей таблицей.

	Вероятности				
	исходный алфавит $A$	сжатые алфавиты			
	$A$	$A_1$	$A_2$	$A_3$	$A_4$
$a$	0,4	0,4	0,4	0,4	0,6
$b$	0,2	0,2	0,2	0,4	0,4
$c$	0,2	0,2	0,2	0,2	
$d$	0,1	0,1	0,2		
$e$	0,05	0,1			
$f$	0,05				

Символам последнего алфавита  $A_4$  поставим в соответствие кодовые обозначения 0 и 1. Процесс присвоения символам исходного алфавита кодовых обозначений можно описать следующей таблицей.

вероятности и кодовые обозначения.							
сжатые алфавиты						исходный алфавит	
$A_4$	$A_3$	$A_2$	$A_1$	$A_0$			
0,6	0,4	0,4	0,4	0,4	0	0	$a$
0,4	0,4	0,2	0,2	0,2	1	10	$b$
	0,2	0,2	0,2	0,2		11	$c$
		0,2	0,1	0,1		110	$d$
			0,1	0,05		1101	$e$
				0,05		1100	$f$

Приведем другой пример кода Хаффмана для этого же алфавита.

вероятности и кодовые обозначения.					
сжатые алфавиты				исходный алфавит	
$A_4$	$A_3$	$A_2$	$A_1$	$A_0$	
0,6    1	0,4    0	0,4    11	0,4    11	0,4    11	$a$
0,4    0	0,4    11	0,2    10	0,2    01	0,2    01	$b$
	0,2    10	0,2    01	0,2    00	0,2    00	$c$
		0,2    00	0,1    101	0,1    100	$d$
			0,1    100	0,05    1011	$e$
				0,05    1010	$f$



Недостатки метода Хаффмана:

- самой большой сложностью работы с кодами Хаффмана, как следует из предыдущего обсуждения, является необходимость иметь таблицы вероятностей для каждого типа сжимаемых данных. В общем же случае, когда вероятность символов для входных данных неизвестна, статистические коды Хаффмана работают неэффективно. Решением этой проблемы является статистический анализ кодируемых данных, выполняемый в ходе первого прохода по данным, и составление на его основе кодового дерева. Собственно кодирование при этом выполняется вторым проходом;
- минимальная длина кодового слова для них не может быть меньше единицы, тогда как энтропия сообщения вполне может составлять и 0,1, и 0,01 бит/букву. В этом случае код Хаффмана становится существенно избыточным. Проблема решается применением алгоритма к блокам символов, но тогда усложняется процедура кодирования/декодирования;
- код Хаффмана обеспечивает среднюю длину кода, совпадающую с энтропией, только в том случае, когда вероятности символов источника являются целыми отрицательными степенями двойки:  $1/2 = 0,5$ ;  $1/4 = 0,25$ ;  $1/8 = 0,125$  и т.д. На практике же такая ситуация встречается очень редко.

## 2.1.2. Арифметическое кодирование

Исходными данными алгоритма являются:

- алфавит  $A = \{a_1, \dots, a_n\}$ , состоящий из  $n$  символов;
- строка  $B = b_1 b_2 b_3 \dots b_s$ , которую надо закодировать и элементы которой принадлежат алфавиту  $A$ .

Арифметическое кодирование основано на идее представления кодируемого текста в виде дроби. По мере кодирования исходного текста отображающий его интервал уменьшается, а количество десятичных разрядов, служащих для его представления, возрастает. Очередные символы входного текста сокращают величину интервала исходя из значений их вероятностей.

Рассмотрим процесс кодирования (построим дробь на интервале  $[0,1)$ ).

**Алгоритм кодирования** состоит из нескольких шагов

Шаг 1. Инициализация.

1) определим количество (вероятность) каждого из символов алфавита в сообщении и назначим каждому из них интервал, пропорциональный его вероятности;

2) введем вспомогательные переменные: нижняя\_граница, верхняя\_граница, интервал;

3) присвоим нижняя\_граница=0,0, верхняя\_граница=1,0.

Шаг 2. Работа алгоритма (шаг два выполняется до конца строки которую надо закодировать)

1) считываем очередной символ;

2) интервал = верхняя\_граница – нижняя\_граница;

3) верхняя\_граница = нижняя\_граница + интервал \* верхняя граница для очередного символа;

4) нижняя\_граница = нижняя\_граница + интервал \* нижняя граница для очередного символа.

Шаг 3. Выдать любое число принадлежащие интервалу [нижняя\_граница, верхняя\_граница).

Алгоритм окончен.

*Пример 2.2.* Дан алфавит

$A = \{ "Д", ".", " ", "X", "а", "ф", "м", "н" \}$  и строка  $B = "Д. Хаффман"$ .

Закодировать арифметическим методом строку  $B$ .

*Решение.*

Инициализация. Построим интервалы вероятностей для символов сообщения.

символ	вероятность	интервал вероятностей
пробел	1/10	[0,0 0,1)
.	1/10	[0,1 0,2)
а	2/10	[0,2 0,4)
Д	1/10	[0,4 0,5)
м	1/10	[0,5 0,6)
н	1/10	[0,6 0,7)
ф	2/10	[0,7 0,9)
Х	1/10	[0,9 1,0)

Располагать символы в таблице можно в любом порядке: по мере их появления в тексте, в алфавитном или по возрастанию вероятностей – это совершенно не принципиально. Результат кодирования при этом будет разным, но эффект – одинаковым.

Процесс кодирования можно описать следующей таблицей.

очередной символ	верхняя граница интервала а для очередного символа	нижняя граница интервала а для очередного символа	интервал	верхняя граница	нижняя граница
				1,0	0,0
Д	0,5	0,4	1	0,5	0,4
.	0,2	0,1	0,1	0,42	0,41
пробел	0,1	0,0	0,01	0,411	0,410
Х	1	0,9	0,001	0,411	0,4109
а	0,4	0,2	0,0001	0,41094	0,41092
ф	0,9	0,7	0,00002	0,410938	0,410934
ф	0,9	0,7	0,000004	0,4109376	0,4109368
м	0,6	0,5	0,0000008	0,41093728	0,4109372
а	0,4	0,2	0,00000008	0,410937232	0,410937216
н	0,7	0,6	0,000000016	0,4109372272	0,4109372256

Следовательно, строку «Д. Хаффман» можно представить числом 0,410937226.□

Нетрудно убедиться в том, что чем шире конечный интервал, тем меньшим числом десятичных (и, следовательно, двоичных) разрядов он может быть представлен. Ширина же интервала зависит от распределения вероятностей кодируемых символов – более вероятные символы сужают интервал в меньшей степени. Покажем это на простом примере

*Пример 2.3.* Дан алфавит  $A = \{ "A", "B" \}$  и строка  $B = "AAAAAB"$ . Закодировать арифметическим методом строку  $B$ .

*Решение.*

Инициализация. Построим интервалы вероятностей для символов сообщения.

символ	вероятность	интервал вероятностей
$A$	8/10	[0,0 0,8)
$B$	2/10	[0,8 1,0)

Процесс кодирования можно описать следующей таблицей.

очередной символ	верхняя граница интервала для очередного символа	нижняя граница интервала для очередного символа	интервал	верхняя граница	нижняя граница
				1,0	0,0
$A$	0,8	0,0	1	0,8	0,0
$A$	0,8	0,0	0,8	0,64	0,0
$A$	0,8	0,0	0,64	0,512	0,0
$A$	0,8	0,0	0,512	0,4096	0,0
$B$	1,0	0,8	0,4096	0,4096	0,32768

Следовательно, строку «AAAAAB» можно представить числом 0,3.□



**Алгоритм арифметического декодирования** может быть описан следующим образом:

Исходные данные алгоритма: число, алфавит  $A = \{a_1, \dots, a_n\}$ , для каждого элемента которого определен интервал вероятностей;

Шаг 1. Инициализация.

введем вспомогательные переменные: интервал, символ;

Шаг 2. Работа алгоритма

Пока символ не равен символу конца блока, выполняется цикл

1) символ = найти символ, в интервал которого попадает число;

2) выдать символ;

3) интервал = верхняя граница интервала(символа) – нижняя граница интервала(символа);

4) число = число – нижняя граница интервала(символа);

5) число = число / интервал.

цикл окончен. Алгоритм окончен.

*Пример 2.4.* Дано число 0,410937226, число символов в закодированной строке  $|B| = 10$  и алфавит  $A = \{ "Д", ".", " ", "Х", "а", "ф", "м", "н" \}$ , каждому элементу алфавита соответствует некоторый интервал вероятностей.

символ	пробел	.	а	Д	м	н	ф	Х
интервал вероятностей	[0 0,1)	[0,1 0,2)	[0,2 0,4)	[0,4 0,5)	[0,5 0,6)	[0,6 0,7)	[0,7 0,9)	[0,9 1,0)

Найти закодированную строку  $B$ .

### Решение

Процесс декодирования можно описать следующей таблицей.

итерация	число	верхняя граница интервала (символа), куда попадает число	нижняя граница интервала (символа), куда попадает число	интервал	символ
	0,410937226				
1	0,410937226	0,5	0,4	0,1	Д
2	0,10937226	0,2	0,1	0,1	.
3	0,0937226	0,1	0,0	0,1	пробел
4	0,937226	1	0,9	0,1	Х
5	0,37226	0,4	0,2	0,2	а
6	0,8613	0,9	0,7	0,2	ф
7	0,8065	0,9	0,7	0,2	ф
8	0,5325	0,6	0,5	0,1	м
9	0,325	0,4	0,2	0,2	а
10	0,625	0,7	0,6	0,1	н

В результате работы алгоритма получили строку «Д. Хаффман».□

Это основная идея арифметического кодирования, его практическая реализация несколько сложнее.

Некоторые проблемы, возникающие при арифметическом кодировании:

- одна проблема состоит в том, что окончательный результат кодирования – конечный интервал – станет известен только по окончании процесса кодирования;
- другая проблема – это вопрос точности представления;
- третья состоит в том, что декодеру нужно обязательно каким-либо образом дать знать об окончании процедуры декодирования.

## 2.2. Преобразующие методы

### 2.2.1. Словарные методы

Идея словарных методов:

- рассматриваем входную последовательность как последовательность строк, содержащих произвольное количество символов;
- при кодировании заменяем строки символов на коды (индексы строк в некотором словаре);
- при декодировании осуществляется замена индексов на соответствующие им фразы словаря.

Словарь – это набор строк, которые предположительно будут встречаться в обрабатываемой последовательности. Индексы строк должны быть построены таким образом, чтобы в среднем их представление занимало меньше места, чем требуют замещаемые строки. За счет этого и происходит сжатие.

#### 2.2.1.1. Алгоритм LZ77

Алгоритм был основан в 1977 году и является родоначальником класса алгоритмов со скользящим окном. В качестве словаря в нем используется блок уже закодированной последовательности. Обычно при работе алгоритма положение этого блока относительно начала последовательности меняется, словарь как бы «скользит» по входному потоку данных.

Скользящее окно состоит из двух частей:

- последовательности уже закодированных символов (словарем) длины  $W$ ,
- буфера предварительного просмотра длины  $n$ .

#### Опишем работу алгоритма кодирования.

Пусть мы уже закодировали  $t$  символов  $S_1, \dots, S_t$ . Тогда словарем будет строка  $S_{t-(W-1)}, \dots, S_t$ , а в буфере предварительного просмотра будет находиться строка  $S_{t+1}, \dots, S_{t+n}$ . Идея алгоритма заключается в поиске самого длинного совпадения между подстрокой буфера и фразами словаря. Эти фразы могут начинаться с любого символа  $S_{t-(W-1)}, \dots, S_t$  и выходить за пределы словаря, но должны лежать в окне. Полученная в результате строка

$S_{t-(i-1)}, \dots, S_{t-(i-1)+(j-1)}$  кодируется с помощью триады  $[i, j, s]$ , где  $i$  есть смещение от начала буфера,  $j$  – длина соответствия,  $s$  – первый символ, непосредственно следующий за совпадающей строкой буфера. Затем скользящее окно сдвигается на  $j + 1$  символ вправо и осуществляет переход к новому циклу кодирования. Алгоритм окончен.

Легко заметить, что декодирование сжатых данных осуществляется путем замены кода на блок символов (фразы словаря и явно передаваемого символа). Декодер должен выполнять те же действия по изменению окна, что и кодер.

Объем памяти, требуемый кодеру и декодеру, ограничивается размером окна. Количество бит, необходимое для представления смещения  $i$  в триаде, равняется округленному в большую сторону  $\log_2 W$ . Количество бит, которое требуется для кодирования длины соответствия  $j$ , может быть вычислено как округленный в большую сторону  $\log_2 n$ .

*Пример 2.5.* Рассмотрим работу алгоритма LZ77 с помощью кодирования фразы «красная \_краска». Пусть длина буфера, будет равна 5 символам, а размер словаря больше длины сжимаемой строки.

*Решение.*

Положим также, что

- символ  $s_t$  соответствует единичному смещению относительно символа  $s_{t+1}$ , с которого начинается буфер;
- нулевое смещение зарезервируем для обозначения конца кодирования;
- если имеется несколько фраз с одинаковой длиной совпадения, то выбираем ближайшую к буферу.

Процесс кодирования можно описать таблицей.

Шаг	Скользящее окно		Совпадающая фраза	Зашифрованные данные		
	Словарь	Буфер		$i$	$j$	$s$
1	—	красн	-	1	0	«к»
2	к	расна	-	1	0	«р»
3	кр	асная	-	1	0	«а»
4	кра	сная_	-	1	0	«с»
5	крас	ная_к	-	1	0	«н»
6	красн	ая_кр	а	3	1	«я»
7	красная	крас	-	1	0	«_»
8	красная_	краск	крас	8	4	«к»
9	красная_краск	а	-	0	0	«а»

Длину полученного кода можно вычислить следующим образом: для кодирования  $i$  достаточно 4 бит, для  $j$  нужно 3 бита, символы  $s$  требуют 1 байт для своего представления. Следовательно, длина полученного кода равна  $9 \cdot (4 + 3 + 8) = 135$  бит, против  $14 \cdot 8 = 112$  бит исходной строки.

Рассмотрим процесс декодирования

Зашифрованные данные			Печать	Словарь
$i$	$j$	$s$		
1	0	«к»	к	к
1	0	«р»	р	кр
1	0	«а»	а	кра
1	0	«с»	с	крас
1	0	«н»	н	красн
3	1	«я»	ая	красная
1	0	«_»	_	красная_
8	4	«к»	краск	красная_краск
0	0	«а»	а	красная_краска



Недостатки LZ77:

- с ростом размера словаря скорость работы алгоритма кодирования пропорционально замедляется;
- кодирование одиночных символов очень неэффективно.

### **2.2.1.2 Алгоритм LZSS**

Этот алгоритм был предложен в 1982 году и является модификацией алгоритма LZ77. Главное преимущество по сравнению с LZ77 – это то, что символы записываются в явном виде, если соответствующий им код имеет большую длину.

Идея алгоритма аналогична идее алгоритма LZ77. Единственное отличие заключается в добавлении к каждому указателю и к каждому незакодированному символу 1-битового флага  $f$ , позволяющего различить эти объекты. Флаг указывает тип и длину следующих за ним данных.

Если  $f = 0$ , код состоит из пары  $[f, (i, j)]$ , где  $i$  есть смещение от начала буфера,  $j$  – длина соответствия. Если  $f = 1$ , код состоит из пары  $[f, s]$ , где  $s$  есть передаваемый в явном виде символ. В LZSS окно сдвигается ровно на длину найденной подстроки или на 1, если не найдено вхождение подстроки из буфера в словаре.

Легко заметить, что декодирование сжатых данных осуществляется в зависимости от значения флага:

- или путем замены кода на блок символов (фразы словаря);
- или путем добавления явно передаваемого символа.

Декодер должен выполнять те же действия по изменению окна, что и кодер.

*Пример 2.6.* Рассмотрим работу алгоритма LZSS с помощью кодирования фразы «перепел». Пусть длина буфера равна 5 символам, а размер словаря больше длины сжимаемой строки.

*Решение.*

Процесс кодирования можно описать таблицей.

Шаг	Скользящее окно		Совпадающая фраза	Зашифрованные данные			
	Словарь	Буфер		$i$	$i$	$j$	$s$
1	—	переп	-	0	-	-	«п»
2	п	ерепе	-	0	-	-	«е»
3	пе	репел	-	0	-	-	«р»
4	пер	епел	е	1	2	1	-
5	пере	пел	пе	1	4	2	-
6	перепе	л	-	0	-	-	«л»

Для кодирования строки потребовалось 6 итераций (4 раза символы были переданы в явном виде и 2 раза были переданы указатели). Заметим, что для кодирования  $i$  достаточно 3 бита, для  $j$  нужно 3 бита, для  $j$  нужен 1 бит, символы  $s$  требуют 1 байт для своего представления. Следовательно, длина полученного кода равна  $2 \cdot (3+3+1) + 4 \cdot 8 = 46$  бит, против  $7 \cdot 8 = 56$  бит исходной строки.  $\square$

Недостатки LZSS и LZ77:

- строку  $S_i$  нельзя закодировать строкой  $S_j$ , если расстояние между ними больше длины словаря;
- длина строки, которую можно закодировать, ограничена размером буфера.

### 2.2.1.3. Алгоритм LZ78

Алгоритм опубликован в 1978 году и является родоначальником класса алгоритмов. LZ78 не использует скользящее окно, он хранит словарь из уже просмотренных фраз. При старте алгоритма этот словарь содержит только одну пустую строку. На каждом шаге алгоритма в словарь добавляется новая фраза, которая представляет собой соединение одной из фраз словаря  $S$ , имеющей самое длинное совпадение со строкой буфера, и символа  $s$ . Символ  $s$  является символом, следующим за строкой буфера, для которой найдена совпадающая фраза  $S$  из словаря.

Кодер порождает только последовательность кодов фраз. Каждый код состоит из номера  $n$  «родительской фразы» в словаре и символа  $s$ .

Легко заметить, что декодирование сжатых данных осуществляется путем замены кода на блок символов (фразы словаря и явно передаваемого символа). Декодер должен выполнять те же действия по построению словаря, что и кодер.

*Пример 2.7.* Рассмотрим работу алгоритма LZ78 с помощью кодирования фразы «перепел». Размер словаря не ограничен.

*Решение.*

Положим, что

- фраза с номером 0 зарезервирована для обозначения конца сжатой строки;
- фраза с номером 1 зарезервирована за пустой строкой.

шаг	добавляемая в словарь фраза		буфер	совпадающая фраза $S$	закодированные данные	
	сама фраза	ее номер			$n$	$s$
1	п	2	перепел	-	1	«п»
2	е	3	ерепел	-	1	«е»
3	р	4	репел	-	1	«р»
4	еп	5	епел	е	3	«п»
5	ел	6	ел	е	3	«л»

Фразу удалось закодировать за 5 шагов. Для представления  $n$  посредством кодов фиксированной длины потребуется 3 бита. Размер сжатой строки равен  $5 \cdot (3+8) = 55$  битам.

Рассмотрим процесс декодирования

шаг	закодированные данные		печать	добавляемая в словарь фраза	
	$n$	$s$		сама фраза	ее номер
1	1	«п»	п	п	2
2	1	«е»	е	е	3
3	1	«р»	р	р	4
4	3	«п»	еп	еп	5
5	3	«л»	ел	ел	6

□



### 2.2.1.4. Алгоритм LZW

Этот алгоритм был предложен в 1984 году и является модификацией алгоритма LZ78.

Алгоритм кодирования состоит из нескольких шагов

Шаг 1. Инициализация словаря всеми возможными односимвольными фразами. Инициализация входной фразы  $w$  первым символом сообщения.

Шаг 2. Считать очередной символ  $K$  из кодируемого сообщения.

Шаг 3. Если конец\_сообщения

    Выдать код для  $w$

    Конец

    Если фраза  $wK$  уже есть в словаре

        Присвоить входной фразе значение  $wK$

        Перейти к Шагу 2

    Иначе

        Выдать код  $w$

        Добавить  $wK$  в словарь

        Присвоить входной фразе значение  $K$

        Перейти к Шагу 2.

Алгоритм окончен.

*Пример 2,8.* Рассмотрим работу алгоритма LZW с помощью кодирования фразы «красная краска». Размер словаря не ограничен.

*Решение.*

Процесс кодирования можно описать таблицей.

входная фраза, $wK$ (в словарь)	код для $w$	позиция словаря
ASCII		0-255
“кр”	код ‘к’	256
“ра”	код ‘р’	257
“ас”	код ‘а’	258
“сн”	код ‘с’	259
“на”	код ‘н’	260
“ая”	код ‘а’	261

“я ”	код ’я’	262
“ к”	код ’ ’	263
“кра”	<256>	264
“аск”	<258>	265
“ка”	код ’к’	266
“а”	код ’а’	

В этом примере длина полученного кода равна  $12 \cdot 9 = 108$  битам. □

При распаковке нужно придерживаться следующего правила. Словарь пополняется после считывания первого символа, идущего за текущим кодом.

*Пример 2.9.* Рассмотрим процесс декодирования алгоритма LZW.

*Решение.*

Процесс декодирования можно описать таблицей.

входной код	печать	словарь	позиция словаря
		ASCII	0-255
код ’к’	«к»	“кр”	256
код ’р’	«р»	“ра”	257
код ’а’	«а»	“ас”	258
код ’с’	«с»	“сн”	259
код ’н’	«н»	“на”	260
код ’а’	«а»	“ая”	261
код ’я’	«я»	“я ”	262
код ’ ’	« »	“ к”	263
<256>	«кр»	“кра”	264
<258>	«ас»	“аск”	265
код ’к’	«к»	“ка”	266
код ’а’	«а»		

□

Улучшать сжатие алгоритмов LZ можно двумя путями:

- уменьшением количества указателей при неизменной или большей общей длине закодированных фраз за счет более

эффективного разбиения входной последовательности на фразы словаря;

- увеличением эффективности кодирования индексов фраз словаря и литеров, то есть уменьшением количества битов, в среднем требуемых для кодирования индекса или литеры.

### 2.2.2. Алгоритм RLE

Данный алгоритм часто используется при сжатии изображений.

Работа алгоритма RLE (кодирование длин повторений) основана на следующей идее:

- изображение вытягивается в цепочку байтов по строкам раstra;
- сжатие происходит за счет того, что в исходном изображении встречаются цепочки одинаковых байтов. Замена таких цепочек на пары (счетчик повторений, значение) уменьшает избыточность данных.

В формате РСХ это реализовано следующим образом: в выходной поток пишется либо непосредственно значение пикселя (если в двух его старших разрядах не единицы):

*[ XX значение ]*

либо пара

*[ 11 счетчик ] [ что повторять ]*

причем повторяемый символ повторяется число раз, на единицу большую счетчика.

Отметим, что возможны ситуации, когда размер сжатого файла больше размера исходного изображения.

Алгоритм кодирования длин повторений может быть очень эффективен при сжатии двоичных данных, например, черно-белых фиксированных изображений.

В этом случае работа алгоритма основана на следующей идее:

- изображение вытягивается в цепочку бит по строкам раstra;
- вместо кодирования собственно данных кодированию подвергаются числа, соответствующие длинам участков, на которых данные сохраняют неизменное значение.

*Пример 2.10.* Дан двоичный вектор данных  
 $X = (0111000011110000000100000001000000010000000111100011110111101111)$   
 длиной 64 бита.

Закодировать методом длин повторений

*Решение.*

Выделим в векторе  $X$  участки, на которых данные сохраняют неизменное значение, и определим их длины. Результирующая последовательность длин участков – положительных целых чисел, соответствующих исходному вектору данных  $X$ , – будет иметь вид  $r = (1, 3, 4, 4, 7, 1, 7, 1, 7, 1, 7, 4, 3, 4, 1, 4)$ . Теперь эту последовательность можно закодировать каким-либо статистическим кодом, например, кодом Хаффмана. В результате получим

длина участка	4	1	7	3
код	0	10	110	111

Для того, чтобы указать, что кодируемая последовательность начинается с нуля, добавим в начале кодового слова символ 0. В результате получим кодовое слово  $h = (0101110011010110101101011001110100100)$  длиной в 37 бит, то есть информация была сжата почти наполовину.  $\square$

## Задачи

1. Сжать методом Хаффмана алфавит из 6 символов с вероятностями  $\left\{ \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \frac{5}{100}, \frac{5}{100}, \frac{3}{10} \right\}$ . Вычислить среднее количество бит на единицу сжатого сообщения.

2. Сжать методом Хаффмана алфавит из 9 символов с вероятностями  $\left\{ \frac{1}{10}, \frac{2}{10}, \frac{3}{10}, \frac{5}{100}, \frac{5}{100}, \frac{9}{100}, \frac{1}{100}, \frac{2}{10} \right\}$ . Вычислить среднее количество бит на единицу сжатого сообщения.

3. Дан алфавит  $A = \{ "C", "e", "к", "р", "т" \}$  и строка  $B = \text{"Секрет"}$ . Сжать с помощью арифметического кодирования строку  $B$ .

4. Дан алфавит  $A = \{ "к", "о", "с" \}$  и строка  $B = \text{"кокос"}$ . Сжать с помощью арифметического кодирования строку  $B$ .

5. Закодировать сообщения «АААВВС», «кибернетики» и «кокос», вычислить длины в битах полученных кодов, используя алгоритмы:

- LZ77 (словарь – 12 байт, буфер – 4 байта);
- LZSS (словарь – 12 байт, буфер – 4 байта);
- LZ78 (размер словаря не ограничен);
- LZW (размер словаря не ограничен).

## Литература

1. Акритас, А. Основы компьютерной алгебры с приложениями / А. Акритас. – М. : Мир, 1994.
2. Берлекэмп, Э. Алгебраическая теория кодирования / Э. Берлекэмп. – М. : Мир, 1971.
3. Блейхут, Р. Теория и практика кодов, контролирующих ошибки / Р. Блейхут. – М.: Мир, 1984.
4. Тимофеев, Е.А. Защита информации в распределенных сетях : учебное пособие / Е.А. Тимофеев; Яросл. гос. ун-т. – Ярославль : ЯрГУ, 2001.
5. Казарин, Л.С. Теория кодирования : учебное пособие / Л.С. Казарин; Яросл. гос. ун-т. –Ярославль : ЯрГУ, 1987.
6. Ватолин, Д. Методы сжатия данных / Д. Ватолин, А. Ратушняк. – М. : Диалог-Мифи, 2002.
7. Лидовский, В.В. Теория информации : учебное пособие / В.В. Лидовский. – М.: Компания Спутник+, 2004.
8. Шульгин, В.И. Основы теории передачи информации. Ч. I. Экономное кодирование : учебное пособие / В.И. Шульгин. – Харьков, 2003.

## Оглавление

<b>1. Коды, исправляющие ошибки.....</b>	<b>3</b>
1.1. Основные определения .....	3
1.2. Некоторые сведения из алгебры .....	4
1.3. Поля Галуа.....	6
1.4. Минимальные многочлены .....	9
1.5. Линейные блочные коды.....	10
1.5.1. Стандартное расположение .....	11
1.5.2. Описание алгоритма декодирования по синдрому.....	12
1.5.3. Коды Хэмминга .....	13
1.6. Циклические коды.....	14
1.7. БЧХ-коды.....	16
1.7.1 Построение примитивного БЧХ-кода.....	16
1.7.2. Построение общего БЧХ кода максимальной длины.....	18
1.7.3. Декодер БЧХ-кодов.....	19
Задачи .....	24
<b>2. Методы сжатия информации.....</b>	<b>25</b>
2.1. Статистические методы .....	26
2.1.1. Алгоритм Хаффмана .....	26
2.1.2. Арифметическое кодирование .....	30
2.2. Преобразующие методы.....	35
2.2.1. Словарные методы .....	35
2.2.1.1. Алгоритм LZ77.....	35
2.2.1.2 Алгоритм LZSS .....	38
2.2.1.3. Алгоритм LZ78.....	39
2.2.1.4. Алгоритм LZW .....	41
2.2.2. Алгоритм RLE.....	43
Задачи .....	45
<b>Литература .....</b>	<b>46</b>

Учебное издание

# Теория кодирования

*Методические указания*

Составитель **Краснов** Михаил Владимирович

Редактор, корректор А.А. Аладьева  
Компьютерная верстка Е.Л. Шелеховой

Подписано в печать 22.06.2006 г. Формат 60х84/16.

Бумага тип. Усл. печ. л. 2,79. Уч.-изд. л. 1,54.

Тираж 100 экз. Заказ

Оригинал-макет подготовлен  
в редакционно-издательском отделе ЯрГУ.

Отпечатано на ризографе.

Ярославский государственный университет.  
150000 Ярославль, ул. Советская, 14.





# ***Т***еория ***К***одирования

