МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

М.А. КУДРИНА, К.А. КУДРИН

ТЕОРИЯ ИНФОРМАЦИИ

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве практикума для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника

САМАРА
Издательство Самарского университета
2023

ISBN 978-5-7883-1968-1

УДК 519.72(075)+004.62(075)
ББК 3811я7+А635я7
К888

Рецензенты: канд. тех. наук, доц. Е. В. Г о ш и н;
канд. тех. наук А. М. Ч и н г а е в а

Кудрина, Мария Александровна

К888 Теория информации: практикум / *М.А. Кудрина, К.А. Кудрин;* Министерство науки и высшего образования Российской Федерации, Самарский университет. – Самара: Издательство Самарского университета, 2023. – 1 CD-ROM (1,45 Мб). – Загл. с титул. экрана. – Текст. Изображение: электронные.

ISBN 978-5-7883-1968-1

Практикум направлен на получение обучающимися практических навыков в тематических рамках, охватываемых курсом «Теория информации», предназначен для выполнения лабораторных работ по темам «Моделирование случайных величин с заданным законом распределения», «Алгоритмы кодирования информации», «Сжатие информации», «Помехозащитное кодирование».

Каждая глава сопровождается примерами, упражнениями и вариантами заданий, направленными на повышение качества усвоения материала.

Практикум предназначен для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.03.01 Информатика и вычислительная техника.

Подготовлено на кафедре информационных систем и технологий.

Минимальные системные требования:
тип ЭВМ – x86-64 совместимый;
2,3 ГГц и выше;
ОС Windows 7 и выше;
объем ОЗУ не менее 2 Гб;
объем свободного дискового пространства не менее 10 Мб.

Редакционно-издательская обработка издательства
Самарского университета

Подписано для тиражирования 09.11.2023.
Объем издания 1,45 Мб.
Количество носителей 1 диск.
Тираж 11 дисков.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, САМАРА, МОСКОВСКОЕ ШОССЕ, 34.

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Практикум «Теория информации» является сборником теоретических сведений и заданий для проведения лабораторных работ по курсу «Теория информации». В начале каждой лабораторной работы имеется теоретический материал, необходимый для выполнения работы, приведены подробные примеры решения задач, позволяющие студентам качественно освоить курс.

При подготовке теоретической части авторы руководствовались в первую очередь материалами учебных изданий Лидовского В. В. [1], Фурсова В.А., Козина Н.Е. [2], Гошина Е.В. [3], Чикрина Д.Е. [4] и других авторов.

Практикум подготовлен на основе курса лекций для обучающихся по направлению подготовки 09.03.01 Информатика и вычислительная техника. Первая лабораторная работа посвящена моделированию случайных величин с заданным законом распределения, вторая – алгоритмам кодирования информации, третья – алгоритмам сжатия информации и четвертая– помехозащитным способам кодирования. Практикум предназначен в основном для подготовки бакалавров по направлению 09.03.01 Информатика и вычислительная техника, но может быть полезен и для студентов других специальностей и направлений.

Авторы выражают благодарность студентам Самарского университета Чеснокову Я., Гребенщикову Д., Щурихину Д. за помощь в подготовке отдельных материалов практикума.

1 МОДЕЛИРОВАНИЕ СЛУЧАЙНЫХ ВЕЛИЧИН С ЗАДАННЫМ ЗАКОНОМ РАСПРЕДЕЛЕНИЯ

1.1 Теоретическая часть

На ЭВМ моделирование случайных величин (непрерывных или дискретных), случайных процессов реализуется методами, основанными на использовании равномерно распределённой на отрезке [0, 1] случайной величины.

Наиболее «чисто» эта задача решается методом *нелинейного преобразования*. Суть метода заключается в следующем.

Пусть имеется датчик случайных чисел ξ, имеющих закон распределения $F_{\xi}(x)$. Требуется построить последовательность случайных чисел η с заданным законом распределения $F_{\eta}(y)$.

Рассмотрим монотонно возрастающую функцию $g(x)$, имеющую обратную функцию $g^{-1}(y)$, такую что

$$\eta = g(\xi) \tag{1}$$

Определим закон распределения случайной величины η, полученной в результате функционального преобразования (1):

$$F_{\eta}(y) = P\{\eta < y\} = P\{g(\xi) < y\} = P\{\xi < g^{-1}(y)\} = F_{\xi}(g^{-1}(y)),$$

$$\text{т.е. } F_{\eta}(y) = F_{\xi}(g^{-1}(y)). \tag{2}$$

Если $F_{\xi}(x)$ – закон равномерно распределённой случайной величины, то

$$F_{\eta}(y) = F_{\xi}(g^{-1}(y)) = \begin{cases} 0, & \xi \notin [0,1], \\ g^{-1}(y), & \xi \in [0,1], \end{cases}$$

или

$$F_{\eta}(y) = g^{-1}(y).$$

Откуда имеем

$$g(y) = F_{\eta}^{-1}(y). \qquad (3)$$

Таким образом, для того чтобы равномерно распределённую величину ξ с помощью нелинейного преобразования (1) «превратить» в случайную величину η с заданным законом распределения $F_{\eta}(y)$, необходимо найти обратную функцию $F_{\eta}^{-1}(y)$. Например, пусть $F_{\eta}(x) = 1 - e^{-\lambda x}$ (требуется сгенерировать экспоненциально распределённую случайную величину), тогда $F_{\eta}^{-1}(y) = -\frac{1}{\lambda}\ln(1-y)$, откуда имеем $\eta = -\frac{1}{\lambda}\ln(1-\xi)$, где ξ – равномерно распределённая случайная величина. На рисунке 1 представлена геометрическая интерпретация метода нелинейного преобразования.

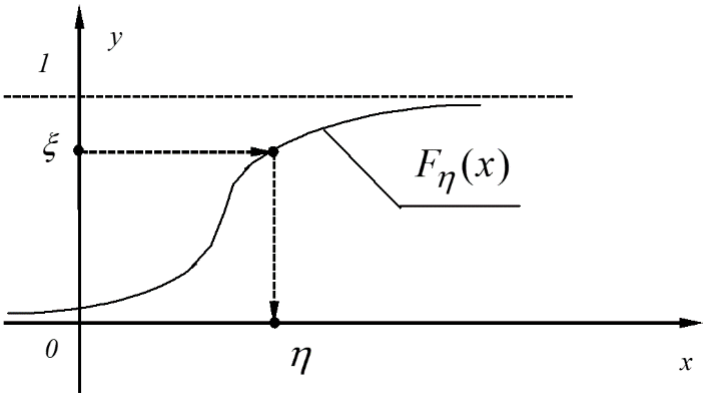


Рис. 1. Моделирование случайных величин методом нелинейного преобразования

Метод нелинейного преобразования – теоретически строгий метод моделирования случайных последовательностей. Он позволяет получать случайные числа даже в тех случаях, когда величина

определена на бесконечных интервалах числовой оси (как для экспоненциальной случайной величины, когда $\xi \in (0,\infty)$).

Однако в тех случаях, когда для интегрального закона распределения невозможно получить обратную функцию, данный метод применить невозможно. Например, для гауссовской случайной величины не существует представления через элементарные функции не только обратной функции, но и выражения интегрального закона распределения.

В этих случаях приходится использовать иные методы. Так для моделирования гауссовской случайной величины используется центральная предельная теорема.

Пусть ξ_i – независимые, равномерно распределенные на отрезке [0, 1] случайные величины с математическим ожиданием $M(\xi) = 1/2$ и дисперсией $D(\xi) = 1/12$, тогда случайная величина $\eta = \sum_{i=1}^{n} \xi_i$ будет иметь при $n \to \infty$ нормальный закон распределения с $M(\eta) = 1/2$ и $D(\eta) = D(\xi)/n$. После нормализации случайной величины $\eta = \sum_{i=1}^{n} \xi_i$ имеем

$$\eta = \sqrt{\frac{12}{n}} \sum_{i=1}^{n} (\xi_i - 1/2), \tag{4}$$

при этом полученная случайная величина имеет нулевое математическое ожидание и единичную дисперсию.

При $n = 12$

$$\eta = \sum_{i=1}^{12} \xi_i - 6. \tag{5}$$

Причем как показали статистические исследования, даже при $n = 12$ случайная величина (5) хорошо согласуется с гипотезой о нормальности распределения случайной величины η.

1.2 Лабораторная работа № 1

1. Смоделировать СВ, плотность распределения вероятностей которой представлена на рисунке 2.

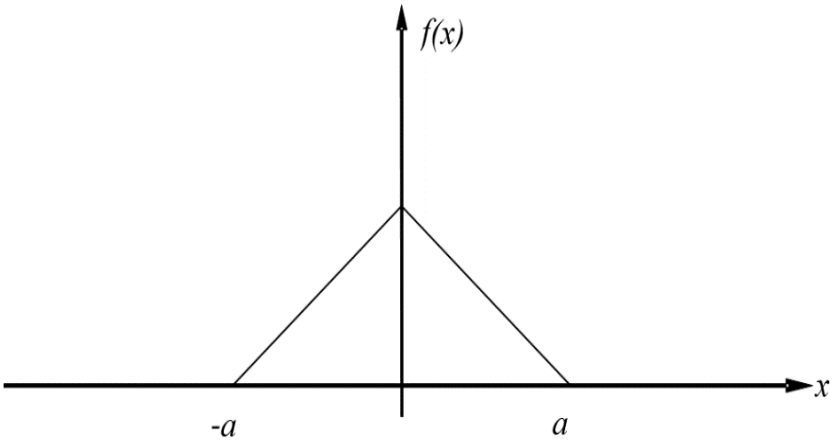
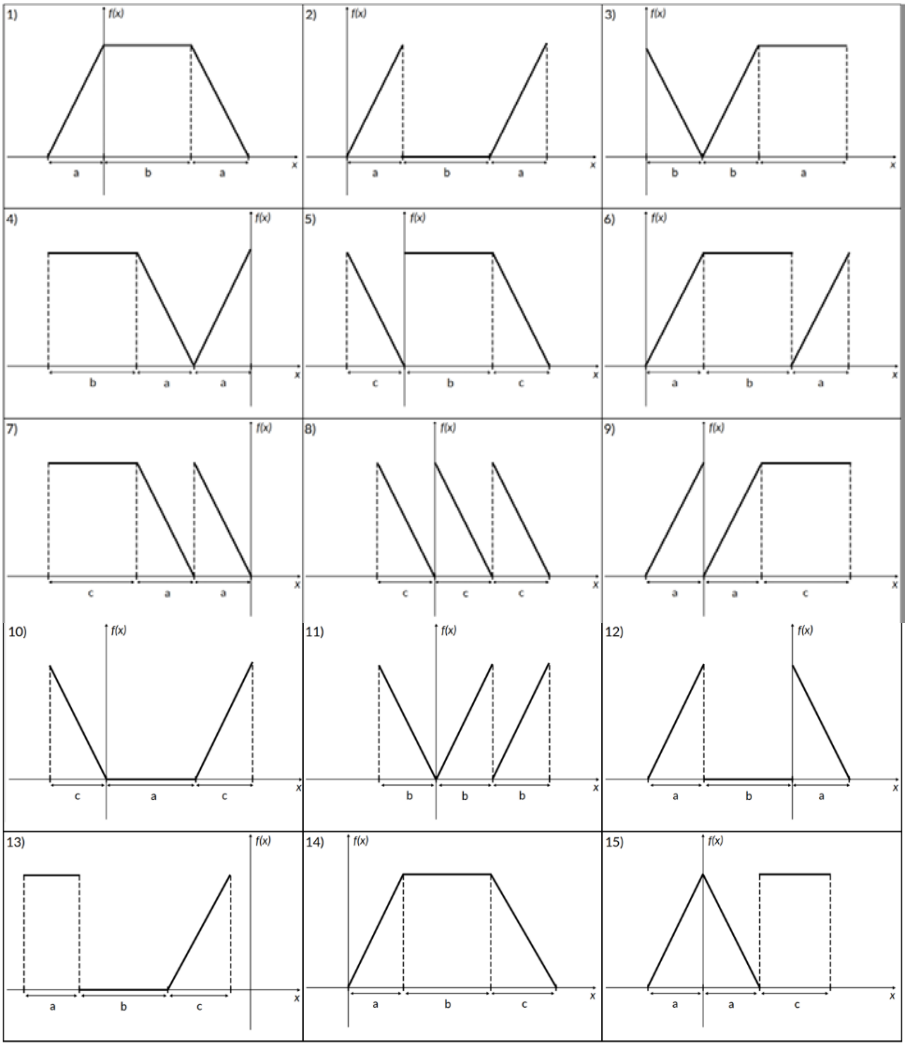
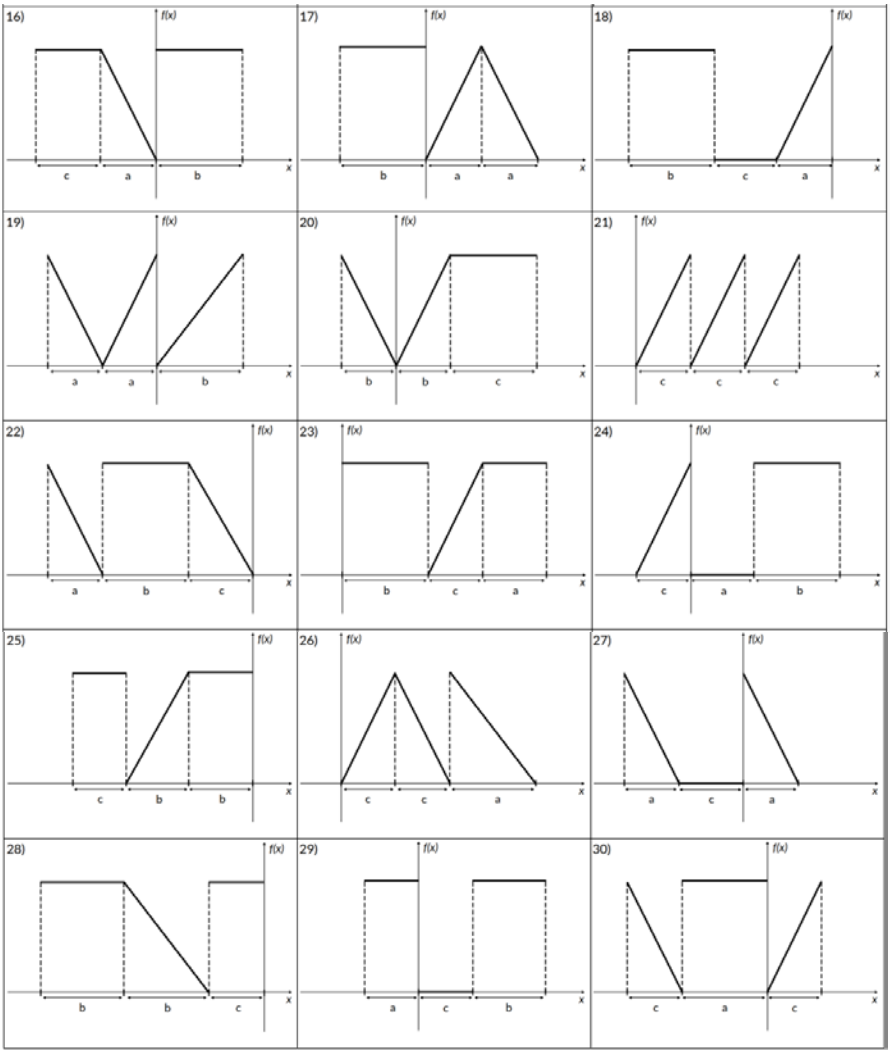


Рис. 2. Плотность распределения вероятностей случайной величины

2. Получить индивидуальное задание у преподавателя.
3. Смоделировать СВ с заданным законом распределения.
4. Смоделировать СВ с экспоненциальным законом распределения.
5. Смоделировать СВ с нормальным законом распределения (центрированным и нормированным, т.е. с MX=0 и DX=1).
6. Оформить отчёт.

Варианты заданий




16)
f(x)
c
a
b
x
17)
f(x)
b
a
a
x
18)
f(x)
b
c
a
x
19)
f(x)
a
a
b
x
20)
f(x)
b
b
c
x
21)
f(x)
c
c
c
x
22)
f(x)
a
b
c
x
23)
f(x)
b
c
a
x
24)
f(x)
c
a
b
x
25)
f(x)
c
b
b
x
26)
f(x)
c
c
a
x
27)
f(x)
a
c
a
x
28)
f(x)
b
b
c
x
29)
f(x)
a
c
b
x
30)
f(x)
c
a
c
x

2 АЛГОРИТМЫ КОДИРОВАНИЯ ИНФОРМАЦИИ

2.1 Теоретическая часть

Для адаптации сообщений к параметрам канала связи применяются следующие методы кодирования:

– *преобразующее кодирование* (осуществляющее преобразование алфавитов);

– *эффективное кодирование* (минимизирующее избыточность сообщения);

– *помехоустойчивое кодирование*.

В общем случае кодирование– это преобразование алфавита исходного сообщения $U\{u_i\}$, i=1…M в алфавит кодовых символов $R\{r_j\}$, j=1…K.

При этом обычно, но не обязательно, $K<M$ или даже $K<<M$.

В рамках данной лабораторной работы познакомимся с преобразующими и эффективными методами кодирования. Примеры помехозащитного кодирования будут рассмотрены в 4-ой лабораторной работе.

Примером *преобразующего кодирования* является запись исходного сообщения, представляемого в виде числа, в выбранной позиционной системе счисления разрядности m:

$$u_i = \sum_{k=0}^{n-1} r_k m^k = r_{n-1}m^{n-1} + r_{n-2}m^{n-2} + \cdots + r_0 m^0 .$$

Алфавитное неравномерное двоичное кодирование

При алфавитном неравномерном двоичном кодировании с равной длительностью элементарных сигналов 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 – 1 (есть напряжение в линии) и 0 (напряжение отсутствует – пауза) при этом одинаковы ($\tau_0 = \tau_1 = \tau$). За счёт 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 τ, 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. 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 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. Но число бит в коде (длина кода), как указывалось ранее, соответствует количеству

информации, содержащейся в знаке h_i. С учётом коррекции вероятностей знаков алфавита, поскольку в алфавит включен «пробел» как самостоятельный символ, получаем таблицу кодов (см. таблицу 1).

Пример декодирования:

111100 11000 1000 101100 11000 100 1111100000 111100 100 1101000 100 101000

(дневной дозор)

1100001010100110100100111000100101000011011001000

(лукоморье)

Таблица 1. Коды букв при неравномерном кодировании

Буква	**Код**	p_i	k_i
пробел	000	0,145	3
о	100	0,095	3
е,ё	1000	0,074	4
а	1100	0,064	4
и	10000	0,064	5
т	10100	0,056	5
н	11000	0,056	5
с	11100	0,047	5
р	10100	0,041	6
в	101100	0,039	6
л	110000	0,036	6
к	110100	0,029	6
м	111000	0,026	6
д	111100	0,026	6
п	1010000	0,024	7
у	1010100	0,021	7
я	1011000	0,019	7
ы	1011100	0,016	7
з	1101000	0,015	7
ь,ъ	1101100	0,015	7

Продолжение таблицы 1

б	1110000	0,015	7
г	1110100	0,014	7
ч	1111000	0,013	7
й	1111100	0,01	7
х	10101000	0,009	8
ж	10101100	0,007	8
ю	10110000	0,006	8
ш	10110100	0,006	8
ц	10111000	0,004	8
щ	10111100	0,003	8
э	11010000	0,003	8
ф	11011000	0,002	8

Методы эффективного кодирования

Код будем считать эффективным или оптимальным, если каждый элементарный символ будет передавать максимальную информацию. А это, согласно свойству энтропии, имеет место только в случае равновероятности состояний, поэтому в основе оптимального кодирования лежит требование: элементарные символы в закодированном тексте должны встречаться в среднем с одинаковой частотой.

Изложим способ построения кода, удовлетворяющего поставленному выше условию, который известен под названием кода Шеннона-Фано [5, 6].

Код Шеннона-Фано

Для кодирования используется следующий алгоритм:

1) символы выстраиваются в порядке убывания вероятности их появления;

2) все символы делятся на две приблизительно равновероятные группы. Для первой группы на первом месте комбинации ставится «0», для второй «1»;

3) каждая полученная группа рассматривается по отдельности, и к ней применяется пункт 2, пока каждое из подмножеств не будет состоять из одного символа.

Пример составления кода Шеннона-Фано:

Символ	p_i	y_1	y_1	y_1	y_1	***код***
A	1/4	0	0			00
B	1/4	0	1			01
C	1/8	1	0	0		100
D	1/8	1	0	1		101
E	1/16	1	1	0	0	1100
F	1/16	1	1	0	1	1101
G	1/16	1	1	1	0	1110
H	1/16	1	1	1	1	1111

В таблице 2 приведены коды букв алфавита при кодировании методом Шеннона-Фано.

Таблица 2. Коды букв при кодировании методом Шеннона-Фано

Буква	**Код**	p_i	k_i
пробел	000	0,145	3
о	001	0,095	3
е,ё	0100	0,074	4
а	0101	0,064	4
и	0110	0,064	4
т	0111	0,056	4
н	1000	0,056	4
с	1001	0,047	4
р	10100	0,041	5
в	10101	0,039	5
л	10110	0,036	5
к	10111	0,029	5
м	11000	0,026	5
д	110010	0,026	6

Продолжение таблицы 2

п	110011	0,024	6
у	110100	0,021	6
я	110110	0,019	6
ы	110111	0,016	6
з	111000	0,015	6
ь,ъ	111001	0,015	6
б	111010	0,015	6
г	111011	0,014	6
ч	111100	0,013	6
й	1111010	0,01	7
х	1111011	0,009	7
ж	1111100	0,007	7
ю	1111101	0,006	7
ш	11111100	0,006	8
ц	11111101	0,004	8
щ	11111110	0,003	8
э	111111110	0,003	9
ф	111111111	0,002	9

Средняя информация, содержащаяся в одной букве текста, то есть энтропия на букву:

$$E = -\sum_{i=1}^{32} p_i \log_2 p_i = -0{,}145 \log 0{,}145 - \ldots - 0{,}002 \log \log 0{,}002 \approx$$

$\approx 4{,}4082.$

Среднее число элементарных символов на букву при неравномерном кодировании:

$$k = \sum_{i=1}^{32} k_i p_i = 3 \cdot 0{,}145 + 3 \cdot 0{,}095 + \ldots + 8 \cdot 0{,}002 \approx 5{,}02 .$$

Среднее число элементарных символов на букву при кодировании методом Шеннона-Фано:

$$k_{ШФ} = \sum_{i=1}^{32} k_i p_i = 3 \cdot 0{,}145 + 3 \cdot 0{,}095 + ... + 9 \cdot 0{,}002 \approx 4{,}46 .$$

Разделив энтропию E на k, получаем среднюю информацию на один двоичный символ при неравномерном кодировании:

$$I_S = \frac{E}{k} = \frac{4{,}4082}{5{,}02} \approx 0{,}8776 .$$

Средняя информация на один двоичный символ при кодировании методом Шеннона-Фано:

$$I_S^{ШФ} = \frac{E}{k_{ШФ}} = \frac{4{,}4082}{4{,}46} \approx 0{,}9884 .$$

Таким образом, информация на один двоичный символ при кодировании методом Шеннона-Фано близка к своему верхнему пределу 1, и, следовательно, данный код весьма близок к оптимальному.

Метод Хаффмана

Метод Хаффмана (*Huffman*) разработан в 1952 г. Он более практичен и по степени сжатия не уступает методу Шеннона-Фано [7]. Код строится при помощи двоичного (бинарного) дерева.

Алгоритм:

1) символы сортируются по убыванию вероятности их появления;

2) две последние позиции списка объединяются в одну. Ей приписывается вероятность, равная сумме вероятностей породивших ее позиций;

3) повторяем пункты 1 и 2 до тех пор, пока не останется одна позиция с вероятностью, равной единице;

4) строится кодовое дерево.

Правила построения кодового бинарного дерева:

1) корню ставится в соответствие вероятность = 1;

2) каждому узлу приписываются два потомка с вероятностями, которые участвовали в формировании значения вероятности обрабатываемого узла;

3) пункт 2 повторяется до тех пор, пока терминальные узлы не станут однозначно соответствовать вероятностям исходных знаков.

После постройки дерева нужно приписать каждой из ветвей, исходящих из родительских узлов, значения 0 или 1. Условимся левым (более тяжелым) узлам приписывать 1, а правым – 0.

Код символа – это число, получаемое при обходе ветвей от корня к листу, соответствующему данному символу.

Пример:

A	0,22	A	0,22	A	0,22	S_3	0,26	S_4	0,32	S_5	0,42				
B	0,2	B	0,2	B	0,2	A	0,22	S_3	0,26	S_4	0,32				
C	0,16	C	0,16	C	0,16	B	0,2	A	0,22	S_3	0,26				
D	0,16	D	0,16	D	0,16	C	0,16	B	0,2						
E	0,1	E	0,1	S_2	0,16	D	0,16								
F	0,1	F	0,1	E	0,1										
G	0,04	S_1	0,06												
H	0,02														

S_6	0,58	S_7	1
S_5	0,42		

На рисунке 3 приведено бинарное дерево Хаффмана.

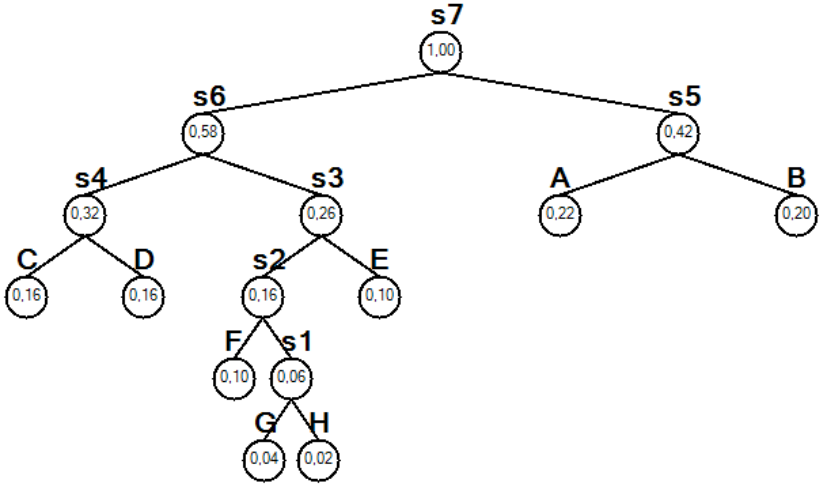


Рис. 3. Дерево Хаффмана

Символ	Код
A	01
B	00
C	111
D	110
E	100
F	1011
G	10101
H	10100

Для методов Хаффмана и Шеннона-Фано каждый раз вместе с собственно сообщением нужно передавать и таблицу кодов.

Универсальные коды

Универсальное кодирование применяется, когда декодеру приходится работать с данными по мере поступления [3].

Для построения кодов Элиаса не требуется использовать вероятности появления символов, в этом они выигрывают у кодов Хаффмана и Шеннона. Данные коды могут быть использованы для шифрования, так как по скорости построения и декодирования эти коды сильно выигрывают у большинства остальных. Однако длины кодов Элиаса зачастую превышают длины обычных двоичных представлений чисел, что накладывает ограничения на область их использования.

Гамма-код Элиаса

От англ. *Elias gamma code* — это универсальный код для кодирования положительных целых чисел, разработанный Питером Элиасом. Он обычно используется при кодировании целых чисел, максимальное значение которых не может быть определено заранее, или для сжатия данных, в которых маленькие значения встречаются более часто, чем большие.

Алгоритм построения гамма-кода Элиаса

1. Записать число N в двоичном представлении.

2. Перед двоичным представлением дописать нули, количество которых на единицу меньше количества битов двоичного представления числа.

Декодирование гамма-кода Элиаса

1. Считать все нули, встречающиеся до первой единицы. Пусть N — количество этих нулей.

2. Считать N+1 цифр целого числа.

Пример построения гамма-кода Элиаса

Пример кодирования числа 38:

1. Двоичное представление числа 38: 100110_2.

2. Дописать перед числом пять нулей: γ(38)=00000 100110.

Пример декодирования гамма-кода Элиаса:

Декодируем последовательность битов 00000100110

1. Считываем нули до первой единицы, N=5.

2. Считываем единицу и 5 бит, следующих за ней. Получаем 100110_2=38.

Дельта-код Элиаса

От англ. *Elias delta code* — это модификация гамма-кода Элиаса.

Алгоритм построения дельта-кода Элиаса

1. Сосчитать L — количество значащих бит в двоичном представлении числа N. Закодировать L с помощью гамма-кода Элиаса.

2. Дописать к гамма-коду L справа двоичное представление числа N без старшей единицы.

Декодирование дельта-кода Элиаса

1. Сосчитать M — количество нулей во входном потоке до первой единицы.

2. Считать в L число, представленное следующими M+1 битами.

3. Считать L-1 двоичных разрядов и дописать к ним слева единицу. Это будет двоичное представление закодированного числа.

Пример построения дельта-кода Элиаса

Пример кодирования числа N=38:

1. Двоичное представление числа 38: 100110_2. Данное двоичное представление имеет L=6 двоичных разрядов. Гамма-код Элиаса числа 6: 00110.

2. Дописать к гамма-коду L справа двоичное представление числа N без старшей единицы: δ(38)=00110 00110.

Пример декодирования гамма-кода Элиаса:

Декодируем последовательность битов 0011000110.

1. Считываем нули до первой единицы, M=2.

2. Считываем в L 2+1=3 бита, следующих за нулями. Получаем 110. Это число 6.

3. Считываем следующие 6-1=5 бит из потока. Это 00110. Добавляем в начало единицу: 100110. Это число 38.

Омега-код (рекурсивный код) Элиаса

От англ. *Elias omega code* — это универсальный код для кодирования положительных целых чисел, который рекурсивно кодирует префикс, именно поэтому он так же известен, как *рекурсивный код Элиаса*.

Алгоритм построения омега-кода Элиаса

1. Записать 0 в выходной поток.

2. Если N – единица, алгоритм останавливается.

3. В выходной поток слева заносится двоичное представление числа N.

4. Переменной N присваивается новое значение – количество только что записанных в выходной поток бит минус один.

5. Вернуться к шагу 2.

Декодирование омега-кода Элиаса

1. Записать в переменную N единицу (N=1).

2. Проверяем первый слева бит. Если он равен единице, то считываем группу бит длиной (N+1). Записываем в N число, двоичное представление которого равно этой группе бит.

Если он равен нулю, то N и есть наше число.

3. Удаляем считанную группу из последовательности и переходим к шагу 2.

Пример построения омега-кода Элиаса

Пример кодирования числа N=38:

1. Записываем в выходной поток 0.

2. N=38.

3. Слева в выходной поток записывается 100110_2 – двоичное представление числа 38.

4. $N = 6–1 = 5$.

5. Слева в выходной поток записывается 101_2 – двоичное представление числа 5.

6. $N = 3–1 = 2$.

7. Слева в выходной поток записывается 10_2 – двоичное представление числа 2.

8. $N = 2–1 = 1$. Остановка алгоритма.

ω(38)= 10 101 100110 0.

Пример декодирования омега-кода Элиаса:

Декодируем последовательность битов 101011001100.

1. N=1.

2. Проверяем первый слева бит. Это единица. Считываем N+1=1+1=2 бита слева, это 10. N=2.

3. Удаляем 10 из входной последовательности. Остается 1011001100.

4. Проверяем первый слева бит. Это единица. Считываем N+1=2+1=3 бита слева, это 101. N=5.

5. Удаляем 110 из входной последовательности. Остается 1001100.

6. Проверяем первый слева бит. Это единица. Считываем N+1=5+1=6 бит слева, это 100110. N=38.

7. Удаляем 100110 из входной последовательности. Остается 0.

8. Проверяем первый слева бит. Это ноль. N=38 – ответ.

2.2 Лабораторная работа № 2

1.1. Раскодировать фразу A (неравномерным методом).

1.2. Раскодировать фразу B (методом Шеннона-Фано).

2. Для символов фразы C построить таблицы неравномерного кода, кода Шеннона-Фано и кода Хаффмана.

3. Найти:

- энтропию алфавита;

- среднее число элементарных символов на букву при неравномерном кодировании;

- среднее число элементарных символов на букву при кодировании Шеннона-Фано;

- среднюю информацию на один двоичный символ при неравномерном кодировании;

- среднюю информацию на один двоичный символ при кодировании методом Шеннона-Фано.

4. Построить гамма-код, дельта-код и омега-код Элиаса для чисел из таблицы с заданием.

Варианты заданий

Вариант №1

A =

1100010000011010000101001000000011000100000011100001000111001110011000010001111001100010000100

B =

0101000111000010100011001001001000111001111011010111000011000000110000001000010000001100110011001011100101100111

C = НАСЛЕДНИК ВСЕХ СВОИХ РОДНЫХ. ДРУЗЬЯ ЛЮДМИЛЫ И РУСЛАНА!

Вариант №2

A =

1110000100010100011000010000100110100000010110011011100101
0000101010011111000100001100000001110100110000110011010001
100

B =

1110110011010000111001000011010011111000100000111110001101
0110000101010100111100010010100100001001111010000111110001
0110111000100011100111111101

C = НО ТАК И БЫТЬ — РУКОЙ ПРИСТРАСТНОЙ ПРИМИ СОБРАНЬЕ ПЕСТРЫХ ГЛАВ,

Вариант №3

A =

1001111001100011001101001000001010100111100110011110001000
0000011000100000011100001011100110000100

B =

1011100110110010010000110000111111001010101001111010111111
0101011010001010001100110011100100011110111000110100101100
1101001111001

C = ХОТЕЛ БЫ Я ТЕБЕ ПРЕДСТАВИТЬ ЗАЛОГ ДОСТОЙНЕЕ ТЕБЯ,

Вариант №4

A =

1101100010001111001001010000001000010110011001100010111001
1111000000111001000111110011110001100111000001011001000101
0001100010001010011100101100 0

B =

0111110111000101010001111111010100101001011100110101111001
0001001011100110100001111111000100110000001100110011001011
11101001100110110

C = ЧЕМУ-НИБУДЬ И КАК-НИБУДЬ, ТАК ВОСПИТАНЬЕМ, СЛАВА БОГУ,

Вариант №5
A =
10100100101000001011001110010100001011100101010001100010101001100000000100101000001100010001110100100111100100101100110001100100000
B =
01101010101011000000101000101100110010100101001100100110101101001110110
C = НО ВРЕДЕН СЕВЕР ДЛЯ МЕНЯ. КОГДА ЖЕ ЮНОСТИ МЯТЕЖНОЙ

Вариант №6
A =
11000100000101001000111000010000001010000101000100001111001000101001110000001000100000011101001001011001001010001000100000101001101100
B =
11001101101011001010111000001111010111001110110100110000110101101001110110
C = ПОЛУСМЕШНЫХ, ПОЛУПЕЧАЛЬНЫХ, ПРОСТОНАРОДНЫХ, ИДЕАЛЬНЫХ,

Вариант №7
A =
10110011100100000011100001011100110000100000110001100000111000100011100101001000
B =
11001110100011010101010011100010110011000010101010110010 0

0100111111011100100101000100110101110110111000010110001000
011011111000
C = У НАС НЕМУДРЕНО БЛЕСНУТЬ. ОНЕГИН БЫЛ ПО МНЕ-
НЬЮ МНОГИХ

Вариант №8
A =
1110000101110011000010000010011010010011000010000001111001
0001110010110001010010000000111100011001110010010110000001
0101001010010100011100
B =
1010100010010101110010110100000111010110111101100010000111
101101111011001
C = ПОЭЗИИ ЖИВОЙ И ЯСНОЙ, ВЫСОКИХ ДУМ И ПРО-
СТОТЫ;

Вариант №9
A =
1110010011010001100011001100010000100000010100001001101001
0000110001010100110000100000100011101001000
B =
1100110100101000100101010011100101111000110101110000011111
0110010111100000100000111101011100111011010011000011010110
1001110110
C = ОГНЕМ НЕЖДАННЫХ ЭПИГРАММ. ЛАТЫНЬ ИЗ МОДЫ
ВЫШЛА НЫНЕ:

Вариант №10
A =
1100010000010000000110100110011000011011001110010011000101
1110000010100110011010010101100100 0

B =
100001101011101011000001101000000110101010101100000110101
0110111100000001011110111101001101111011000010100001011

C = ОСТРИЖЕН ПО ПОСЛЕДНЕЙ МОДЕ, ОН ПО-ФРАНЦУЗ-
СКИ СОВЕРШЕННО

Вариант №11

A =
101001100110100000111001011001000011110010001010010001100
001101100111001010010110010101001011000010100000110000101
1000011110010000

B =
101010001100111001011011110110110010101111010001101111000
1001001101111101001111101

C = КОСНУТЬСЯ ДО ВСЕГО СЛЕГКА, С УЧЕНЫМ ВИДОМ
ЗНАТОКА

Вариант №12

A =
110001000000100000000101001001110100111100110000001100010000
00100101001101001010001011100110000010000

B =
110010101001101001110110110111101100010011011000110101000
1000010000011101100110101001101000110011111001

C = ИЛИ БЛИСТАЛИ, МОЙ ЧИТАТЕЛЬ; ТАМ НЕКОГДА ГУ-
ЛЯЛ И Я:

Вариант №13

A =
111000101110000001011001110010000001101000010100100000101 0
000101000100011010010100011001110011000100000101000010011
0001000001110001100100011100 0

B =
01110011011011100110111001000100001000001000010111001110100110110111011010111110100111010000011000001111000111011
C = ПЕЧАЛЬНО ПОДНОСИТЬ ЛЕКАРСТВО, ВЗДЫХАТЬ И ДУМАТЬ ПРО СЕБЯ:

Вариант №14
A =
11000110011100101001010100101000010000110000100000111000100110000111100011001100010000100001000
B =
10101100101000000011011110000101000111010110111101100110000001011110111101001101110111101111
C = ЧТО ОН УМЕН И ОЧЕНЬ МИЛ. МЫ ВСЕ УЧИЛИСЬ ПОНЕМНОГУ

Вариант №15
A =
10011000110000010101001010110010000001001010000101010011100101001000110000100
B =
11001101001010001001100100001000011011000000111010110111101100010000111101000110000110011110100011101100
C = НЕ ОТХОДЯ НИ ШАГУ ПРОЧЬ! КАКОЕ НИЗКОЕ КОВАРСТВО

Вариант №16
A =
10110000011010010010100010000111100100101000100000011100001011100110000100000101001000111000110001000
B =
10101110111000110100110001010001000111010000011001010100100

10100111011001111110100010011100001001010001111110011111101
C = КОГДА НЕ В ШУТКУ ЗАНЕМОГ, ОН УВАЖАТЬ СЕБЯ ЗАСТАВИЛ

Вариант №17
A =
11100101001010001000110000110100110000010100001001100001101000110000110000011010000010011110010000110001100011001111001011100011001010010000
B =
11001110100011011110110011100100110101100011001111001000101010100101000110011111100101
C = РОДИЛСЯ НА БРЕГАХ НЕВЫ, ГДЕ, МОЖЕТ БЫТЬ, РОДИЛИСЬ ВЫ

Вариант №18
A =
10011000000101100101110010101100100001110100110000010000000101000011000010101001010100
B =
11001010100110100111011001111101000011101110110010101110000000011001011101011011010011101110000111000010110111101001101110111011111000
C = МОГ ИЗЪЯСНЯТЬСЯ И ПИСАЛ; ЛЕГКО МАЗУРКУ ТАНЦЕВАЛ

Вариант №19
A =
11000100000100000001010010011101001111001100000110001000000100101001101001010001011100110000100000
B =
10000010000111010011101001000001100111010001101100100100

1111001110110000010001000001110110011010100110100011001111
111001
C = (СУДЕЙ РЕШИТЕЛЬНЫХ И СТРОГИХ) УЧЕНЫЙ МАЛЫЙ, НО ПЕДАНТ:

Вариант №20
A =
1001100011000001010100101011001000000100101000010101001111
0010100100011000011100
B =
1001001111000100001011000011001000001100110011011101101000
1101001011000100001001110110011
C = ОСТРИЖЕН ПО ПОСЛЕДНЕЙ МОДЕ, ОН ПО-ФРАНЦУЗ-СКИ СОВЕРШЕННО

Вариант №21
A =
1010000100010100010001011001001111001111000100001101000000
1001010100010010100110001000001001110000110110010110001111
0011000100001100001110010110000
B =
1111111110100110010001101000000110101010101100011011111110
0000010010100111101011110001011001000101010100101001000010
0001111001110110
C = ПОЗВОЛЬТЕ ПОЗНАКОМИТЬ ВАС: ОНЕГИН, ДОБРЫЙ МОЙ ПРИЯТЕЛЬ,

Вариант №22
A =
1011000001010000111001000010101000100001100101001010001000
0011110001000111001101001010100101100000

B =
1001011110100010010110101110101000110011001101101110000101
1001010001011100000111001001101000100001011100101111110101
0101110110
C = ИЛИ БЛИСТАЛИ, МОЙ ЧИТАТЕЛЬ; ТАМ НЕКОГДА ГУЛЯЛ И Я:

Вариант №23
A =
1111001010001010100111010010011111000001110100110000110011
0100000010011100101001100110000111001011000000110100011001
1010010100010111001010010111001110001
B =
1110101101111011000100000110111001101100010001100100100100
1111011001110110000111100010110010011010100011010001111010
00101
C = ХОТЕЛ БЫ Я ТЕБЕ ПРЕДСТАВИТЬ ЗАЛОГ ДОСТОЙНЕЕ ТЕБЯ,

Вариант №24
A =
1010000101000100001011001000110100011000010000000101100110
0111000001110010110000111100110000011100101100101100011010
0001100110001100010111001110001
B =
0011100101000010110111001000110100110010010111110001100001
0000100000111010110111101100 01
C = МОГ ИЗЪЯСНЯТЬСЯ И ПИСАЛ; ЛЕГКО МАЗУРКУ ТАНЦЕВАЛ

Вариант №25

A =

1011001011100000101010011100010100010001010010000000111100
1010001010100111010010010110000000011100111000100010100010
100110110010110000

B =

1100011011100010101100101000001111111100111001000110011110
1000100101111010001011001100000100011001100110000011011000
0101010011000

C = НЕ МЫСЛЯ ГОРДЫЙ СВЕТ ЗАБАВИТЬ, ВНИМАНЬЕ ДРУЖБЫ ВОЗЛЮБЯ,

Вариант №26

A =

1010010111000001011000001011100010001010001101001001011001
1011000001110010100100101000100101011001000111000000101000
0010011100101001010100101000010000

B =

1010100010111001101000110110010001101000100000111010110111
1101100010000111010011000100000 1

C = КОГДА НЕ В ШУТКУ ЗАНЕМОГ, ОН УВАЖАТЬ СЕБЯ ЗАСТАВИЛ

Вариант №27

A =

1010000100010100010001111000001100010000111000000111000010
1110011000010000010100101000100000001010000101010010100100
0000

B =

1000010110010111110100110011011010110001000110000011011011
1110001011000001100100

C = ПРИШЛА ЕВГЕНИЮ ПОРА, ПОРА НАДЕЖД И ГРУСТИ НЕЖНОЙ,

Вариант №28

A =

10100110011010000011100101100100001111001000101001000110000110110011100101001011001010100101100001010000011000010110000011100100000

B =

10101000100101011100101101000001110101101111011000100001110110111011001

C = С ГЕРОЕМ МОЕГО РОМАНА БЕЗ ПРЕДИСЛОВИЙ, СЕЙ ЖЕ ЧАС

Вариант №29

A =

10000101100110011000000101000110011100111001000101000111100100000110000111001011000

B =

10101100101000000011011110000101000111010110111011001100000010111101111010011011101111101111

C = ДОСТОЙНЕЕ ДУШИ ПРЕКРАСНОЙ, СВЯТОЙ ИСПОЛНЕННОЙ МЕЧТЫ,

Вариант №30

A =

11010010011000010001100010000000101101001011001000111110010111000110010100011000001010000100111100100111010011000101010011000010000111001101100

B =

10000010001111111100111011000010000100000111010010010011001101100100001

C = НЕБРЕЖНЫЙ ПЛОД МОИХ ЗАБАВ, БЕССОННИЦ, ЛЕГКИХ ВДОХНОВЕНИЙ,

№ вар.	Задание № 4 (коды Элиаса)
1	22
2	15
3	45
4	27
5	34
6	19
7	23
8	43
9	35
10	51
11	31
12	26
13	18
14	44
15	41
16	52
17	39
18	25
19	21
20	42
21	20
22	33
23	24
24	32
25	48
26	40
27	17
28	36
29	50
30	30

3 СЖАТИЕ ИНФОРМАЦИИ

3.1 Теоретическая часть

Цель сжатия – уменьшение количества бит, необходимых для хранения или передачи заданной информации, что дает возможность передавать сообщения более быстро и хранить более экономно.

Сжатие данных не может быть большим некоторого теоретического предела.

Доказано, что среднее количество бит, приходящихся на один символ первичного алфавита, не может быть меньшим, чем энтропия этого алфавита, т.е. $ML(X) \geq EX$ для любой дискретной случайной величины X и любого ее кода.

С ростом длины сообщения n при кодировании методом Шеннона-Фано всего сообщения целиком среднее количество бит на единицу сообщения будет сколь угодно мало отличаться от энтропии единицы сообщения. Подобное кодирование практически не реализуемо из-за того, что с ростом длины сообщения трудоемкость построения этого кода становится недопустимо большой. Кроме того, такое кодирование делает невозможным отправку сообщения по частям, что необходимо для непрерывных процессов передачи данных. Дополнительным недостатком этого способа кодирования является необходимость отправки или хранения собственно полученного кода вместе с его исходной длиной, что снижает эффект от сжатия.

Арифметическое кодирование

Арифметическое кодирование разработано в 70-х гг. XX века.

При арифметическом кодировании текст представляется вещественными числами в интервале от 0 до 1. По мере кодирования текста отображающий его интервал уменьшается, а количество битов для его представления возрастает. Очередные символы текста сокращают величину интервала исходя из значений их вероятностей, определяемых моделью. Более вероятные символы делают это

в меньшей степени, чем менее вероятные, и, следовательно, добавляют меньше битов к результату.

Алгоритм кодирования:

1) рассчитываются частоты встречаемости букв;

2) берется отрезок единичной длины и разбивается отрезками длины, равной вероятностям букв;

3) кодирование:

```
min := 0;
max := 1;
for i:=1 to N do begin
 C := БуквавСлове[i];
 Δ := max-min;
 max := min + Δ*Конец(C);
 min := min + Δ*Начало(C);
end;
```

Итоговый интервал [*min*, *max*) и есть сжатая фраза. Однако декодировщику нет необходимости знать значения обеих границ итогового интервала, полученного от кодировщика. Даже единственного значения, лежащего внутри него, уже достаточно. Это число всегда меньше 1 и больше 0, значит достаточно хранить его дробную часть (мантиссу). Однако, чтобы завершить процесс, декодировщику нужно вовремя распознать конец текста. Для устранения неясности мы должны обозначить завершение каждого текста специальным символом *EOF*, известным и кодировщику, и декодировщику.

Декодирование:

```
N := СжатыеДанные;
i:=0;
while NOT Конец do begin
 C :=СимволСоотвествующийИнтервалу(N);
 БуквавСлове[i]:=C;
 Δ :=Конец(C) - Нач ало(C);
 N := (N - Начало(C))/Δ;
```

```
i:=i+1;
end;
```

Пример: закодируем слово «КОЛОКОЛЬНЯ».

Шаг 1. Рассчитываем частоты встречаемости букв:

О	0,3
К	0,2
Л	0,2
Ь	0,1
Н	0,1
Я	0,1

Шаг 2. Берем отрезок единичной длины и делим на части пропорционально частотам встречаемости букв (см. рисунок 4).

Буква	**Начало**	**Конец**
О	0	0,3
К	0,3	0,5
Л	0,5	0,7
Ь	0,7	0,8
Н	0,8	0,9
Я	0,9	1

Рис. 4. Интервалы на единичном отрезке, соответствующие символам

Кодирование:

Буква	**Δ**	***min***	***max***
		0	1
К	1	0,3	0,5
О	0,2	0,3	0,36
Л	0,06	0,33	0,342
О	0,012	0,33	0,3336

К	0,0036	0,33108	0,3318
О	0,00072	0,33108	0,331296
Л	0,000216	0,331188	0,3312312
Ь	4,32E-05	0,33121824	0,33122256
Н	4,32E-06	0,331221696	0,331222128
Я	4,32E-07	**0,331222085**	**0,331222128**

На рисунке 5 приведена иллюстрация процесса арифметического кодирования.

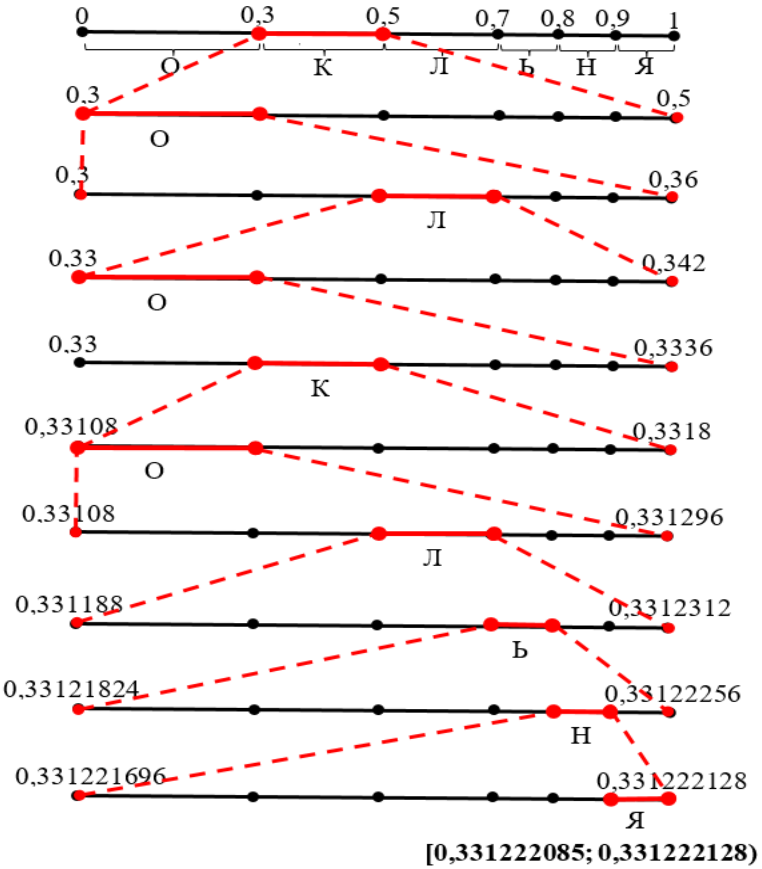


Рис. 5. Иллюстрация процесса разбиения отрезков при арифметическом кодировании

Сжатым сообщением может являться любое число из интервала [0,331222085; 0,331222128). Например, число 0,331222086. Мантисса данного числа в двоичном виде: 10011101111100000110001000110 (29 бит).

Но более целесообразно выбирать число, содержащее наименьшее количество значащих разрядов после запятой. В данном примере – это число 0,3312221.

Декодирование:

N	Буква	Δ
0,331222086	К	0,2
0,156110424	О	0,3
0,52036808	Л	0,2
0,1018404	О	0,3
0,339468	К	0,2
0,19734	О	0,3
0,6578	Л	0,2
0,789	Ь	0,1
0,89	Н	0,1
0,9	Я	0,1

Эффективность арифметического кодирования растет с ростом длины сжимаемого сообщения (для кодирования Хаффмана или Шеннона-Фано этого не происходит). Хотя арифметическое кодирование дает обычно лучшее сжатие, чем кодирование Хаффмана, оно пока используется на практике сравнительно редко, т.к. оно появилось гораздо позже и требует больших вычислительных ресурсов.

При сжатии данных все ранее рассмотренные методы (метод Шеннона-Фано, Хаффмана, арифметический метод) требуют двух проходов. Первый – для сбора частот символов, используемых как приближенные значения вероятностей символов, и второй – для собственно сжатия.

Метод блокирования

Метод блокирования применяется для повышения степени сжатия. По выбранному $\varepsilon > 0$ можно выбрать такое s, что если разбить все сообщение на блоки длиной s (всего будет n / s блоков), то кодированием Шеннона-Фано или методом Хаффмана таких блоков, рассматриваемых как единицы сообщения, можно сделать среднее количество бит на единицу сообщения большим энтропии менее, чем на ε.

Пример. Пусть дискретная случайная величина X принимает значение 0 с вероятностью 3/4 и значение 1 с вероятностью 1/4.

Тогда $E(X) = -3/4 \log_2(3/4) - 1/4 \log_2(1/4) \approx 0{,}811$ бит/симв.

Минимальное кодирование здесь – это коды 0 и 1 с длиной 1 бит каждый. При таком кодировании количество бит в среднем на единицу сообщения равно 1.

Разобьем сообщение на блоки длины 2. Закон распределения вероятностей и кодирование для 2-мерной д.с.в. $\bar{X} = (X_1, X_2)$ методом Шеннона-Фано:

$\bar{X}$	**00**	**01**	**10**	**11**
p	9/16	3/16	3/16	1/160
код	0	10	110	111
$L(\bar{X})$	1	2	3	3

При таком кодировании количество бит в среднем на единицу сообщения будет

$ML(\bar{X}) = (1 \cdot 9/16 + 2 \cdot 3/16 + 3 \cdot 3/16 + 3 \cdot 1/16)/2 = 27/32 = 0{,}84375$.

Т.е. меньше, чем для неблочного кодирования. Для блоков длины 3 количество бит в среднем на единицу сообщения можно сделать $\approx 0{,}823$, для блоков длины 4 – $\approx 0{,}818$ и т.д.

Пример. Найти среднее количество бит на единицу сообщения $ML(\bar{X})$ для блочного кода с использованием сжатия Шеннона-Фано

и Хаффмана. Длина блока 2 символа. Д.с.в. X описывается следующим законом распределения вероятностей:

X_i	p_i
A	0,4
B	0,2
C	0,4

Энтропия случайной величины X равна

$$E(X) = -0{,}4\log_2 0{,}4 - 0{,}2\log_2 0{,}2 - 0{,}4\log_2 0{,}4 \approx 1{,}522 .$$

Сначала закодируем символы обычным способом методами Шеннона-Фано и Хаффмана.

Метод Шеннона-Фано:

символы	p_i	**код**	$L(\bar{X})$
A	0,4	0	1
C	0,4	10	2
B	0,2	11	2

$ML_{ШФ}(\bar{X})=(0{,}4\cdot 1+0{,}4\cdot 2+0{,}2\cdot 2)=1{,}6$.

На рисунке 6 приведено бинарное дерево Хаффмана.

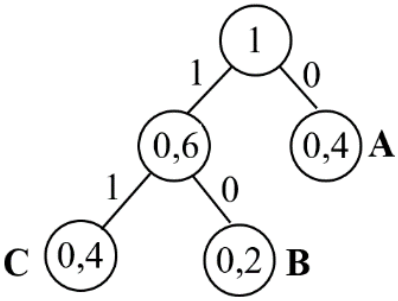


Рис. 6. Дерево Хаффмана

Коды: A – 0, B – 10, C – 11.

$ML_X(\bar{X})=(0{,}4\cdot1+0{,}4\cdot2+0{,}2\cdot2)=1{,}6$.

Теперь составим таблицу возможных сочетаний символов (блоки длиной 2 символа) и вероятностей их появления в сообщении:

блоки	p_i
AA	0,16
AB	0,08
AC	0,16
BA	0,08
BB	0,04
BC	0,08
CA	0,16
CB	0,08
CC	0,16

Составим код Шеннона-Фано для блочного кодирования:

блоки	p_i	**код**	$L(\bar{X})$
AA	0,16	00	2
AC	0,16	010	3
CA	0,16	011	3
CC	0,16	100	3
AB	0,08	101	3
BA	0,08	1100	4
BC	0,08	1101	4
CB	0,08	1110	4
BB	0,04	1111	4

Среднее количество бит на единицу сообщения для блочного кода с использованием сжатия Шеннона-Фано:

$ML_{ШФбл}(\bar{X})=(0{,}16\cdot11+0{,}08\cdot15+0{,}04\cdot4)/2=1{,}56$.

Составим коды блоков методом Хаффмана:

AA	0,16	AA	0,16	AA	0,16	S_3	0,2	S_4	0,32	S_4	0,32
AC	0,16	AC	0,16	AC	0,16	AA	0,16	S_3	0,2	S_5	0,32
CA	0,16	CA	0,16	CA	0,16	AC	0,16	AA	0,16	S_3	0,2
CC	0,16	CC	0,16	CC	0,16	CA	0,16	AC	0,16	AA	0,16
AB	0,08	S_1	0,12	S_2	0,16	CC	0,16	CA	0,16		
BA	0,08	AB	0,08	S_1	0,12	S_2	0,16				
BC	0,08	BA	0,08	AB	0,08						
CB	0,08	BC	0,08								
BB	0,04										

S_6	0,36	S_7	0,64	S_8 1
S_4	0,32	S_6	0,36	
S_5	0,32			

На рисунке 7 приведено бинарное дерево Хаффмана.

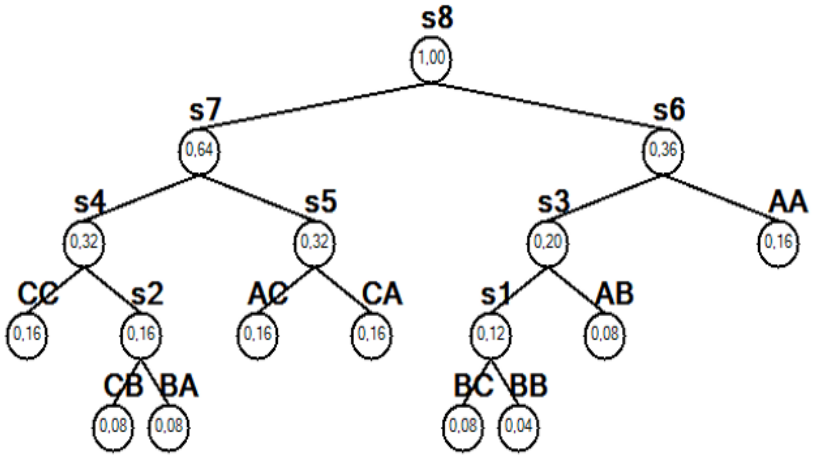


Рис. 7. Дерево Хаффмана (метод блокирования)

Коды блоков для блочного кодирования методом Хаффмана будут выглядеть следующим образом:

блоки	p_i	код	$L(\bar{X})$
AA	0,16	00	2
AC	0,16	101	3
CA	0,16	100	3
CC	0,16	111	3
AB	0,08	010	3
BA	0,08	1100	4
BC	0,08	0111	4
CB	0,08	1101	4
BB	0,04	0110	4

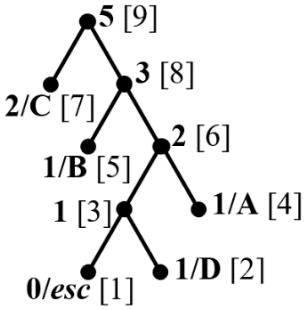
Адаптивный алгоритм сжатия Хаффмана

Адаптивный алгоритм Хаффмана является модификацией обычного алгоритма Хаффмана сжатия сообщений. Он позволяет не передавать таблицу кодов и ограничиться одним проходом по сообщению как при кодировании, так и при декодировании.

Суть адаптивного алгоритма состоит в том, что при каждом сопоставлении символу кода изменяется внутренний ход вычислений так, что в следующий раз этому же символу может быть сопоставлен другой код, т.е. происходит адаптация алгоритма к поступающим для кодирования символам. При декодировании происходит аналогичный процесс.

В адаптивном алгоритме сжатия Хаффмана используется упорядоченное бинарное дерево. Бинарное дерево называется *упорядоченным*, если его узлы могут быть перечислены в порядке неубывания веса. Перечисление узлов происходит по ярусам снизу вверх и слева направо в каждом ярусе. Узлы, имеющие общего родителя, находятся рядом на одном ярусе.

На рисунке 8 приведен пример упорядоченного дерева Хаффмана. Рядом с узлами дерева на данном рисунке приведены их веса, а в квадратных скобках приведены порядковые номера узлов дерева при перечислении.



№ узла	1	2	3	4	5	6	7	8	9
Вес узла	0	1	1	1	1	2	2	3	5

Рис. 8. Пример упорядоченного дерева Хаффмана

Правила построения и упорядочивания бинарного дерева Хаффмана

1. В начале работы алгоритма дерево кодирования содержит только один специальный символ, всегда имеющий частоту 0. Он необходим для занесения в дерево новых символов. Обычно этот символ называют *escape*-символом (<*esc*>).

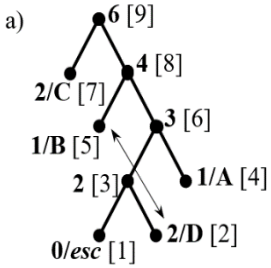
2. Левые ветви дерева помечаются 0, а правые – 1.

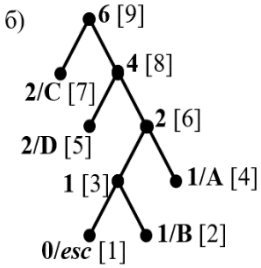
3. При нарушении упорядоченности дерева (после добавлении нового листа или изменении веса имеющегося листа) его необходимо упорядочить.

4. Чтобы упорядочить дерево необходимо поменять местами два узла: узел, вес которого нарушил упорядоченность, и последний из следующих за ним узлов меньшего веса. После перемены мест узлов необходимо пересчитать веса всех их узлов-предков.

Пример. Рассмотрим пример упорядочивания дерева Хаффмана. Возьмем в качестве исходного дерева дерево, представленное на рисунке 9. Пусть на вход поступил очередной символ *D*. Вес листа *D* увеличился и стал равен 2. На рисунке 9 (а) приведен порядок перечисления узлов и последовательность весов, которая не является неубывающей. Следовательно, двоичное дерево не

упорядочено. Для упорядочивания необходимо поменять местами лист *D* (нарушивший упорядоченность) и лист *B* (последний из меньших по весу узлов при перечислении). Результат упорядочивания приведен на рисунке 9 (б).





№ узла	1	2	3	4	5	6	7	8	9
Вес узла	0	2	2	1	1	3	2	4	6

№ узла	1	2	3	4	5	6	7	8	9
Вес узла	0	1	1	1	2	2	2	4	6

Рис. 9. Пример упорядочивания дерева Хаффмана

Правила кодирования

1. Элементы входного сообщения считываются побайтно.

2. Если входной символ присутствует в дереве, в выходной поток записывается код, соответствующий последовательности нулей и единиц, которыми помечены ветки дерева при проходе от корня дерева к данному листу. Вес данного листа увеличивается на единицу. Веса узлов-предков корректируются. Если дерево становится неупорядоченным – упорядочивается.

3. Если очередной символ, считанный из входного сообщения при сжатии, отсутствует в дереве, в выходной поток записывается набор нулей и единиц, которыми помечены ветки бинарного дерева при движении от корня к *escape*-символу, а затем 8 бит *ASCII*-кода нового символа. В дерево вместо *escape*-символа добавляется ветка:

родитель, два потомка. Левый потомок становится *escape*-символом, правый – новым добавленным в дерево символом. Веса узлов-предков корректируются, а дерево при необходимости упорядочивается.

Например, если мы имеем дерево Хаффмана как на рисунке 10 (а), и очередной символ на входе *C*, которого еще нет в дереве, то выходной код нового символа будет 00'*C*', где '*C*' – *ASCII*-код символа *C*. В дерево добавляется ветка с новым символом (см. рисунок 10 (б)), и дерево упорядочивается (см. рисунок 10 (в)).

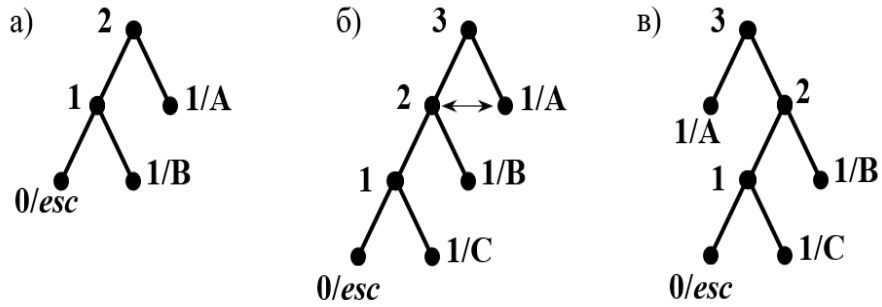


Рис. 10. Пример добавления нового символа в дерево Хаффмана

Пример. Рассмотрим пример работы алгоритма архивации на следующей входной последовательности символов: ***ABCCDDDDBB***.

На рисунке 11 приведены деревья Хаффмана для каждого шага.

Вх. данные	Вых. данные	№ дерева	Длина кода, бит
A	'A'	1	8
B	0'B'	2	9
C	00'C'	3	10
C	101	4	3
D	100'D'	5	11
D	1101	6	4
D	10	7	2
D	0	8	1
B	1101	9	4
B	111	10	3

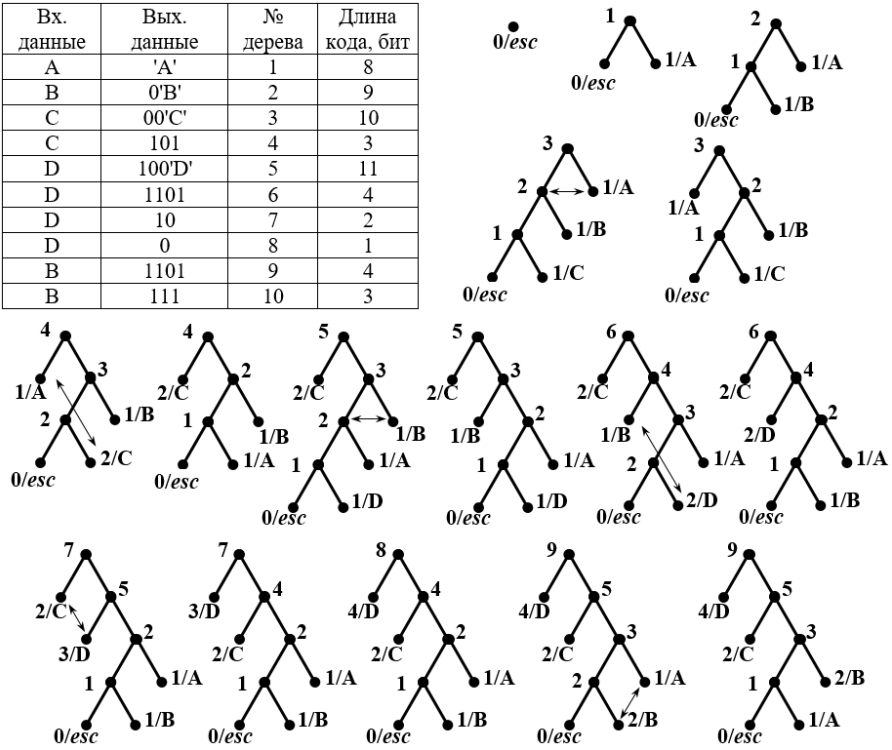


Рис. 11. Пример работы адаптивного алгоритма сжатия Хаффмана

Сообщение из 10 символов занимает 80 бит. Сжатые данные: **'*A*'0'*B*'00'*C*'101100'*D*'11011001101111**. Сжатое сообщение занимает 55 бит.

Для сравнения подсчитаем, сколько бит займет данное сообщение, закодированное обычным методом Хаффмана. Таблица кодировки и кодовое дерево приведены на рисунке 12.

Символ	Вероятность	Код
D	0,4	0
B	0,3	11
C	0,2	101
A	0,1	100

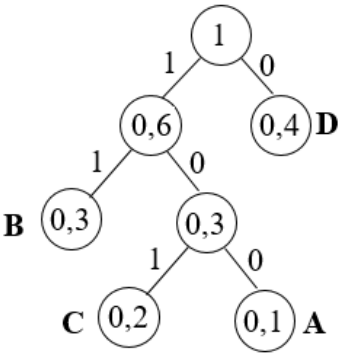


Рис. 12. Дерево Хаффмана

Закодированное сообщение: **1001110110100001111** (всего 19 бит). Кроме того, следует учесть, что вместе с закодированным сообщением передается и таблица кодировки. В данном случае это как минимум еще 4 байта *ASCII*-кодов символов плюс соответствующие им коды Хаффмана. Всего 68 бит.

Правила декодирования

1. Элементы входного сообщения считываются побитно.

2. Каждый раз при считывании 0 или 1 происходит перемещение от корня вниз по соответствующей ветке бинарного дерева Хаффмана до тех пор, пока не будет достигнут какой-либо лист дерева.

3. Если достигнут лист, соответствующий символу, в выходное сообщение записывается *ASCII*-код данного символа. Вес листа увеличивается на 1, веса узлов-предков корректируются, дерево при необходимости упорядочивается.

4. Если же достигнут *escape*-символ, из входного сообщения считываются 8 следующих бит, соответствующих *ASCII*-коду нового символа. В выходное сообщение записывается *ASCII*-код дан-

ного символа. В дерево добавляется новый символ, веса узлов-предков корректируются, затем при необходимости производится его упорядочивание.

Пример. Рассмотрим процесс декодирования сообщения **'*A*'0'*B*'0100'*C*'11**. В начале декодирования дерево Хаффмана содержит только *escape*-символ с частотой 0. С раскодированием каждого нового символа дерево перестраивается (см. рисунок 13). На выходе получим последовательность ***ABBCBB***.

Вх. данные	Вых. данные	№ дерева
'A'	A	1
0'B'	B	2
01	B	3
00'C'	C	4
1	B	5
1	B	6

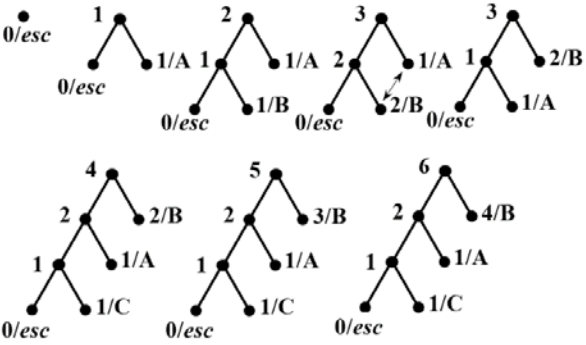


Рис. 13. Пример работы алгоритма декомпрессии

Адаптивное кодирование Хаффмана имеет ряд особенностей при своей работе. Одна из них связана с хранением значений кодов символов при кодировании длинных последовательностей. Дело в том, что языки высокого уровня, обычно, оперируют с числами максимум в 32 бит, в которых можно записывать числа от 0 до 4294967296, т.е. можно работать с файлами размером 4Гб. Но иногда этих значений недостаточно и приходится вырабатывать дополнительные меры по предотвращению переполнения счётчиков символов. Правильное решение может заключаться в накоплении битов кода в связанном списке, к которому можно добавлять новые узлы. Следующая проблема связана с определением конца передачи. Ведь для декомпрессора не существует ни одного способа определения

конца передачи. Есть, по крайней мере, два способа решения этой проблемы. Компрессор может добавить к началу сжатых данных длину файла, делая их на пару байт длиннее. Либо можно в начале кодирования добавить в дерево помимо листа с escape-символом еще один вспомогательный лист с нулевой частотой, код которого сообщит декомпрессору о конце передачи данных.

Но, несмотря на указанные сложности, он часто применяется на практике, например, в протоколе *V*.32 передачи данных по модему со скоростью 14400 бод или в программе compact операционной системы *UNIX*. Этот алгоритм наиболее эффективен для сжатия текстов или программных файлов. Изображения лучше сжимаются другими алгоритмами сжатия.

Словарно-ориентированные алгоритмы

Метод *LZ77*

Методы Шеннона-Фано, Хаффмана и арифметическое кодирование обобщенно называются *статистическими* методами. Словарные алгоритмы носят менее математически обоснованный, но более практичный характер.

Алгоритм *LZ*77 был опубликован в 1977 г. Разработан израильскими математиками Якобом Зивом (*Ziv*) и Авраамом Лемпелом (*Lempel*). Многие программы сжатия информации используют ту или иную модификацию *LZ*77.

Основная идея *LZ*77 состоит в том, что второе и последующие вхождения некоторой строки символов в сообщении заменяются ссылками на более ранние вхождения.

*LZ*77 использует «скользящее» по сообщению окно, разделенное на две неравные части (словарь и буфер). Первая, большая по размеру, часть – «словарь», включает уже просмотренную часть сообщения. Вторая, намного меньшая, часть, является «буфером», содержащим еще не закодированные символы входного потока.

Чтобы добиться сжатия, алгоритм пытается заменить фрагмент сообщения на указатель в содержимое словаря.

Обычно размер окна составляет несколько килобайт, а размер буфера – не более ста байт.

Правила кодирования

Алгоритм пытается найти в словаре максимально длинный фрагмент, совпадающий с начальной частью буфера.

Пример. Если в словаре находится строка «*AABABCBCDE*», а в буфере строка «*ABCDE*», то алгоритм найдет фрагменты «*A*» и «*AB*» в словаре, но выдаст совпадение «*ABC*», поскольку оно самое длинное. Совпадающая подстрока «*BCDE*» тоже не подходит, поскольку ищется совпадение от начала буфера.

Словарь (10)	**Буфер (5)**
AABABCBCDE	***ABCDE***

Алгоритм *LZ77* выдает коды, состоящие из трех элементов (см. рисунок 14):

– смещение совпадающего с началом содержимого буфера фрагмента в словаре относительно начала словаря (левого края);

– длина совпадающей подстроки;

– *ASCII*-код первого символа буфера, следующего за совпадающей подстрокой.

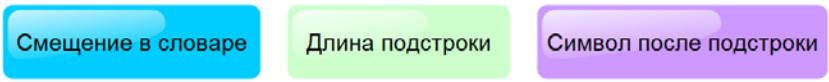


Рис. 14. Структура кода алгоритма *LZ77*

Если в начале буфера находится символ, которого нет в словаре, в выходной поток поступает код <**0, 0, *ASCII*-код символа**>.

Поскольку обязательным элементом кода является «первый символ буфера, следующий за совпадающей подстрокой», длина

совпадающей подстроки в коде не может быть больше, чем (размер буфера – 1).

Если в буфере остался единственный символ, то по этой же причине он кодируется как **<0, 0, *ASCII*-код символа>**, даже если в словаре он присутствует.

Если в словаре имеется несколько одинаковых подстрок, совпадающих с началом содержимого буфера, то ссылка дается на первую от начала словаря подстроку.

После того как код поступил в выходной поток, алгоритм сдвигает влево все содержимое окна на длину совпадающей подстроки +1 и одновременно считывает столько же символов из входного потока в буфер.

Пример. Закодировать по алгоритму *LZ77* строку «*AABABCABCDABCABBCBA*». Размер словаря – 8 символов, а буфера – 5.

Словарь (8)								Буфер (5)					Код
								A	A	B	A	B	<0, 0, 'A'>
							A	A	B	A	B	C	<7, 1, 'B'>
					A	A	B	A	B	C	A	B	<6, 2, 'C'>
		A	A	B	A	B	C	A	B	C	D	A	<5, 3, 'D'>
B	A	B	C	A	B	C	D	A	B	C	A	B	<1, 4, 'B'>
B	C	D	A	B	C	A	B	B	C	B	A		<0, 2, 'B'>
A	B	C	A	B	B	C	B	A					<0, 0, 'A'>

В последней строчке буква «*A*» берется не из словаря, т.к. она последняя в сообщении и после неё нет буквы, которую можно вставить в третий элемент кода.

Длина кода вычисляется следующим образом: смещение в словаре – число в диапазоне от 0 до (размер словаря – 1), длина подстроки – число от 0 до (размер буфера – 1). Следовательно, длина двоичного кода смещения будет округленным в большую сторону

$\log_2$(размер словаря), а длина двоичного кода для длины подстроки будет округленным в большую сторону $\log_2$(размер буфера). Символ кодируется 8 битами.

В последнем примере длина полученного кода равна $7\times([\log_2(8)]+[\log_2(5)]+8])=7\times(3+3+8)=98$ бит против $19\times8=152$ бит исходной длины строки.

Для декодирования достаточно знать только размер словаря.

Пример. Распаковать сообщение, сжатое с помощью алгоритма *LZ*77, размер словаря – 8 символов. Сообщение: <0, 0, '*A*'>, <7, 1, '*B*'>, <6, 2, '*C*'>, <5, 3, '*D*'>, <1, 4, '*B*'>, <0, 2, '*B*'>, <0, 0, '*A*'>.

Словарь (8)								**Код**	**Фраза в выходной поток**
								<0, 0, '*A*'>	***A***
							A	**<7, 1, '*B*'>**	***AB***
					A	***A***	***B***	**<6, 2, '*C*'>**	***ABC***
		A	***A***	***B***	***A***	***B***	***C***	**<5, 3, '*D*'>**	***ABCD***
B	***A***	***B***	***C***	***A***	***B***	***C***	***D***	**<1, 4, '*B*'>**	***ABCAB***
B	***C***	***D***	***A***	***B***	***C***	***A***	***B***	**<0, 2, '*B*'>**	***BCB***
A	***B***	***C***	***A***	***B***	***B***	***C***	***B***	**<0, 0, '*A*'>**	***A***

Получаем фразу: *AABABCABCDABCABBCBA*.

Декодирование кодов *LZ*77 проще и быстрее их получения, т.к. не нужно осуществлять поиск в словаре.

Недостатки *LZ*77:

1) с ростом размеров словаря скорость работы алгоритма-кодера пропорционально замедляется;

2) кодирование одиночных символов очень неэффективно.

Метод *LZSS*

В 1982 г. Сторером (*Storer*) и Шиманским (*Szimanski*) на базе *LZ*77 был разработан алгоритм *LZSS*, который отличается от *LZ*77 производимыми кодами.

Код, выдаваемый *LZSS*, начинается с однобитного префикса, различающего собственно код от незакодированного символа. Если префиксный бит 0 – после него идет незакодированный символ, если 1 – пара (смещение, длина подстроки) (см. рисунок 15). Смещение и длина подстроки такие же, как и для *LZ*77. В *LZSS* окно сдвигается ровно на длину найденной подстроки или на один символ, если не найдено вхождение подстроки из буфера в словарь.

Если префиксный бит = 0, то длина кода равна 9 битам, (префиксный бит + 8 бит *ASCII*-кода символа).

Если префикс = 1, то смещение – число в диапазоне от 0 до (размер словаря – 1), длина подстроки – число от 1 до (размер буфера). Соответственно длина двоичного кода равна:

$1 + [\log_2(\text{размер словаря})] + [\log_2(\text{размер буфера})]$.

Рис. 15. Структура кода алгоритма *LZSS*

Пример. Закодировать по алгоритму *LZSS* строку «*AABABCABCDABCABBCBA*».

Словарь (8)								Буфер (5)					Код	Длина кода
								A	***A***	***B***	***A***	***B***	**0'*A*'**	**9**
							A	***A***	***B***	***A***	***B***	***C***	**1<7, 1>**	**7**
						A	***A***	***B***	***A***	***B***	***C***	***A***	**0'*B*'**	**9**
					A	***A***	***B***	***A***	***B***	***C***	***A***	***B***	**1<6, 2>**	**7**
				A	***A***	***B***	***A***	***B***	***C***	***A***	***B***	***C***	**0'*C*'**	**9**
			A	***A***	***B***	***A***	***B***	***C***	***A***	***B***	***C***	***D***	**1<5, 3>**	**7**
A	***B***	***A***	***B***	***C***	***A***	***B***	***C***	***D***	***A***	***B***	***C***	***A***	**0'*D*'**	**9**

B	***A***	***B***	***C***	***A***	***B***	***C***	***D***	***A***	***B***	***C***	***A***	***B***	**1<1, 5>**	**7**
B	***C***	***D***	***A***	***B***	***C***	***A***	***B***	***B***	***C***	***B***	***A***		**1<0, 2>**	**7**
D	***A***	***B***	***C***	***A***	***B***	***B***	***C***	***B***	***A***				**1<2, 1>**	**7**
A	***B***	***C***	***A***	***B***	***B***	***C***	***B***	***A***					**1<0, 1>**	**7**

Длина полученного кода равна 4×9+7×7=85 бит.

Пример. Распаковать сообщение, сжатое с помощью алгоритма *LZSS*, размер словаря – 8 символов. Сообщение: 0'*A*', 1<7, 1>, 0'*B*', 1<6, 2>, 0'*C*', 1<5, 3>, 0'*D*', 1<1, 5>, 1<0, 2>, 1<2, 1>, 1<0, 1>.

Словарь (8)								**Код**	**Фраза в выходной поток**
								0'*A*'	***A***
							A	**1<7, 1>**	***A***
						A	***A***	**0'*B*'**	***B***
					A	***A***	***B***	**1<6, 2>**	***AB***
				A	***A***	***B***	***A***	**0'*C*'**	***C***
			A	***A***	***B***	***A***	***B***	**1<5, 3>**	***ABC***
		A	***A***	***B***	***A***	***B***	***C***	**0'*D*'**	***D***
A	***B***	***A***	***B***	***C***	***A***	***B***	***C***	**1<1, 5>**	***ABCAB***
B	***A***	***B***	***C***	***A***	***B***	***C***	***D***	**1<0, 2>**	***BC***
B	***C***	***D***	***A***	***B***	***C***	***A***	***B***	**1<2, 1>**	***B***
D	***A***	***B***	***C***	***A***	***B***	***B***	***C***	**1<0, 1>**	***A***
A	***B***	***C***	***A***	***B***	***B***	***C***	***B***		

Получаем фразу: *AABABCABCDABCABBCBA*.

*LZ*77 и *LZSS* обладают следующими очевидными недостатками:

1) невозможность кодирования подстрок, отстоящих друг от друга на расстоянии, большем длины словаря;

2) длина подстроки, которую можно закодировать, ограничена размером буфера.

Если чрезмерно увеличивать размеры словаря и буфера, то это приведет к снижению эффективности кодирования, т.к. с ростом этих величин будут расти и длины кодов для смещения и длины, что сделает коды для коротких подстрок недопустимо большими. Кроме того, резко увеличится время работы алгоритма-кодера.

Метод *LZ*78

В 1978 г. авторами *LZ*77 был разработан алгоритм *LZ*78, лишенный названных недостатков.

*LZ*78 не использует «скользящее» окно, он хранит словарь из уже просмотренных фраз. При старте алгоритма этот словарь содержит только одну пустую строку (строку длины ноль). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока.

Одна из главных проблем *LZ*78 – размер словаря. Он ничем не ограничен и может достигать огромных размеров на больших объемах входных данных. В различных модификациях установлены лимиты на размер словаря: при достижении определенного лимита словарь может «замораживаться» (в него перестают добавляться новые слова), из словаря могут удалять менее используемые или самые старые слова и так далее.

Ключевым для размера получаемых кодов является размер словаря во фразах, потому что каждый код при кодировании по методу *LZ*78 содержит номер фразы в словаре. Из последнего следует, что эти коды имеют постоянную длину, равную округленному в большую сторону двоичному логарифму размера словаря +8 (это количество бит в байт-коде расширенного *ASCII*).

Пример. Закодировать с помощью алгоритма *LZ*78 строку «*AABABCABCDABCABBCBA*», используя словарь длиной 16 фраз.

Входная фраза (в словарь)	Код	Позиция словаря
""		0
"*A*"	0'*A*'	1
"*AB*"	1'*B*'	2
"*ABC*"	2'*C*'	3
"*ABCD*"	3'*D*'	4
"*ABCA*"	3'*A*'	5
"*B*"	0'*B*'	6
"*BC*"	6'*C*'	7
"*BA*"	6'*A*'	8

Указатель на любую фразу такого словаря — это число от 0 до 15, для его кодирования достаточно четырех бит.

В последнем примере длина полученного кода равна 8×(4 + 8)= = 96 бит.

Пример. Распаковать сообщение, сжатое с помощью алгоритма *LZ*78, размер словаря – 16 фраз. Сообщение: 0'*A*', 1'*B*', 2'*C*', 3'*D*', 3'*A*', 0'*B*', 6'*C*', 6'*A*'.

Код на входе	Фраза в словарь	Позиция словаря	Фраза на выход
	""	0	
0'*A*'	"*A*"	1	*A*
1'*B*'	"*AB*"	2	*AB*
2'*C*'	"*ABC*"	3	*ABC*
3'*D*'	"*ABCD*"	4	*ABCD*
3'*A*'	"*ABCA*"	5	*ABCA*
0'*B*'	"*B*"	6	*B*
6'*C*'	"*BC*"	7	*BC*
6'*A*'	"*BA*"	8	*BA*

Получаем фразу: *AABABCABCDABCABBCBA*.

Алгоритмы *LZ*77, *LZ*78 и *LZSS* разработаны математиками и могут использоваться свободно.

3.2 Лабораторная работа № 3

1. Составить таблицу кодов блоков для метода Хаффмана с блокированием. Вероятности букв считать по фрагменту сообщения в задании. Длина блока указана. Вычислить EX, $ML(X)$, $ML(X_{бл})$.

Здесь EX – энтропия алфавита из букв сообщения,

$ML(X)$ – среднее количество элементарных символов на букву при сжатии методом Хаффмана,

$ML(X_{бл})$ – среднее количество элементарных символов на букву при сжатии методом Хаффмана с блокированием.

2. Сжать сообщение адаптивным методом Хаффмана.

3. Сжать сообщение методами *LZ*77, *LZSS*, *LZ*78. Для методов *LZ*77, *LZSS* размер словаря – 10 символов, буфера – 6 символов. Для метода *LZ*78 размер словаря 32 записи.

4. Сжать сообщение из задания №2 арифметическим методом.

5. Распаковать сообщения, сжатые адаптивным методом Хаффмана, методами *LZ*77, *LZSS*, *LZ*78 и арифметическим методом.

Для методов *LZ*77, *LZSS* размер словаря – 10 символов. Для метода *LZ*78 размер словаря – 16 записей. При декодировании таблица состоит из следующих столбцов: «Код», «Словарь» и «Выходной поток».

Задания для раскодировки метода Хаффмана, *LZ*77, *LZSS*, *LZ*78 приведены ниже в таблицах. Арифметическим методом раскодировать свое закодированное слово.

Варианты заданий

№	Задание № 1 (блочный Хаффмана)	Длина блока	Задание № 2 (адаптивный Хаффмана)	Задание № 3 (*LZ*77, *LZSS*, *LZ*78) (словарь – 10, буфер – 6)
1	ББААБББББ	3 симв.	КЕЕЕНООННН	КУКУКУ_КУКУШКА_КУКИШ
2	САСВВАВВВС	2 симв.	АББААСКААС	ЛЯЛЯЛЯ_ЛЯЛЯ_ЯЛИК_МЯЛ
3	ТИИИИКККК	2 симв.	ПРОВППРРРО	ТАРАРА_ТАРТАР_ТАРТ_ТАРА
4	ДДУДУУУУУУ	3 симв.	АППРОПММММ	СЫР_СЫН_СЫРОК_СЫНОК
5	ТОКООКККТК	2 симв.	РККЕАРРООО	ОСЫ_ОСЫ_СЫПЬ_НАСЫПЬ
6	КООКЛЛЛЛЛЛ	2 симв.	СРОССКРРРР	КУСКУС_ КУСАКА_СОБАКА
7	ТТУТТТТТТТ	3 симв.	ОРОПАВРРРР	РОЗА_РОЗАРИЙ_ЗАРЯДКА
8	ТООТТТТТО	3 симв.	РОПВПАРВВВ	ПОЛ_ПОЛОВНИК_ПОЛОВЕЦ
9	СОККККООО	2 симв.	РОРНРПОООО	МУМУМУ_МУКА_МУРКА
10	СТТТТССТТТ	3 симв.	КАВАПВПППА	КОК_КОКЛЮШ_КЛУБ_КЛУБОК
11	ВВВАССССС	2 симв.	ЕНКПКЕКИЕЕ	ВАРВАР_ВАРИТ_ВАРЕНЬЕ
12	ТИИИКТККТ	3 симв.	УКВАУКВСАК	СОКОЛ_СОК_КОЛ_КОЛОСОК
13	БОБББОБОО	3 симв.	ЛПРИРПТОРТ	ПЕС_ПЕСОК_СОКОЛ_СКОЛ
14	КРООРТТТТТ	2 симв.	САВИПВАИИ	РАБ_РАБА_БАК_БАКЕН_БАК
15	БИБББИИИБ	3 симв.	УВАААУУКПУ	ТАРА_ТАРТАР_ТАРЕЛКА_ЕЛКА
16	ДЕЕДКУДДКК	2 симв.	РПЕАКАРРПП	УКУС_КУСКУС_УКСУС_КСИ
17	ГННООМНГ	2 симв.	ГНРНГРНПРР	ДОМ_ДОМИК_ОМИК_МИР
18	КРРРАКККК	2 симв.	ГНРПАНПППП	РИМ_РОМ_МУРОМ_МУРКА
19	КУКУУУУУУУ	3 симв.	ВУКАУВУААА	ОЛОВО_ЛОВЕЦ_ОВЦА_ЦАП
20	РРРРАААААА	3 симв.	КУИРЕИИККК	КАКТУС_ТУСА_ТУЗ_УСА
21	ЛЕЛЕЛЕЕЕЕЕ	3 симв.	СВИВТРИИИИ	ЛОДКА_ЛОДОЧКА_ОЧКИ
22	КЛЛЛККККОО	2 симв.	ДЕИМЕИДДДД	КЛУБ_КЛУБОК_КЛУБНИ
23	РРРООРТТТР	2 симв.	НЕЕИИННЕАА	БОЛОТО_БОЛТ_БОЛЬ_ОЛЯ
24	ККЛКЮВВВВ	2 симв.	ЕАКАККККРАА	ЛАПКИ_ЛАПЫ_ЛАПИТАЛЬ
25	ЛЛИМЛЛЛМИИ	2 симв.	ГОРОНПОРРР	КУКУРУЗА_УРЮК_КРЮК
26	БРБББРРББББ	3 симв.	ВУАКУВАММММ	ДОДО_ДОМ_ДОМИК_МИГ
27	КВКККВВВВВ	3 симв.	УЧЧРККЧУУУ	ЗИГЗАГ_ЗАЗОР_ЗОРКИЙ
28	УККУУКККККК	3 симв.	КЛЮЧЧИИИИИК	ТИКТАК_ТИК_ТАК_ТАКСА
29	ИИММИИИРРР	2 симв.	БАЗАААРРРР	КУРКУЛЬ_КУЛЕК_ЛЕКАЛО
30	ОККОЛТККККК	2 симв.	КРЫЛЛЛЛЫРР	СКЛАД_КЛАД_КЛАДЕЗЬ

№	Задание № 5 на декодирование (адаптивный Хаффмана)
1	'O'0'P'00'П'100'H'11011001001111
2	'R'0'T'01100'N'010111100'D'1001
3	'S'0'D'00'A'1101000'R'011001001
4	'H'0'K'00'N'11100'F'1111110111
5	'D'0'C'00'B'101100'F'11011011011101001
6	'G'0'H'00'F'100'D'000'C'100110100100
7	'V'0'B'00'C'100'N'11000'F'00001101
8	'K'0'N'00'M'101100'H'110111010111111
9	'T'0'Y'00'H'100'G'0010111111111111
10	'K'0'J'00'N'100'M'000'H'0010001
11	'L'0'K'00'M'100'N'01000'B'10010111
12	'Y'0'T'00'R'100'F'01001111101111111
13	'D'0'C'00'V'1110111100'F'100111110
14	'S'0'X'00'C'100'D'010011001001111
15	'K'0'C'00'B'100'V'100110100111110
16	'Z'0'X'00'Y'10110110111100'D'11110
17	'R'0'F'00'T'100'D'101111011111101001
18	'K'0'L'00'N'100'B'0011111110111001
19	'V'0'B'00'N'101000101110000'F'
20	'P'0'O'0100'K'000'M'110110110111110
21	'C'0'X'0100'V'001100'R'10010111111
22	'K'0'H'00'B'101101110000'N'0001111
23	'P'0'O'0100'U'0011110110111100'K'
24	'A'0'B'00'C'10110111100'S'100111110
25	'K'0'L'0100'M'000'N'110110110111110
26	'H'0'B'00'V'100'N'0011111011111101001
27	'D'0'F'00'C'100'S'010011010011100'H'01
28	'K'0'L'0100'F'0111100'V'10011101001
29	'T'0'R'00'F'100'V'00111110110111001
30	'S'0'K'00'T'100'R'10111101110111110

№	**Задание № 5 на декодирование (*LZ*77).** **Словарь – 10 символов**
1	<0,0,в> <0,0,ы> <8,2,в> <6,1,х> <0,0,у> <8,1,о> <0,0,л> <0,0,ь> <0,0, > <1,3,о> <0,0,д>
2	<0,0,д> <0,0,о> <8,2, > <5,2,р> <3,1,г> <3,5,о> <3,1,а>
3	<0,0,к> <0,0,у> <8,2, > <5,4,ш> <0,1,а> <2,4,л> <0,0,а>
4	<0,0,э> <0,0,х> <0,0,о> <0,0, > <6,3,л> <4,1,т> <3,1,л> <6,2,о> <5,1,т> <1,1,л>
5	<0,0,г> <0,0,о> <0,0,л> <0,0, > <7,2,о> <0,0,в> <3,1, > <5,3, > <1,3,и>
6	<0,0,о> <0,0,л> <8,2,о> <6,2, > <3,2,т> <5,5,о> <0,0,к>
7	<0,0,д> <0,0,а> <0,0, > <7,2,р> <6,4,ы> <1,5,р> <1,1,д> <4,1,р>
8	<0,0,т> <0,0,у> <8,2, > <5,2,к> <7,3, > <3,3,а> <0,0,н> <4,1,к> <6,2,к> <3,1,н>
9	<0,0,к> <0,0,а> <0,0,н> <8,1,в> <6,1, > <6,3,т> <0,1,р> <3,1,в> <4,3, > <1,3,а>
10	<0,0,н> <0,0,о> <0,0,с> <0,0, > <7,2,ь> <6,3,а> <2,1,с> <7,1,н> <0,0,и> <5,4,к> <0,0,и>
11	<0,0,к> <0,0,о> <8,2, > <5,2,т> <6,4,и> <2,1, > <6,4,т> <2,1,п>
12	<0,0,к> <0,0,и> <8,2,м> <0,0,о> <0,0,р> <0,0,а> <0,0, > <5,3, > <1,3,е> <1,1,о> <0,0,р>
13	<0,0,р> <0,0,о> <0,0,в> <0,0, > <6,3,н> <3,1, > <7,2,с> <6,4,и> <0,0,к>
14	<0,0,к> <0,0,и> <0,0,т> <0,0, > <6,3,е> <0,0,л> <0,0,ь> <3,1,т> <5,2,о> <5,1,л> <7,1,т>
15	<0,0,к> <0,0,о> <0,0,с> <0,0,а> <0,0, > <6,4,к> <1,1,т> <2,1,с> <3,1,д> <6,4,и> <0,0,к>
16	<0,0,о> <0,0,с> <0,0,а> <0,0, > <6,2,е> <0,0,л> <5,1,с> <6,2,о> <5,4,ь> <0,1,е> <1,1,ь>
17	<0,0,ц> <0,0,а> <0,0,п> <0,0, > <6,2,р> <4,5,п> <0,0,л> <0,0,я> <4,1,п> <6,2,с> <0,0,к> <0,0,а>
18	<0,0,к> <0,0,л> <0,0,ю> <0,0,ч> <0,0, > <5,2,а> <0,0,д> <5,1,л> <6,4,а> <2,1,а> <0,0,н>

19	<0,0,к> <0,0,о> <0,0,л> <0,0, > <6,3,о> <2,4,л> <4,3,н> <4,1,к> <0,1,н>
20	<0,0,к> <0,0,о> <0,0,р> <8,1,в> <0,0,а> <0,0, > <5,3, > <0,0,к> <5,5,о> <4,1,а>
21	<0,0,с> <0,0,и> <0,0,л> <0,0,а> <0,0, > <7,1,и> <3,1,а> <5,4, > <3,2,л> <0,0,о> <1,2,о> <1,1,ь>
22	<0,0,к> <0,0,о> <0,0,с> <0,0,а> <0,0, > <7,2,х> <5,1,р> <4,1,о> <0,5,л> <0,0,о>
23	<0,0,к> <0,0,у> <0,0,с> <0,0,т> <0,0, > <7,2,о> <2,1, > <6,4,т> <2,2,а> <0,0,р> <0,0,ь>
24	<0,0,с> <0,0,и> <0,0,л> <0,0,а> <0,0, > <6,2, > <4,2,с> <0,0,т> <1,2,с> <6,2,л> <0,0,ь>
25	<0,0,р> <0,0,о> <0,0,т> <0,0, > <6,2,г> <6,3,т> <0,1,р> <0,1,т> <6,4,о> <2,1,е> <0,0,ц>
26	<0,0,к> <0,0,л> <0,0,у> <0,0,б> <0,0, > <5,3,м> <4,1,а> <3,1,б> <7,1,м> <4,1,у> <0,0,к>
27	<0,0,б> <0,0,о> <0,0,р> <0,0, > <6,3,е> <0,0,ц> <4,4,щ> <5,2,а> <0,0,р>
28	<0,0,к> <0,0,о> <0,0,р> <8,1,б> <0,0, > <4,5,к> <0,0,а> <2,4,к> <0,0,а>
29	<0,0,т> <0,0,е> <0,0,л> <8,1,ц> <0,0, > <4,3,о> <5,4,е> <0,0,г> <0,0,а> <3,1,г> <7,1,р> <0,0,ь>
30	<0,0,л> <0,0,о> <0,0,т> <8,1,к> <0,0, > <6,4,л> <0,1,т> <6,4,о> <0,0,к>

№	**Задание № 5 на декодирование (*LZSS*).** **Словарь – 10 символов**
1	[0'л'] [0'о'] [0'т'] [1<8,1>] [0'к'] [0' '] [1<6,4>] [1<0,3>] [1<6,4>] [1<0,1>] [0'к']
2	[0'к'] [0'и'] [0'з'] [1<8,1>] [0'л'] [0' '] [1<6,2>] [0'м'] [0'а'] [1<5,1>] [1<7,2>] [0'к'] [1<6,4>] [0'е'] [0'т']
3	[0'к'] [0'а'] [0'н'] [0'в'] [1<7,1>] [0' '] [1<7,2>] [0'т'] [1<5,5>] [0'н'] [0'и'] [0'к'] [1<3,4>] [0'а']
4	[0'л'] [0'и'] [1<8,2>] [0'я'] [0' '] [1<4,2>] [0'с'] [1<6,4>] [0'а'] [1<1,4>] [0'т']
5	[0'с'] [0'и'] [0'л'] [0'а'] [0' '] [1<5,1>] [0'е'] [1<5,1>] [0'о'] [1<5,1>] [1<7,2>] [1<3,1>] [0'ь'] [1<0,1>] [1<6,2>] [0'а']
6	[0'л'] [0'е'] [0'с'] [0' '] [1<6,3>] [0'о'] [0'к'] [1<4,1>] [1<6,4>] [1<2,3>] [1<0,1>] [0'л']
7	[0'л'] [0'о'] [0'к'] [1<8,1>] [0'н'] [0' '] [1<6,4>] [1<2,3>] [0'ь'] [1<1,3>] [1<3,2>] [0'н']
8	[0'р'] [0'о'] [0'с'] [0'т'] [0' '] [1<7,2>] [1<4,1>] [0'л'] [1<5,3>] [0'у'] [1<5,4>] [0'а'] [1<0,1>] [0'ь']
9	[0'с'] [0'о'] [0'т'] [0'ы'] [0' '] [1<5,2>] [0'к'] [1<6,1>] [1<8,1>] [1<6,2>] [1<4,1>] [1<2,1>] [1<0,4>] [1<2,1>] [0'л']
10	[0'л'] [0'о'] [0'с'] [0'к'] [0' '] [1<5,2>] [0'т'] [1<6,4>] [1<4,1>] [0'с'] [1<0,1>] [1<8,1>] [1<5,2>] [0'л']
11	[0'к'] [0'о'] [0'л'] [0'е'] [0'с'] [1<6,1>] [0' '] [1<7,2>] [1<1,3>] [1<4,1>] [1<6,4>] [1<0,2>] [0'к']
12	[0'л'] [0'о'] [0'т'] [1<8,1>] [0' '] [1<5,3>] [1<6,1>] [1<3,2>] [1<4,1>] [1<2,1>] [0'с'] [1<5,2>] [0'л']
13	[0'б'] [0'е'] [0'р'] [1<8,1>] [0'з'] [0'а'] [0' '] [1<3,4>] [0'г'] [1<4,4>] [0'л'] [0'о'] [1<3,1>] [0'а']
14	[0'л'] [0'у'] [0'ж'] [0'а'] [0' '] [1<7,2>] [0'б'] [1<5,2>] [1<6,3>] [1<2,1>] [0'у'] [0'р']
15	[0'л'] [0'у'] [0'к'] [1<8,1>] [0'м'] [0' '] [1<4,3>] [1<6,1>] [1<2,4>] [1<3,1>] [0'м']
16	[0'к'] [0'у'] [1<8,2>] [0'р'] [1<6,1>] [0'з'] [0'а'] [0' '] [1<3,3>] [0'о'] [1<6,1>] [1<4,1>] [1<6,2>] [0'к']

17	[0'к'] [0'о'] [0'р'] [0'м'] [0' '] [1<7,1>] [1<5,1>] [1<6,4>] [1<2,1>] [0'а'] [0'н'] [1<0,1>] [1<2,1>] [1<0,1>] [0'р']
18	[0'к'] [0'и'] [0'л'] [0'ь'] [0' '] [1<5,4>] [1<1,1>] [0'а'] [1<3,4>] [0'о']
19	[0'р'] [0'у'] [0'к'] [0'и'] [0' '] [1<7,2>] [0'л'] [0'о'] [1<5,1>] [1<7,2>] [0'в'] [1<2,2>] [0'й']
20	[0'к'] [0'р'] [0'а'] [0'б'] [0' '] [1<6,4>] [1<4,1>] [1<1,2>] [1<6,4>] [0'к']
21	[0'к'] [0'о'] [0'н'] [0' '] [1<6,3>] [0'и'] [1<5,3>] [1<3,3>] [1<4,3>] [0'н']
22	[0'п'] [0'а'] [0'р'] [1<8,2>] [0'у'] [1<6,2>] [0' '] [1<1,2>] [1<8,2>] [0'м'] [1<4,3>] [0'р'] [1<2,1>] [0'м']
23	[0'р'] [0'е'] [0'з'] [1<8,1>] [0'д'] [0'а'] [0' '] [1<3,3>] [1<5,1>] [0'к'] [1<4,2>] [0'ю'] [1<6,1>] [1<3,2>] [0'к']
24	[0'з'] [0'и'] [0'г'] [1<7,1>] [0'а'] [1<7,1>] [0' '] [1<6,3>] [1<4,1>] [0'р'] [1<4,1>] [1<6,3>] [0'е'] [0'м'] [1<4,4>] [0'ь']
25	[0'п'] [0'а'] [0'р'] [0' '] [1<8,1>] [1<6,1>] [0'м'] [1<4,1>] [1<5,1>] [0'о'] [1<6,2>] [1<0,2>] [0'к'] [1<4,3>] [0'р']
26	[0'р'] [0'а'] [0'м'] [0'п'] [1<7,1>] [0' '] [1<5,3>] [0'и'] [1<0,1>] [1<4,1>] [1<6,4>] [1<1,1>] [1<2,1>] [0'р']
27	[0'м'] [0'у'] [0'р'] [1<7,3>] [0' '] [1<3,3>] [0'к'] [0'а'] [1<4,4>] [0'з'] [0'и'] [0'к']
28	[0'н'] [0'о'] [0'с'] [1<8,1>] [0'к'] [0' '] [1<5,4>] [0'а'] [1<4,1>] [1<0,4>] [0'н'] [1<1,1>] [0'с']
29	[0'к'] [0'о'] [0'л'] [1<8,1>] [0'с'] [0' '] [1<6,3>] [1<1,1>] [1<5,1>] [1<2,2>] [0'а'] [1<1,1>] [1<3,2>] [1<0,1>] [0'л']
30	[0'б'] [0'а'] [0'т'] [0'у'] [1<8,1>] [0' '] [1<6,3>] [1<4,1>] [1<5,1>] [1<2,2>] [0'к'] [0'а'] [1<0,3>] [1<5,2>] [0'н']

№	**Задание № 5 на декодирование (*LZ*78).** **Словарь – 16 фраз**
1	[0'д'] [0'о'] [0'р'] [2'г'] [2' '] [3'о'] [0'г'] [0'а'] [0' '] [7'о'] [3'а'] [9'р'] [2'т']
2	[0'м'] [0'и'] [0'р'] [0' '] [0'п'] [2'р'] [4'т'] [6' '] [0'т'] [2'г'] [0'р']
3	[0'г'] [0'о'] [0'р'] [2'д'] [0' '] [1'о'] [3'а'] [5'р'] [4' '] [3'о'] [0'т']
4	[0'л'] [0'о'] [0'с'] [2'с'] [0'ь'] [0' '] [1'о'] [3'ь'] [6'о'] [8' '] [4'а']
5	[0'л'] [0'е'] [0'с'] [0' '] [1'е'] [3'а'] [4'л'] [2'с'] [0'к'] [0'а'] [7'е'] [3'о'] [0'к']
6	[0'с'] [0'о'] [1'у'] [0'д'] [0' '] [3'д'] [0'н'] [2' '] [6' '] [4'н'] [0'о']
7	[0'б'] [0'а'] [0'з'] [2'р'] [0' '] [1'а'] [0'р'] [5'з'] [4'я'] [5'а'] [0'м'] [6'р']
8	[0'б'] [0'р'] [0'и'] [0'з'] [0' '] [1'р'] [0'а'] [5'б'] [7'р'] [5'р'] [7'б'] [5'а'] [0'р']
9	[0'л'] [0'е'] [0'т'] [0'о'] [0' '] [3'о'] [0'н'] [5'т'] [4'н'] [0'у'] [0'с'] [5'у'] [11'ы']
10	[0'к'] [0'у'] [0'с'] [0'о'] [1' '] [3'о'] [1'о'] [0'л'] [0' '] [6'к'] [9'к'] [4'л']
11	[0'у'] [0'к'] [0'с'] [1'с'] [0' '] [1'к'] [4' '] [2'у'] [3'т'] [0'ы'] [5'к'] [4'т']
12	[0'д'] [0'о'] [0'р'] [2'г'] [0'а'] [0' '] [0'г'] [2'р'] [5' '] [7'о'] [3'о'] [0'д']
13	[0'п'] [0'о'] [0'р'] [0'т'] [0' '] [1'о'] [3'а'] [5'р'] [0'а'] [6'р'] [0'т']
14	[0'т'] [0'о'] [0'р'] [1' '] [0'с'] [2'р'] [4'с'] [6' '] [5'п'] [6'т']
15	[0'т'] [0'о'] [0'н'] [0'и'] [0'к'] [0' '] [1'о'] [3' '] [7'н'] [3'а']
16	[0'с'] [0'и'] [0'л'] [0'а'] [0' '] [3'а'] [1'к'] [4' '] [6'с'] [0'т'] [8'с'] [10'а'] [0'н']
17	[0'т'] [0'о'] [0'с'] [0'к'] [0'а'] [0' '] [3'к'] [5'л'] [5' '] [0'л'] [5'с'] [1'и'] [0'к']
18	[0'к'] [0'о'] [0'с'] [0'т'] [0'ь'] [0' '] [1'о'] [3'а'] [6'о'] [8' '] [2'к'] [0'о']
19	[0'б'] [0'е'] [0'р'] [2'т'] [0' '] [1'е'] [3'е'] [0'г'] [5'б'] [7'г']
20	[0'в'] [0'а'] [0'р'] [1'а'] [3' '] [3'в'] [2'н'] [0'ь'] [0' '] [4'н'] [0'н'] [0'а']
21	[0'н'] [0'о'] [0'с'] [2'к'] [0' '] [2'с'] [4'а'] [5'с'] [4'о'] [0'л']
22	[0'к'] [0'о'] [0'л'] [2'б'] [2'к'] [0' '] [0'б'] [5' '] [7'о'] [1'а'] [0'л']
23	[0'к'] [0'л'] [0'у'] [0'б'] [0' '] [1'л'] [3'б'] [0'о'] [1' '] [4'о'] [0'к']

24	[0'к'] [0'и'] [1'и'] [0'м'] [0'о'] [0'р'] [0'а'] [0' '] [4'о'] [6' '] [9'р'] [0'ж']
25	[0'м'] [0'а'] [1'а'] [0' '] [0'р'] [2'м'] [2' '] [5'а'] [0'к'] [0'и']
26	[0'л'] [0'о'] [0'г'] [2'в'] [2' '] [0'в'] [2'л'] [0' '] [6'о'] [1'к']
27	[0'т'] [0'и'] [0'н'] [0'а'] [0' '] [1'и'] [0'к'] [5'н'] [2'т'] [2' '] [3'и'] [1'к'] [0'и']
28	[0'к'] [0'а'] [0'б'] [2'н'] [0' '] [3'а'] [0'н'] [1'а'] [5'б'] [2'к'] [0'е'] [0'н']
29	[0'з'] [0'а'] [0'р'] [0'я'] [0' '] [1'а'] [3'я'] [0'д'] [0'к'] [2' '] [7'д']
30	[0'к'] [0'л'] [0'а'] [0'д'] [0' '] [0'с'] [1'л'] [3'д'] [5'л'] [8' '] [2'а'] [4'ь'] [0'я']

4 ПОМЕХОЗАЩИТНОЕ КОДИРОВАНИЕ

4.1 Теоретическая часть

Задача обнаружения ошибки может быть решена довольно просто. Достаточно просто передавать каждую букву сообщения дважды. Например, при необходимости передачи слова «*гора*» можно передать «*ггоорраа*». При получении искаженного сообщения, например, «*гготрраа*» с большой вероятностью можно догадаться, каким было исходное слово. Конечно, возможно такое искажение, которое делает неоднозначным интерпретацию полученного сообщения, например, «*гпоорраа*», «*ггоорреа*» или «*кгоорраа*». Однако цель такого способа кодирования состоит не в исправлении ошибки, а в фиксации факта искажения и повторной передаче части сообщения в этом случае. Недостаток данного способа обеспечения надежности состоит в том, что избыточность кода оказывается очень большой – очевидно, что $L = 2$.

Поскольку ошибка должна быть только обнаружена, можно предложить другой способ кодирования. Пусть имеется цепочка информационных бит длиной k_0. Добавим к ним еще один бит, значение которого определяется тем, что новая кодовая цепочка из k_0+1 бита должна содержать чётное количество единиц – по этой причине такой контрольный бит называется ***битом чётности***. Например, для информационного байта 01010100 бит чётности будет иметь значение 1, а для байта 11011011 бит чётности равен 0. В случае одиночной ошибки передачи число 1 перестает быть чётным, что и служит свидетельством сбоя. Например, если получена цепочка 110110111 (контрольный бит выделен подчеркиванием), ясно, что передача произведена с ошибкой, поскольку общее количество единиц равно 7, т.е. нечётно. В каком бите содержится ошибка при таком способе кодирования установить нельзя. Избыточность кода в данном случае, очевидно, равна:

$$L = \frac{k_0 + 1}{k_0}.$$

На первый взгляд кажется, что путем увеличения k_0 можно сколь угодно приближать избыточность к ее минимальному значению (L_{min} = 1). Однако с ростом k_0, во-первых, растет вероятность парной ошибки, которая контрольным битом не отслеживается; во-вторых, при обнаружении ошибки потребуется заново передавать много информации. Поэтому обычно k_0 = 8 или 16 и, следовательно, L=1,125 (1,0625).

Коды, исправляющие одиночную ошибку

В 1948 г. Р.Хеммингом был предложен способ кодирования информации, который позволяет не только обнаружить наличие ошибки, но и локализовать её (т.е. определить, в каком бите она находится) [8]. Подобные коды стали называться ***кодами Хемминга***.

Основная идея состоит в добавлении к информационным битам не одного, а нескольких битов чётности, каждый из которых контролирует определенные информационные биты. Если пронумеровать все биты передаваемого кода Хемминга, начиная с 1 слева направо, то контрольными (проверочными) являются биты, номера которых равны степеням числа 2, а все остальные являются информационными. Например, для 12-битного кода Хемминга контрольными окажутся биты с номерами 1, 2, 4 и 8 (см. рисунок 16).

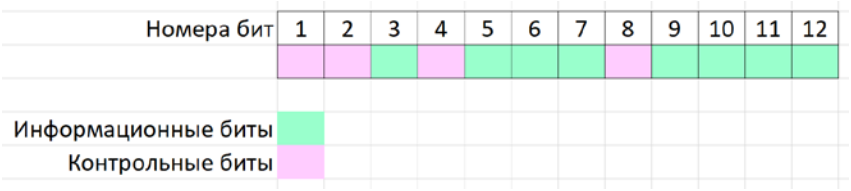


Рис. 16. Структура кода Хемминга

Номера контролируемых бит для каждого проверочного приведены в таблице 3. Состояние проверочного бита устанавливается таким образом, чтобы суммарное количество единиц в контролируемых им битах было бы чётным.

Таблица 3. Контролируемые биты

Пров. биты	**Контролируемые биты**
1	1 3 5 7 9 11 13 15 17 19 21 …
2	2 3 6 7 10 11 14 15 18 19 22 …
4	4 5 6 7 12 13 14 15 20 21 22 …
8	8 9 10 11 12 13 14 15 24 25 26 …
16	16 17 18 19 20 21 22 23 24 25 26 …
32	32 33 34 35 36 37 38 39 40 41 42 …

Для того чтобы легко запомнить номера контролируемых бит, существует такая закономерность:

1-ый контрольный бит контролирует 1 через 1 бит, начиная с самого себя (т.е. 1, 3, 5, 7, 9, 11 и т.д);

2-й контрольный бит контролирует 2 через 2 бита, начиная с самого себя (т.е. 2, 3, 6, 7, 10, 11 и т.д.) и т.д.

Пример. Составить код Хемминга для информационного байта 11010010.

Контрольными являются биты с номерами 1, 2, 4 и 8. В остальные свободные биты заносим информационный байт (см. рисунок 17).

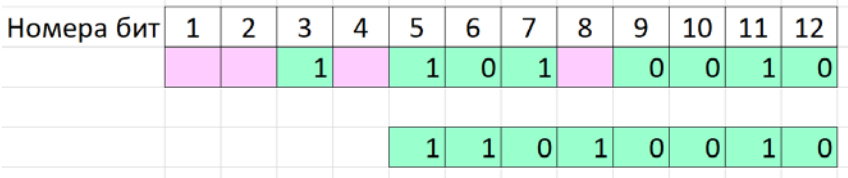


Рис. 17. Заполняем информационные биты

На рисунке 18 отмечено иксами (X), какие биты нужно использовать для вычисления первого контрольного бита (с номером 1).

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
			1		1	0	1		0	0	1	0
	X		X		X		X		X		X	
	1+1+1+0+1=4											

Рис. 18. Вычисление первого контрольного бита

Складываем контролируемые биты, и, если сумма получилась чётная, присваиваем контрольному биту значение 0, если нечётная, то значение 1.

Первому контрольному биту присваиваем значение 0.

Далее, на рисунке 19 отмечено, какие биты нужно использовать для вычисления контрольного бита с номером 2.

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0		1		1	0	1		0	0	1	0
		X	X			X	X			X	X	
	1+0+1+0+1=3											

Рис. 19. Вычисление контрольного бита с номером 2

Сумма контролируемых бит нечётная, поэтому присваиваем биту с номером 2 значение 1.

Переходим к вычислению контрольного бита с номером 4. На рисунке 20 отмечены контролируемые им биты.

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1		1	0	1		0	0	1	0
				X	X	X	X					X
	1+0+1+0=2											

Рис. 20. Вычисление контрольного бита с номером 4

Сумма контролируемых бит чётная, поэтому присваиваем биту с номером 4 значение 0.

И наконец, вычислим контрольный бит с индексом 8. На рисунке 21 отмечены контролируемые им биты.

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	1	0	1	1	0	0	1	0
								X	X	X	X	X
	0+0+1+0=1											

Рис. 21. Вычисление контрольного бита с номером 8

Сумма контролируемых бит нечётная, поэтому присваиваем биту с номером 8 значение 1.

Таким образом, код Хемминга для информационного байта 11010010 – это 011010110010.

Допустим, в процессе передачи по реальному каналу связи произошла ошибка и вместо указанной выше последовательности пришла следующая: 011000110010 (в 5-м бите 1 заменилась на 0, см. рисунок 22).

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	0	0	1	1	0	0	1	0

Рис. 22. Код Хемминга с ошибкой в одном бите

Для того чтобы проверить код Хемминга на наличие или отсутствие ошибки, необходимо проверить все контрольные биты на правильность. Сумма всех контролируемых бит каждого конкретного контрольного бита (включая его самого) должна быть чётной! Если сумма нечётна, значит данный контрольный бит сигнализирует об ошибке среди бит, которые он контролирует.

В нашем примере бит 1 указывает на наличие ошибки в каком-либо бите с нечётным номером (1, 3, 5, 7, 9 или 11), поскольку сумма контролируемых им бит нечётная (см. рисунок 23).

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	0	0	1	1	0	0	1	0
	X		X		X		X		X		X	
	0+1+0+1+0+1=3				Ошибка!							

Рис. 23. Контрольный бит с номером 1 сигнализирует об ошибке

Бит 2 свидетельствует о том, что в битах 2, 3, 6, 7, 10 и 11 ошибки нет, т.е. ошибка в 1, 5 или 9 битах (см. рисунок 24).

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	0	0	1	1	0	0	1	0
		X	X			X	X			X	X	
	1+1+0+1+0+1=4				Ок							

Рис. 24. Контрольный бит с номером 2 верный

Бит 4 указывает, что ошибка находится в одном из бит с номерами 4, 5, 6, 7, 12 (см. рисунок 25). В то же время нам известно, что ошибка в одном из бит с номерами 1, 5, 9. Следовательно, ошибка в 5 бите!

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	0	0	1	1	0	0	1	0
				X	X	X	X					X
	0+0+0+1+0=1				Ошибка!							

Рис. 25. Контрольный бит с номером 4 сигнализирует об ошибке

Таким образом, однозначно устанавливается, что ошибочным является 5-й бит – можно исправить его значение на противоположное и, тем самым, восстановить правильную последовательность.

Контрольный бит с номером 8 дополнительно сообщает, что ошибки в 9 бите нет (см. рисунок 26).

Номера бит	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	1	0	0	0	1	1	0	0	1	0
								X	X	X	X	X
	1+0+0+1+0=2				Ок							

Рис. 26. Контрольный бит с номером 8 верный

Более детальное рассмотрение кодов Хемминга позволяет сформулировать простой ***алгоритм проверки и исправления*** передаваемой последовательности бит:

1) произвести проверку всех битов чётности;

2) если все биты чётности верны, то перейти к п.5;

3) вычислить сумму номеров всех неправильных битов чётности;

4) инвертировать содержимое бита, номер которого равен сумме, найденной в п.3;

5) исключить биты чётности, передать правильный информационный код.

На рассмотренном выше примере легко убедиться в справедливости данного алгоритма. В примере 1 и 4 контрольные биты сигнализировали об ошибке, 1+4=5. В 5 бите присутствовала ошибка!

Избыточность кодов Хемминга для различных длин передаваемых последовательностей приведена в таблице 4.

Таблица 4. Зависимость избыточности кода от числа контрольных бит

Число информационных бит	Число контрольных бит	Избыточность
8	4	1,50
16	5	1,31
32	6	1,06

Из сопоставления видно, что выгоднее передавать и хранить более длинные последовательности бит.

Безусловно, данный способ кодирования требует увеличения объема памяти компьютера приблизительно на одну треть при 16-битной длине машинного слова, однако он позволяет автоматически исправлять одиночные ошибки. Поэтому, оценивая время наработки на отказ, следует исходить из вероятности появления парной ошибки в одной последовательности (т.е. сбои должны произойти в двух битах одновременно). Расчёты показывают, что для указанного ранее количества ячеек в памяти объемом 1 Мбайт среднее время появления ошибки составляет более 80 лет, что, безусловно, можно считать вполне приемлемым с практической точки зрения.

4.2 Лабораторная работа № 4

1. Вычислить значение бита паритета к заданному информационному байту.

2. Построить код Хемминга для информационного байта.

3. Укажите номер бита с ошибкой в коде Хемминга (использовать логические рассуждения и алгоритм Хемминга).

Варианты заданий

№ вар.	Информационный байт	Код Хемминга с ошибкой
1	01100011	0 1 1 0 1 0 1 0 0 0 1 1
2	11100010	1 0 0 1 0 0 1 0 1 0 0 0
3	00101011	0 0 1 1 1 1 1 1 1 0 1 1
4	10101010	1 1 0 1 1 1 1 1 0 0 1 1
5	10000011	1 1 1 0 1 1 0 1 0 0 0 1
6	01000010	0 1 1 1 0 1 1 1 0 1 1 1
7	11001110	0 0 0 1 0 1 1 0 0 1 1 1
8	00101101	0 1 1 0 1 0 1 1 1 1 1 0
9	01001111	1 0 0 1 1 0 1 0 1 1 1 1
10	11101110	1 1 0 1 0 0 1 1 1 0 1 0
11	01101111	1 1 1 0 0 1 1 1 0 1 1 1
12	10100010	1 0 0 0 0 1 1 0 0 0 1 0
13	00100010	0 1 1 1 0 1 1 1 1 0 0 0
14	11001011	1 0 1 0 0 1 1 1 0 0 1 1
15	11001000	1 1 0 1 0 1 0 1 1 0 1 1
16	00010001	0 0 1 1 1 1 1 0 1 0 1 0
17	01001111	1 0 0 1 1 0 1 1 0 1 1 1
18	11000110	0 1 0 1 0 0 1 1 0 1 1 0
19	01100111	1 0 0 1 0 0 1 0 0 1 1 0
20	10100111	0 1 0 1 0 1 1 1 1 0 0 1

21	00101110	0010111101100
22	11011111	0010011101000
23	01010100	1101001110011
24	11010001	0011111110011
25	00000011	0100111000011
26	11000011	0110000110011
27	01001010	1101001010010
28	10101011	1100001110001
29	00110110	1001111100100
30	11110101	0011001110010

СПИСОК ИСТОЧНИКОВ

1. Лидовский, В. В. Теория информации: учебное пособие. – Москва: Компания Спутник+, 2004. – 111 с.

2. Фурсов, В.А. Практикум по теории информации: учебное пособие / В.А. Фурсов, Н.Е. Козин. – Самара: Изд-во Самар, гос. аэрокосм, ун-та, 2007. – 80 с.

3. Гошин, Е.В. Практикум по теории информации и кодирования: учебное пособие. – Самара: Изд-во Самарского ун-та, 2018. – 80 с.

4. Чикрин, Д.Е. Теория информации и кодирования: курс лекций. – Казань: Казанский университет, 2013. – 116 с. – URL: http://biblioclub.ru/index.php?page=book&id=63307 (дата обращения 18.10.2023). – Режим доступа: по подписке.

5. Шеннон, К. Работы по теории информации и кибернетике / К. Шеннон; под ред. Р.Л. Добрушина, О.Б. Лупанова; предисл. А.Н. Колмогорова. – Москва: Издательство иностранной литературы, 1963. – 830 с. – URL: http://biblioclub.ru/index.php?page=book&id=450093 (дата обращения 18.10.2023). – Режим доступа: по подписке.

6. Балюкевич, Э.Л. Основы теории информации: учебно-практическое пособие / Э.Л. Балюкевич. – Москва: Евразийский открытый институт, 2008. 216 с. – URL: http://biblioclub.ru/index.php?page=book&id=90955 (дата обращения 18.10.2023).

7. Гультяева, Т.А. Основы теории информации и криптографии: конспект лекций / Т.А. Гультяева; Министерство образования и науки Российской Федерации, Новосибирский государственный технический университет. – Новосибирск: НГТУ, 2010. – 88 с. –URL: http://biblioclub.ru/index.php?page=book&id=228963 (дата обращения 18.10.2023). – Режим доступа: по подписке.

8. Чечёта, С.И. Введение в дискретную теорию информации и кодирования: учебное пособие / С.И. Чечёта. – Москва: МЦНМО, 2011. – 224 с.