Министерство информационных технологий и связи
Российской Федерации

Сибирский государственный университет
телекоммуникаций и информатики

Е. В. Курапова
Е. П. Мачикина

ОСНОВНЫЕ МЕТОДЫ КОДИРОВАНИЯ ДАННЫХ

Методические указания

Новосибирск 2010

УДК 681.3.06

ктн Е. В. Курапова, кф-мн Е. П. Мачикина
Основные методы кодирования данных: Методические указания. / Сиб. гос. ун-т телекоммуникаций и информатики. – Новосибирск, 2010. – 54 с.

Методические указания предназначены для студентов технических специальностей, изучающих дисциплину «Структуры и алгоритмы обработки данных». Пособие содержит необходимые теоретические сведения об основных методах кодирования информации и варианты заданий для самостоятельного выполнения.

Рисунков —13, таблиц — 8. Список лит. –6 назв.

Кафедра прикладной математики и кибернетики.

Рецензент:

Утверждено редакционно-издательским советом СибГУТИ
в качестве методических указаний.

ОГЛАВЛЕНИЕ

1. ВВЕДЕНИЕ

Изучение дисциплины «Структуры и алгоритмы обработки данных» является одним из основных моментов в процессе подготовки специалистов по разработке программного обеспечения для компьютерных систем. Это связано с тем, что первичная задача программиста заключается в применении решения о форме представления данных и выборе алгоритмов, применяемых к этим данным. И лишь затем выбранная структура программы и данных реализуется на конкретном языке программирования. В связи с этим знание классических методов и приемов обработки данных позволяет избежать ошибок, которые могут возникать при чисто интуитивной разработке программ.

Данные методические указания содержат необходимый теоретический материал по разделу курса «Структуры и алгоритмы обработки данных», посвященного различным методам кодирования информации. Все рассмотренные методы проиллюстрированы наглядными примерами. Для каждого метода приведен конкретный алгоритм, позволяющий легко программировать данный метод. Также методические указания содержат задания для лабораторных работ по каждой теме, выполнив которые можно окончательно уяснить все особенности изучаемых методов.

2. НЕОБХОДИМЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Теория кодирования и теория информации возникли в начале XX века. Начало развитию этих теорий как научных дисциплин положило появление в 1948 г. статей К. Шеннона, которые заложили фундамент для дальнейших исследований в этой области.

Кодирование – способ представления информации в удобном для хранения и передачи виде. В связи с развитием информационных технологий кодирование является центральным вопросом при решении самых разных задач программирования, таких как:

1. представление данных произвольной структуры (числа, текст, графика) в памяти компьютера;
2. обеспечение помехоустойчивости при передаче данных по каналам связи;
3. сжатие информации в базах данных.

Основной моделью, которую изучает теория информации, является *модель системы передачи сигналов:*

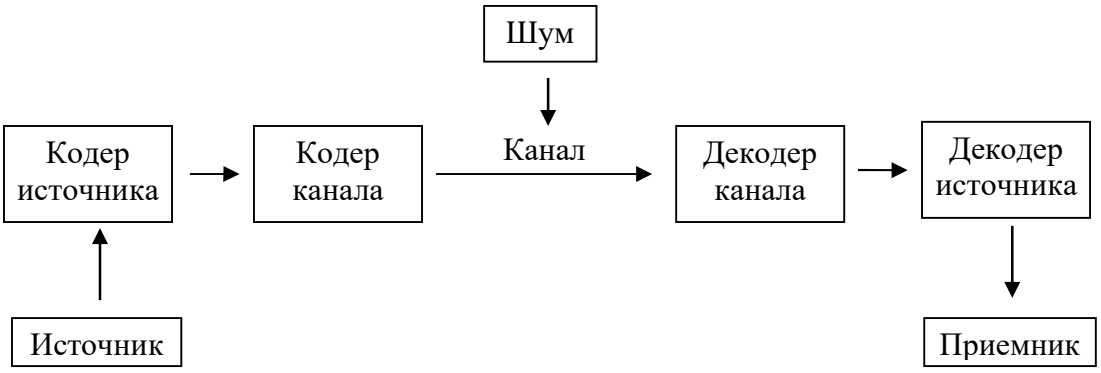


Рисунок 1 Модель системы передачи сигналов

Начальным звеном в приведенной выше модели является *источник информации.* Здесь рассматриваются *дискретные источники без памяти*, в которых *выходом* является последовательность символов некоторого фиксированного алфавита. Множество всех различных символов, порождаемых некоторым источником, называется *алфавитом источника*, а количество символов в этом множестве – *размером алфавита источника*. Например, можно считать, что текст на русском языке порождается источником с алфавитом из 33 русских букв, пробела и знаков препинания.

Кодирование дискретного источника заключается в сопоставлении символов алфавита *A* источника символам некоторого другого алфавита *B*. Причем обычно символу исходного алфавита *A* ставится в соответствие не один, а группа символов алфавита *B*, которая называется *кодовым словом*. *Кодовый алфавит* – множество различных символов, используемых для

записи кодовых слов. *Кодом* называется совокупность всех кодовых слов, применяемых для представления порождаемых источником символов.

Пример. Азбука Морзе является общеизвестным кодом из символов телеграфного алфавита, в котором буквам русского языка соответствуют кодовые слова (последовательности) из «точек» и «тире».

Далее будем рассматривать *двоичное кодирование*, т.е. размер кодового алфавита равен 2. Конечную последовательность битов (0 или 1) назовем *кодовым словом*, а количество битов в этой последовательности – *длиной кодового слова*.

Пример. Код ASCII (американский стандартный код для обмена информацией) каждому символу ставит в однозначное соответствие кодовое слово длиной 8 бит.

Дадим строгое определение кодирования. Пусть даны алфавит источника $A = \{a_1, a_2, ..., a_n\}$, кодовый алфавит $B = \{b_1, b_2, ..., b_m\}$. Обозначим A^* (B^*) множество всевозможных последовательностей в алфавите A (B). Множество всех сообщений в алфавите A обозначим S. Тогда отображение $F : S \to B^*$, которое преобразует множество сообщений в кодовые слова в алфавите B, называется *кодированием*. Если $\alpha \in S$, то $F(\alpha) = \beta \in B^*$ – кодовое слово. Обратное отображение F^{-1} (если оно существует) называется *декодированием*.

Задача кодирования сообщения ставится следующим образом. Требуется при заданных алфавитах A и B и множестве сообщений S найти такое кодирование F, которое обладает определенными свойствами и оптимально в некотором смысле. Свойства, которые требуются от кодирования, могут быть различными. Приведем некоторые из них:

1. существование декодирования;
2. помехоустойчивость или исправление ошибок при кодировании: декодирование обладает свойством $F^{-1}(\beta) = F^{-1}(\beta')$, $\beta \sim \beta'$ (эквивалентно β' с ошибкой);
3. обладает заданной трудоемкостью (время, объем памяти).

Известны *два класса методов кодирования* дискретного источника информации: равномерное и неравномерное кодирование. Под *равномерным кодированием* понимается использование кодов со словами постоянной длины. Для того чтобы декодирование равномерного кода было возможным, разным символам алфавита источника должны соответствовать разные кодовые слова. При этом длина кодового слова должна быть не меньше $\lceil \log_n m \rceil$ символов, где m – размер исходного алфавита, n – размер кодового алфавита.

Пример. Для кодирования источника, порождающего 26 букв латинского алфавита, равномерным двоичным кодом требуется построить кодовые слова длиной не меньше $\lceil \log_2 26 \rceil$ =5 бит.

При *неравномерном кодировании источника* используются кодовые слова разной длины. Причем кодовые слова обычно строятся так, что часто встречающиеся символы кодируются более короткими кодовыми словами, а редкие символы – более длинными (за счет этого и достигается «сжатие» данных).

Под *сжатием данных* понимается *компактное представление данных*, достигаемое за счет избыточности информации, содержащейся в сообщениях. Большое значение для практического использования имеет *неискажающее сжатие*, позволяющее полностью восстановить исходное сообщение. При *неискажающем сжатии* происходит кодирование сообщения перед началом передачи или хранения, а после окончания процесса сообщение однозначно декодируется (это соответствует модели канала без шума (помех)).

Методы сжатия данных можно разделить на две группы: статические методы и адаптивные методы. *Статические* методы сжатия данных предназначены для кодирования конкретных источников информации с известной статистической структурой, порождающих определенное множество сообщений. Эти методы базируются на знании статистической структуры исходных данных. К наиболее известным статическим методам сжатия относятся коды Хаффмана, Шеннона, Фано, Гилберта-Мура, арифметический код и другие методы, которые используют известные сведения о вероятностях порождения источником различных символов или их сочетаний.

Если статистика источника информации неизвестна или изменяется с течением времени, то для кодирования сообщений такого источника применяются *адаптивные методы сжатия*. В *адаптивных* методах при кодировании очередного символа текста используются сведения о ранее закодированной части сообщения для оценки вероятности появления очередного символа. В процессе кодирования адаптивные методы «настраиваются» на статистическую структуру кодируемых сообщений, т.е. коды символов меняются в зависимости от накопленной статистики данных. Это позволяет адаптивным методам эффективно и быстро кодировать сообщение за один просмотр.

Существует множество различных адаптивных методов сжатия данных. Наиболее известные из них – адаптивный код Хаффмана, код «стопка книг», интервальный и частотный коды, а также методы из класса Лемпела-Зива.

3. КОДИРОВАНИЕ ЦЕЛЫХ ЧИСЕЛ

Рассмотрим семейство методов кодирования, не учитывающих вероятности появления символов источника. Поскольку все символы алфавита источника можно пронумеровать, то в будем считать, что алфавит источника состоит из целых чисел. Каждому целому числу из определенного диапазона ставится в соответствие свое кодовое слово, поэтому эту группу методов также называют представлением целых чисел (representation of integers).

Основная идея кодирования состоит в том, чтобы отдельно кодировать порядок значения элемента x_i («экспоненту» E_i) и отдельно – значащие цифры значения x_i («мантиссу» $M_{i\,i}$). Значащие цифры мантиссы начинаются со старшей ненулевой цифры, а порядок числа определяется позицией старшей ненулевой цифры в двоичной записи числа. Как и при десятичной записи, порядок равен числу цифр в записи числа без предшествующих незначащих нулей.

Пример. Порядок двоичного числа 000001101 равен 4, а мантисса – 1101.

В этой главе будут рассмотрены две группы методов кодирования целых чисел. Условно их можно обозначить так:

- Fixed + Variable (фиксированная длина экспоненты + переменная длина мантиссы)
- Variable + Variable (переменная длина экспоненты + переменная длина мантиссы)

3.1 Коды класса Fixed + Variable

В кодах класса Fixed + Variable под запись значения порядка числа отводится фиксированное количество бит, а значение порядка числа определяет, сколько бит потребуется под запись мантиссы. Для кодирования целого числа необходимо произвести с числом две операции: определение порядка числа и выделение бит мантиссы (можно хранить в памяти готовую таблицу кодовых слов). Рассмотрим процесс построения кода данного класса на примере.

Пример. Пусть $R = 15$ – количество бит исходного числа. Отведем $E = 4$ бита под экспоненту (порядок), т.к. $R \leq 2^4$. При записи мантиссы можно сэкономить 1 бит: не писать первую единицу, т.к. это всегда будет только единица. Таким образом, количество бит мантиссы меньше на один бит, чем количество бит для порядка.

Таблица 1 Код класса *Fixed + Variable*

число	двоичное представление	кодовое слово	длина кодового слова
0	000000000000000	0000	4
1	000000000000001	0001	4
2	000000000000010	0010 0	5
3	000000000000011	0010 1	5
4	000000000000100	0011 00	6
5	000000000000101	0011 01	6
6	000000000000110	0011 10	6
7	000000000000111	0011 11	6
8	000000000001000	0100 000	7
9	000000000001001	0100 001	7
10	000000000001010	0100 010	7
…	…	…	..
15	000000000001111	0100 111	7
16	000000000010000	0101 0000	8
17	000000000010001	0101 0001	8
…	…	…	..

3.2 Коды класса Variable + Variable

В качестве кода числа берется двоичная последовательность, построенная следующим образом: несколько нулей (количество нулей равно значению порядка числа), затем единица как признак окончания экспоненты переменной длины, затем мантисса переменной длины (как в кодах Fixed + Variable). Рассмотрим пример построения кода этого класса.

Таблица 2 Код класса *Variable + Variable*

число	двоичное представление	кодовое слово	длина кодового слова
0	00000000000	1	1
1	00000000001	0 1	2
2	00000000010	00 1 0	4
3	00000000011	00 1 1	4
4	00000000100	000 1 00	6
5	00000000101	000 1 01	6
6	00000000110	000 1 10	6
7	00000000111	000 1 11	6
8	00000001000	0000 1 000	8
9	00000001001	0000 1 001	8
10	00000001010	0000 1 010	8
…	…	…	

Если в рассмотренном выше коде исключить кодовое слово для нуля, то можно уменьшить длины кодовых слов на 1 бит, убрав первый нуль. Таким образом строится *гамма-код Элиаса (γ-код Элиаса)*.

Таблица 3 *Гамма-код Элиаса*

число	кодовое слово	длина кодового слова
1	1	1
2	01 0	3
3	01 1	3
4	00 1 00	5
5	00 1 01	5
6	00 1 10	5
7	00 1 11	5
8	000 1 000	7
9	000 1 001	7
10	000 1 010	7
…	…	

Другим примером кода класса Variable + Variable является *омега-код Элиаса (ω-код Элиаса).* В нем первое значение (кодовое слово для единицы) задается отдельно. Другие кодовые слова состоят из последовательности групп длиной $L_1, L_2, ..., L_m$, начинающихся с единицы. Конец всей последовательности задается нулевым битом. Длина первой группы составляет 2 бита, длина каждой следующей группы равна двоичному значению битов предыдущей группы плюс 1. Значение битов последней группы является итоговым значением всей последовательности групп, т.е. первые $m-1$ групп служат лишь для указания длины последней группы.

Таблица 4 *Омега-код Элиаса*

число	кодовое слово	длина кодового слова
1	0	1
2	10 0	3
3	11 0	3
4	10 100 0	6
5	10 101 0	6
6	10 110 0	6
7	10 111 0	6
8	11 1000 0	7
9	11 1001 0	7
..	…	..
15	11 1111 0	7
16	10 100 10000 0	11
17	10 100 10001 0	11
..	…	..

31	10 100 11111 0	11
32	10 101 100000 0	12

При кодировании формируется сначала последняя группа, затем предпоследняя и т.д., пока процесс не будет завершен. При декодировании, наоборот, сначала считывается первая группа, по значению ее битов определяется длина следующей группы, или итоговое значение кода, если следующая группа – 0.

Рассмотренные типы кодов могут быть эффективны в следующих случаях

1. Вероятности чисел убывают с ростом значений элементов и их распределение близко к такому: $P(x) \geq P(x+1)$, при любом x, т.е. маленькие числа встречаются чаще, чем большие.
2. Диапазон значений входных элементов не ограничен или неизвестен. Например, при кодировании 32-битовых чисел реально большинство чисел маленькие, но могут быть и большие.
3. При использовании в составе других схем кодирования, например, кодировании длин серий.

3.3 Кодирование длин серий

Метод кодирования информации, известный как метод кодирования длин серий и предложенный П. Элиасом, при построении использует коды целых чисел. Входной поток для кодирования рассматривается как последовательность из нулей и единиц. Идея кодирования заключается в том, чтобы кодировать последовательности одинаковых элементов (например, нулей) как целые числа, указывающие количество элементов в этой последовательности. Последовательность одинаковых элементов называется *серией*, количество элементов в ней – *длиной серии*.

Пример. Входную последовательность (общая длина 31бит) можно разбить на серии, а затем закодировать их длины.

000000 1 00000 1 0000000 1 1 00000000 1

Используем, например, γ-код Элиаса. Поскольку в коде нет кодового слова для нуля, то будем кодировать длину серии +1, т.е. последовательность 7 6 8 1 9:

$$7\ 6\ 8\ 1\ 9 \Rightarrow 00111\ 00110\ 0001000\ 1\ 0001001$$

Длина полученной кодовой последовательности равна 25 бит.

Метод длин серий актуален для кодирования данных, в которых есть длинные последовательности одинаковых бит. В нашем примере, если $P(0) >> P(1)$.

4. НЕКОТОРЫЕ ТЕОРЕМЫ ПОБУКВЕННОГО КОДИРОВАНИЯ

В этом параграфе приведены некоторые теоремы о свойствах побуквенного кодирования.

Пусть даны алфавит источника $A=\{a_1,a_2,...,a_n\}$, кодовый алфавит $B=\{b_1,b_2,...,b_m\}$. Обозначим A^* (B^*) множество всевозможных последовательностей в алфавите A (B). Множество всех сообщений в алфавите A обозначим S. Кодирование $F:S\to B^*$ может сопоставлять код всему сообщению из множества S как единому целому или строить код сообщения из кодов его частей (побуквенное кодирование).

Пример 1 $A=\{a_1,a_2,a_3\}$, $B=\{0,1\}$ Побуквенное кодирование символов источника $a_1\to 1001$ $a_2\to 0$ $a_3\to 010$
позволяет следующим образом закодировать сообщение

$$a_2a_1a_2a_3\to 010010010$$

Пример 2 Азбука Морзе. Входной алфавит – английский. Наиболее часто встречающиеся буквы кодируются более короткими словами:

$$\text{A}\to 01,\ \text{B}\to 1000,\ \text{C}\to 1010,\ \text{D}\to 100,\ \text{E}\to 0,\ \ldots$$

Побуквенное кодирование задается *таблицей кодовых слов:* $\sigma=\langle a_1\to\beta_1,...,a_n\to\beta_n\rangle$, $a_i\in A$, $\beta_i\in B^*$. Множество кодовых слов $V=\{\beta_i\}$ называется *множеством элементарных кодов.* Используя побуквенное кодирование, можно закодировать любое сообщение $\alpha=\alpha_{i1}...\alpha_{ik}\in S$ следующим образом $F(\alpha)=F(\alpha_{i1})...F(\alpha_{ik})=\beta_{i1}...\beta_{ik}$, т.е. общий код сообщения складывается из элементарных кодов символов входного алфавита.

Количество букв в слове $\alpha=\alpha_1...\alpha_k$ называется *длиной слова*. (Обозначение $|\alpha|=k$) Пустое слово, т.е. слово, не содержащее ни одного символа обозначается Λ. Если $\alpha=\alpha_1\alpha_2$, то α_1 – *начало (префикс)* слова α, α_2 – *окончание (постфикс)* слова α.

Побуквенный код называется *разделимым* (или *однозначно декодируемым*), если любое сообщение из символов алфавита источника, закодированное этим кодом, может быть однозначно декодировано, т.е. если $\beta_{i1}\ ...\beta_{ik}=\beta_{j1}...\beta_{jt}$, то $k=t$ и при любых $s=1,\ldots,k$ $i_s=j_s$. При разделимом кодировании любое кодовое слово единственным образом разлагается на элементарные коды.

Пример. 3 Код из примера 1 не является разделимым, поскольку кодовое слово 010010 может быть декодируемо двумя способами: a_3a_3 или $a_2a_1a_2$.

Побуквенный код называется *префиксным*, если в его множестве кодовых слов ни одно слово не является началом другого, т.е. элементарный код одной буквы не является префиксом элементарного кода другой буквы.

Пример 4. Код из примера 1 не является префиксным, поскольку элементарный код буквы a_2 является префиксом элементарного кода буквы a_3.

Утверждение. *Префиксный код является разделимым.*

Доказательство (от противного). Пусть префиксный код не является разделимым. Тогда существует такая кодовая последовательность β, что она представлена различными способами из элементарных кодов: $\beta = \beta_{i1}\beta_{i2}...\beta_{ik} = \beta_{j1}\beta_{j2}...\beta_{jt}$ (побитовое представление одинаковое) и существует L такое, что при любом $s < L$ следует ($\beta_{is}=\beta_{js}$) и ($\beta_{it}\neq\beta_{jt}$), т.е. начало каждого из этих представлений имеет одинаковую последовательность элементарных кодов. Уберем эту часть. Тогда $\beta_{iL}...\beta_{ik} = \beta_{jL}...\beta_{jt}$, т.е. последовательности элементарных кодов разные и существует β', что $\beta_{iL}=\beta_{jL}\beta'$ или $\beta_{jL}=\beta_{iL}\beta'$, т.е. β_{iL} – начало β_{jL}, или наоборот. Получили противоречие с префиксностью кода.

Заметим, что разделимый код может быть не префиксным.

Пример 5. Разделимый, но не префиксный код: $A=\{a,b\}$, $B=\{0,1\}$,

$$\sigma = \langle a \to 0, b \to 01 \rangle$$

Приведем основные теоремы побуквенного кодирования.

Теорема *(Крафт). Для того, чтобы существовал побуквенный двоичный префиксный код с длинами кодовых слов $L_1,...,L_n$ необходимо и достаточно, чтобы*

$$\sum_{i=1}^{n} 2^{-L_i} \le 1.$$

Доказательство. Докажем необходимость. Пусть существует префиксный код с длинами $L_1,...,L_n$. Рассмотрим полное двоичное дерево. Каждая вершина закодирована последовательностью нулей и единиц (как показано на рисунке).

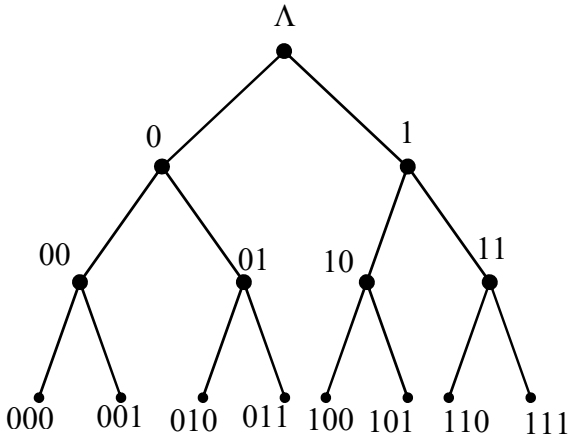


Рисунок 2 Полное двоичное дерево с помеченными вершинами

В этом дереве выделим вершины, соответствующие кодовым словам. Тогда любые два поддерева, соответствующие кодовым вершинам дерева, не

пересекаются, т.к. код префиксный. У i-того поддерева на r-том уровне – 2^{r-Li} вершин. Всего вершин в поддереве 2^r. Тогда $\sum_{i=1}^{n} 2^{r-L_i} \le 2^r$, $\sum_{i=1}^{n} 2^r 2^{-L_i} \le 2^r$, $\sum_{i=1}^{n} 2^{-L_i} \le 1$.

Докажем достаточность утверждения. Пусть существует набор длин кодовых слов такой, что $\sum_{i=1}^{n} 2^{-L_i} \le 1$. Рассмотрим полное двоичное дерево с помеченными вершинами. Пусть длины кодовых слов упорядочены по возрастанию $L_1 \le L_2 \le \ldots \le L_n$. Выберем в двоичном дереве вершину V_1 на уровне L_1. Уберем поддерево с корнем в вершине V_1. В оставшемся дереве возьмем вершину V_2 на уровне L_2 и удалим поддерево с корнем в этой вершине и т.д. Последовательности, соответствующие вершинам $V_1, V_2, \ldots, V_n$ образуют префиксный код. Теорема доказана.

Пример 6. Построить префиксный код с длинами $L_1=1$, $L_2=2$, $L_3=2$ для алфавита $A=\{a_1,a_2,a_3\}$. Проверим неравенство Крафта для набора длин

$$\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^2}=1.$$

Неравенство выполняется и, следовательно, префиксный код с таким набором длин кодовых слов существует. Рассмотрим полное двоичное дерево с 2^3 помеченными вершинами и выберем вершины дерева, как описано выше. Тогда элементарные коды могут быть такими: $a_1 \to 0$, $a_2 \to 10$, $a_3 \to 11$.

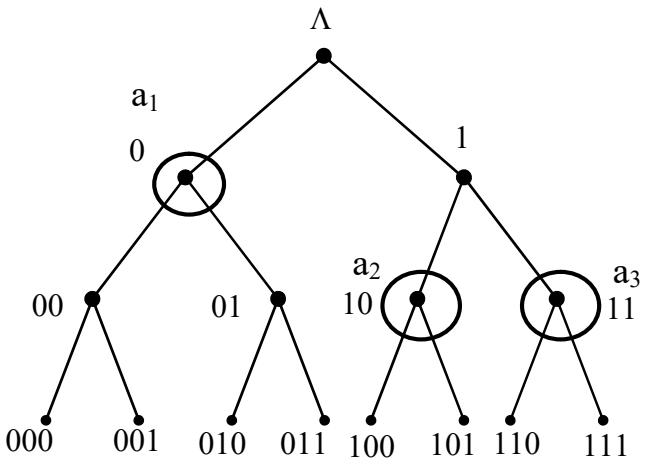


Рисунок 3 Построение префиксного кода с заданными длинами

Процесс декодирования выглядит следующим образом. Просматриваем полученное сообщение, двигаясь по дереву. Если попадем в кодовую вершину, то выдаем соответствующую букву и возвращаемся в корень дерева и т.д.

Теорема *(МакМиллан). Для того чтобы существовал побуквенный двоичный разделимый код с длинами кодовых слов $L_1,\ldots,L_n$, необходимо и достаточно, чтобы* $\sum_{i=1}^{n} 2^{-L_i} \le 1.$

Доказательство. Покажем достаточность. По теореме Крафта существует префиксный код с длинами $L_1,\ldots,L_n$, и он является разделимым.
Докажем необходимость утверждения. Рассмотрим тождество

$$(x_1 + x_2 + \ldots + x_n)^m = \sum_{1\le i_1<\ldots<i_n\le n} x_{i_1} \cdot \ldots \cdot x_{i_m}$$

Положим $x_i = 2^{-l_i}$. Тогда тождество можно переписать следующим образом

$$\left(\sum_{i=1}^{n}\left(2^{-l_i}\right)\right)^m = \sum_{1\le i_1\le\ldots\le i_m\le n} 2^{-(l_{i_1}+\ldots+l_{i_m})} = \sum_{j=1}^{ml_{\max}} \sum_{j=l_{i_1}+\ldots+l_{i_m}} 2^{-j} = \sum_{j=1}^{ml_{\max}} M_j \cdot 2^{-j},$$

где $l_{\max} = \max_i l_i$, M_j – число всевозможных представлений числа j в виде суммы $j = l_{i_1} + \ldots + l_{i_m}$. Сопоставим каждому представлению числа j в виде суммы последовательность нулей и единиц длины j по следующему правилу

$$j = l_{i_1} + \ldots + l_{i_m} \to b_{i_1} \ldots b_{i_m},$$

где b_s – элементарный код длины s. Тогда различным представлениям числа j будут соответствовать различные кодовые слова, поскольку код является разделимым. Таким образом, $M_j \le 2^j$ и $\left(\sum_{i=1}^{n} 2^{-l_i}\right)^m \le \sum_{j=1}^{ml_{\max}} 2^j \cdot 2^{-j} = m \cdot l_{\max}$.

Используя предельный переход получим $\sum_{i=1}^{n} 2^{-l_i} \le \sqrt[m]{ml_{\max}} \to 1$ при $m \to \infty$.

Теорема доказана.

Пример 7. Азбука Морзе – это схема алфавитного кодирования
A→01, B→1000, C→1010, D→100, E→0, F→0010, G→110, H→0000, I→00, J→0111, K→101, L→0100, M→11, N→10, O→111, P→0110, Q→1101, R→010, S→000, T→1, U→001, V→0001, W→011, X→1001, Y→1011, Z→1100.

Неравенство МакМиллана для азбуки Морзе не выполнено, поскольку

$$2 \cdot \frac{1}{2^1} + 4 \cdot \frac{1}{2^2} + 8 \cdot \frac{1}{2^3} + 12 \cdot \frac{1}{2^4} = 3\frac{3}{4} > 1$$

Следовательно, этот код не является разделимым. На самом деле в азбуке Морзе имеются дополнительные элементы – паузы между буквами (и словами), которые позволяют декодировать сообщение. Эти дополнительные элементы определены неформально, поэтому прием и передача сообщений (особенно с высокой скоростью) является некоторым искусством, а не простой технической процедурой.

5. ОПТИМАЛЬНОЕ ПОБУКВЕННОЕ КОДИРОВАНИЕ

5.1 Основные понятия

При кодировании сообщений считается, что символы сообщения порождаются некоторым *источником информации*. Источник считается заданным полностью, если дано вероятностное описание процесса появления сообщений на выходе источника. Это означает, что в любой момент времени определена вероятность порождения источником любой последовательности символов $P(x_1x_2x_3...x_L)$, $L \geq 1$. Такой источник называется *дискретным вероятностным источником.*

Если вероятностный источник с алфавитом $A=\{a_1, a_2, ..., a_n\}$ порождает символы алфавита независимо друг от друга, т.е. знание предшествующих символов не влияет на вероятность последующих, то такой источник называется *бернуллиевским*. Тогда для любой последовательности $x_1x_2...x_L$, $L \geq 1$, порождаемой источником, выполняется равенство:

$$P(x_1x_2...x_L) = P(x_1) \cdot P(x_2) \cdot ... \cdot P(x_L),$$

где $P(x)$ – вероятность появления символа x, $P(x_1x_2x_3...x_L)$ – вероятность появления последовательности $x_1x_2x_3...x_L$.

Для другого класса источников (марковских) существует статистическая взаимосвязь между порождаемыми символами. В дальнейшем мы будем рассматривать кодирование стационарных (с неизменным распределением вероятностей) бернуллиевских дискретных источников без памяти.

Пусть имеется дискретный вероятностный источник без памяти, порождающий символы алфавита $A=\{a_1,...,a_n\}$ с вероятностями $p_i = P(a_i)$, $\sum_{i=1}^{n} p_i = 1$. Основной характеристикой источника является *энтропия,* которая представляет собой среднее значение количества информации в сообщении источника и определяется выражением (для двоичного случая)

$$H(p_1,\ldots,p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i .$$

Энтропия характеризует меру неопределенности выбора для данного источника.

Пример. Если $A=\{a_1,a_2\}$, $p_1=0$, $p_2=1$, т.е. источник может породить только символ a_2, то неопределенности нет, энтропия $H(p_1,p_2)=0$.
Источник с равновероятными символами $A=\{a_1,a_2\}$, $p_1=1/2$, $p_2=1/2$, будет иметь максимальную энтропию $H(p_1,p_2)=1$.

Величина $H_L = \frac{1}{L} \cdot \sum_{x \in A^L} P(x) \log P(x)$ называется *энтропией на символ* последовательности длины L, где A^L – множество всех последовательностей длины L в алфавите A, $x=(x_1,x_2,...,x_L)$ – последовательность L букв

дискретного стационарного источника. Обозначим через H_∞ предел энтропии H_L при $L\to\infty$ $H_\infty = \lim\limits_{L\to\infty} H_L$. Эту величину называют *предельной энтропией источника***.** Показано, что для стационарного бернуллиевского источника

$$H_\infty = H(p_1,...,p_n).$$

Для практических применений важно, чтобы коды сообщений имели по возможности наименьшую длину. *Основной характеристикой неравномерного кода* является количество символов, затрачиваемых на кодирование одного сообщения. Пусть имеется разделимый побуквенный код для источника, порождающего символы алфавита A=$\{a_1,...,a_n\}$ с вероятностями $p_i = P(a_i)$, состоящий из n кодовых слов с длинами $L_1,...,L_n$ в алфавите {0,1}. *Средней длиной кодового слова* называется величина $L_{cp} = \sum\limits_{i=1}^{n} p_i L_i$, которая показывает среднее число кодовых букв на одну букву источника.

Пример. Пусть имеются два источника с одним и тем же алфавитом $A=\{a_1,a_2,a_3\}$ и разными вероятностными распределениями P_1={1/3, 1/3, 1/3}, P_2={1/4, 1/4, 1/2}, которые кодируются одним и тем же кодом

$$\sigma = \langle a_1 \to 10, a_2 \to 000, a_3 \to 01 \rangle.$$

Средняя длина кодового слова для разных источников будет различной

$$L_{cp}(P_1)=1/3\cdot 2 + 1/3\cdot 3 + 1/3\cdot 2 = 7/3 \approx 2.33$$
$$L_{cp}(P_2)=1/4\cdot 2 + 1/4\cdot 3 + 1/2\cdot 2 = 9/4 = 2.25$$

Побуквенный разделимый код называется *оптимальным*, если средняя длина кодового слова *минимальна* среди всех побуквенных разделимых кодов для данного распределения вероятностей символов.

Избыточностью кода называется разность между средней длиной кодового слова и предельной энтропией источника сообщений

$$r = L_{cp} - H(p_1,...,p_n).$$

Избыточность кода является показателем качества кода, оптимальный код обладает минимальной избыточностью. Задача эффективного неискажающего сжатия заключается в построении кодов с наименьшей избыточностью, у которых средняя длина кодового слова близка к энтропии источника. К таким кодам относятся классические коды Хаффмана, Шеннона, Фано, Гилберта-Мура и арифметический код.

Взаимосвязь между средней длиной кодового слова и энтропией дискретного вероятностного источника при побуквенном кодировании выражает следующая теорема.

Теорема 1 (*Шеннон*). *Для источника с алфавитом* $A=\{a_1,\ldots,a_n\}$ *и вероятностями* $p_i = P(a_i)$, $\sum_{i=1}^{n} p_i = 1$ *и любого разделимого побуквенного кода средняя длина кодового слова всегда не меньше энтропии*

$$L_{ср} \geq H(p_1,\ldots,p_n)$$

и можно построить разделимый побуквенный код, у которого средняя длина кодового слова превосходит энтропию не больше, чем на единицу:

$$L_{ср} < H(p_1,\ldots,p_n)+1$$

Можно получить более сильные результаты, если кодовые слова приписывать не отдельными буквами, а блоками из L букв источника. Так, для неравномерных блоковых кодов справедлива следующая теорема.

Теорема 2. *Пусть* H_L *– энтропия на букву в блоке длины L дискретного источник. Тогда существует префиксный код для кодирования блоков длины L, такой, что средняя длина кодового слова* $L_{ср}$ *будет удовлетворять неравенствам:*

$$H_L \leq L_{ср} \leq H_L + \frac{1}{L}.$$

Кроме того, в случае бернуллиевского стационарного источника для любого $\varepsilon>0$ *можно выбрать достаточно большое L, чтобы величина* $L_{ср}$ *удовлетворяла неравенствам:*

$$H(p_1,\ldots,p_n) \leq L_{ср} \leq H(p_1,\ldots,p_n) + \varepsilon,$$

и левое неравенство для $L_{ср}$ *никогда не нарушается для разделимого кода.*

Приведем некоторые свойства, которыми обладает любой оптимальный побуквенный код.

Лемма 1. *Для оптимального кода с длинами кодовых слов* $L_1,\ldots,L_n$*: верно соотношение* $L_1 \leq L_2 \leq \ldots \leq L_n$, *если* $p_1 \geq p_2 \geq \ldots \geq p_n$.

Доказательство (от противного): Пусть есть i и j, что $L_i > L_j$ при $p_i > p_j$. Тогда

$$L_i p_i + L_j p_j =$$
$$= L_i p_i + L_j p_j + L_i p_j + L_j p_i - L_i p_j - L_j p_i =$$
$$= p_i(L_i - L_j) - p_j(L_i - L_j) + L_j p_i + L_i p_j =$$
$$= (p_i - p_j)(L_i - L_j) + L_i p_j + L_j p_i > L_i p_j + L_j p_i,$$

т.е. если поменяем местами L_i и L_j, то получим код, имеющий меньшую среднюю длину кодового слова, что противоречит с оптимальности кода. Лемма 1 доказана.

Лемма 2 *Пусть* $\sigma = \langle a_1 \to b_1, \ldots, a_n \to b_n \rangle$ *– схема оптимального префиксного кодирования для распределения вероятностей P,* $p_1 \geq p_2 \ldots \geq p_n > 0$. *Тогда среди элементарных кодов, имеющих максимальную длину, существуют два, которые различаются только в последнем разряде.*

Доказательство. Покажем, что в оптимальной схеме кодирования всегда найдется два кодовых слова максимальной длины. Предположим обратное. Пусть кодовое слово максимальной длины одно и имеет вид $b_n = bx$, $x \in \{0,1\}$. Тогда длина любого элементарного кода не больше длины b, т.е. $L_i \leq |b|$, $i = 1, \ldots, n$. Поскольку схема кодирования префиксная, то кодовые слова $b_1, \ldots, b_{n-1}$ не являются префиксом b. С другой стороны, b не является префиксом кодовых слов $b_1, \ldots, b_{n-1}$. Таким образом, новая схема кодирования $\sigma' = \langle a_1 \to b_1, \ldots, a_n \to b \rangle$ также является префиксной, причем с меньшей средней длиной кодового слова $L_{\sigma'}(P) = L_\sigma(P) - p_n$, что противоречит оптимальности исходной схемы кодирования. Пусть теперь два кодовых слова b_{n-1} и b_n максимальной длины отличаются не в последнем разряде, т.е. $b_{n-1} = b'x'$, $b_n = b''x''$, $b' \neq b''$, $x', x'' \in \{0,1\}$. Причем b', b'' не являются префиксами для других кодовых слов $b_1, \ldots, b_{n-2}$ и наоборот. Тогда новая схема $\sigma'' = \langle a_1 \to b_1, \ldots, a_{n-2} \to b_{n-2}, a_{n-1} \to b'x', a_n \to b'' \rangle$ также является префиксной, причем $L_{\sigma''}(P) = L_\sigma(P) - p_n$, что противоречит оптимальности исходной схемы кодирования. Лемма 2 доказана.

5.2 Оптимальный код Хаффмана

Метод оптимального побуквенного кодирования был разработан в 1952 г. Д. Хаффманом. Оптимальный код Хаффмана обладает минимальной средней длиной кодового слова среди всех побуквенных кодов для данного источника с алфавитом $A=\{a_1, \ldots, a_n\}$ и вероятностями $p_i = P(a_i)$.

Рассмотрим алгоритм построения оптимального кода Хаффмана, который основывается на утверждениях лемм предыдущего параграфа.

1. Упорядочим символы исходного алфавита $A=\{a_1, \ldots, a_n\}$ по убыванию их вероятностей $p_1 \geq p_2 \geq \ldots \geq p_n$.
2. Если $A=\{a_1, a_2\}$, то $a_1 \to 0$, $a_2 \to 1$.
3. Если $A=\{a_1, \ldots, a_j, \ldots, a_n\}$ и известны коды $<a_j \to b_j>$, $j = 1, \ldots, n$, то для алфавита $\{a_1, \ldots a_j', a_j'' \ldots, a_n\}$ с новыми символами a_j' и a_j'' вместо a_j, и вероятностями $p(a_j)=p(a_j')+p(a_j'')$, код символа a_j заменяется на коды $a_j' \to b_j 0$, $a_j'' \to b_j 1$.

Пример. Пусть дан алфавит A=$\{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями

$$p_1=0.36,\ p_2=0.18,\ p_3=0.18,\ p_4=0.12,\ p_5=0.09,\ p_6=0.07.$$

Здесь символы источника уже упорядочены в соответствии с их вероятностями. Будем складывать две наименьшие вероятности и включать

суммарную вероятность на соответствующее место в упорядоченном списке вероятностей до тех пор, пока в списке не останется два символа. Тогда закодируем эти два символа 0 и 1. Далее кодовые слова достраиваются, как показано на рисунке 4.

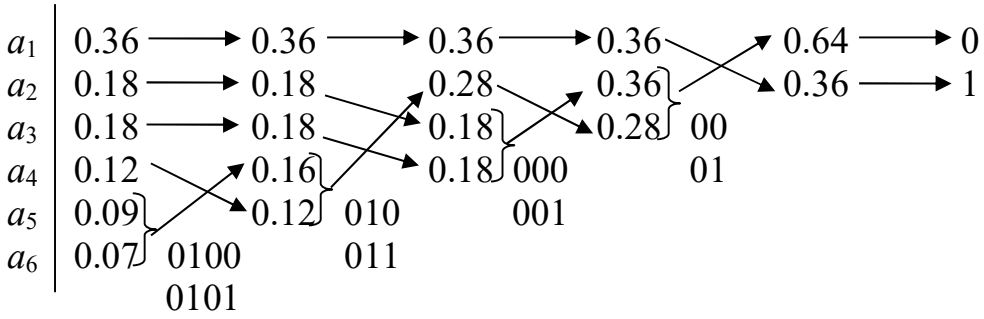


Рисунок 4 Процесс построения кода Хаффмана

Таблица 5 Код Хаффмана

a_i	p_i	L_i	кодовое слово
a_1	0.36	2	1
a_2	0.18	3	000
a_3	0.18	3	001
a_4	0.12	4	011
a_5	0.09	4	0100
a_6	0.07	4	0101

Посчитаем среднюю длину, построенного кода Хаффмана

$$L_{cp}(P)=1\cdot 0.36 + 3\cdot 0.18 + 3\cdot 0.18 + 3\cdot 0.12 + 4\cdot 0.09 + 4\cdot 0.07 = 2.44,$$

при этом энтропия данного источника

$$H(p_1,\ldots,p_6) = -\,0.36\cdot\log 0.36 - 2\cdot 0.18\cdot\log 0.18 - $$
$$- 0.12\cdot\log 0.12 - 0.09\cdot\log 0.09 - 0.07\log 0.07 = 2.37$$

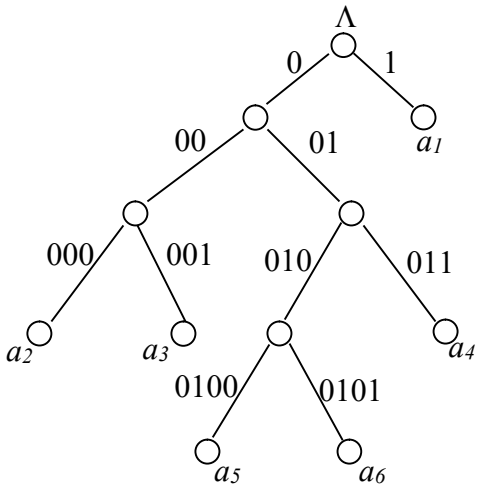


Рисунок 5 Кодовое дерево для кода Хаффмана

Код Хаффмана обычно строится и хранится в виде двоичного дерева, в листьях которого находятся символы алфавита, а на «ветвях» – 0 или 1. Тогда уникальным кодом символа является путь от корня дерева к этому символу, по которому все 0 и 1 собираются в одну уникальную последовательность (рис. 5).

Алгоритм на псевдокоде

Построение оптимального кода Хаффмана (n,P)

Обозначим
n – количество символов исходного алфавита
P – массив вероятностей, упорядоченных по убыванию
C – матрица элементарных кодов
L – массив длин кодовых слов

```
Huffman (n,P)
IF (n=2) C [1,1]:= 0, L [1]:= 1
        C [2,1]:=1, L [2]:=1
ELSE q:= P [n-1]+P [n]
        j:= Up (n,q)                (поиск и вставка суммы)
        Huffman (n-1,P)
        Down (n,j)                  (достраивание кодов)
FI
```

Функция Up (n,q) находит в массиве P место, куда вставить число q, и вставляет его, сдвигая вниз остальные элементы.

```
DO (i=n-1, n-2,…,2)
        IF (P [i-1]≤q) P [i]:=P [i-1]
        ELSE j:=i
 OD
        FI
OD
P [j]:= q
```

Процедура Down (n,j) формирует кодовые слова.

```
S:= C [j,*]  (запоминание j-той строки матрицы элем. кодов в массив S)
L:= L[j]
DO (i=j,…,n-2)
        C [i,*]:= C[i+1,*]  (сдвиг вверх строк матрицы C)
        L [i]:=L [i+1]
OD
C [n-1,*]:= S, C [n,*]:= S  (восстановление префикса кодовых слов из м-ва S)
C [n-1,L+1]:=0
```

C [n,L+1]:=1
L [n-1]:=L+1
L [n]:=L+1

6. ПОЧТИ ОПТИМАЛЬНОЕ КОДИРОВАНИЕ

Рассмотрим несколько классических побуквенных кодов, у которых средняя длина кодового слова близка к оптимальной. Пусть имеется дискретный вероятностный источник, порождающий символы алфавита $A=\{a_1,...,a_n\}$ с вероятностями $p_i = P(a_i)$.

6.1 Код Шеннона

Код Шеннона позволяет построить почти оптимальный код с длинами кодовых слов $L_i < -\log p_i + 1$. Тогда по теореме Шеннона из п. 5.1

$$L_{cp} < H(p_1,..., p_n) + 1.$$

Код Шеннона, удовлетворяющий этому соотношению, строится следующим образом:

1. Упорядочим символы исходного алфавита $A=\{a_1,a_2,...,a_n\}$ по убыванию их вероятностей: $p_1 \geq p_2 \geq p_3 \geq ... \geq p_n$.
2. Вычислим величины Q_i:, которые называются *кумулятивные вероятности*

$$Q_0=0,\ Q_1=p_1,\ Q_2=p_1+p_2,\ Q_3=p_1+p_2+p_3,\ \ldots,\ Q_n=1.$$

3. Представим Q_i в двоичной системе счисления и возьмем в качестве кодового слова первые $\lceil -\log p_i \rceil$ знаков после запятой .

Для вероятностей, представленных в виде десятичных дробей, удобно определить длину кодового слова L_i из соотношения

$$\frac{1}{2^{L_i}} \leq p_i < \frac{1}{2^{L_i-1}},\ i=1,\ldots,n.$$

Пример. Пусть дан алфавит $A=\{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07. Построенный код приведен в таблице 6.

Таблица 6 Код Шеннона

a_i	P_i	Q_i	L_i	кодовое слово
a_1	$1/2^2 \leq 0.36 < 1/2$	0	2	00
a_2	$1/2^3 \leq 0.18 < 1/2^2$	0.36	3	010
a_3	$1/2^3 \leq 0.18 < 1/2^2$	0.54	3	100
a_4	$1/2^4 \leq 0.12 < 1/2^3$	0.72	4	1011
a_5	$1/2^4 \leq 0.09 < 1/2^3$	0.84	4	1101
a_6	$1/2^4 \leq 0.07 < 1/2^3$	0.93	4	1110

Построенный код является префиксным. Вычислим среднюю длину кодового слова и сравним ее с энтропией. Значение энтропии вычислено при построении кода Хаффмана в п. 5.2 (H = 2.37), сравним его со значением средней длины кодового слова кода Шеннона

L_{cp}= 0.36·2+(0.18+0.18)·3+(0.12+0.09+0.07)·4=2.92< 2.37+1,

что полностью соответствует утверждению теоремы Шеннона.

Алгоритм на псевдокоде

Построение кода Шеннона

Обозначим
n – количество символов исходного алфавита
P – массив вероятностей, упорядоченных по убыванию
Q– массив для величин Q_i
L – массив длин кодовых слов
C – матрица элементарных кодов

```
P [0]:=0, Q [0]:=0
DO (i=1,…,n)
        Q [i] := Q [i-1]+P [i]
        L [i]:= -⌈ log2P[i] ⌉                (длину кодового слова определять
OD                                            из соотношения, указанного выше)

DO (i=1,…,n)
        DO (j=1,…,L[i])
                Q [i-1]:=Q [i-1] *2                (формирование кодового слова
                C [i,j]:=⌊Q [i-1]⌋                          в двоичном виде)
                IF (Q [i-1]>1) Q [i-1]:=Q [i-1] - 1 FI
        OD
OD
```

6.2 Код Фано

Метод Фано построения префиксного почти оптимального кода, для которого $L_{cp} < H(p_1,...,p_n)+1$, заключается в следующем. Упорядоченный по убыванию вероятностей Список букв алфавита источника делится на две части так, чтобы суммы вероятностей букв, входящих в эти части, как можно меньше отличались друг от друга. Буквам первой части приписывается 0, а буквам из второй части – 1. Далее также поступают с каждой из полученных частей. Процесс продолжается до тех пор, пока весь список не разобьется на части, содержащие по одной букве.

Пример. Пусть дан алфавит A={a_1, a_2, a_3, a_4, a_5, a_6} с вероятностями p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07. Построенный код приведен в таблице 7 и на рисунке 6.

Таблица 7 Код Фано

a_i	P_i	кодовое слово				L_i
a_1	0.36	**0**	**0**			2
a_2	0.18	**0**	**1**			2
a_3	0.18	**1**	**0**			2
a_4	0.12	**1**	**1**	**0**		3
a_5	0.09	**1**	**1**	**1**	**0**	3
a_6	0.07	**1**	**1**	**1**	**1**	4

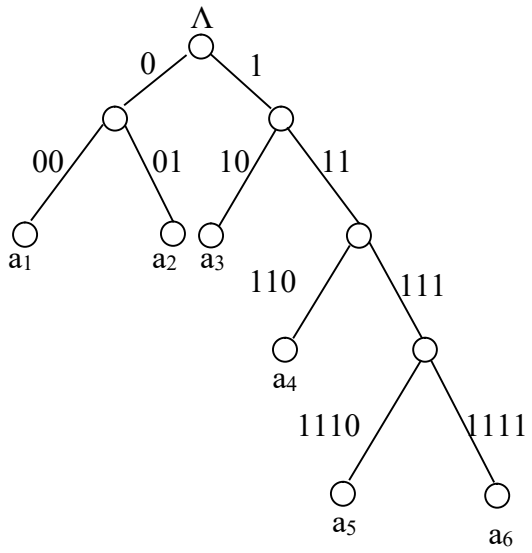


Рисунок 6 Кодовое дерево для кода Фано

Полученный код является префиксным и почти оптимальным со средней длиной кодового слова

$L_{ср}$=0.36·2+0.18·2+0.18·2+0.12·3+0.09·4+0.07·4=2.44

Алгоритм на псевдокоде

Построение кода Фано

Обозначим
P–массив вероятностей символов алфавита
L – левая граница обрабатываемой части массива P
R– правая граница обрабатываемой части массива P
k – длина уже построенной части элементарных кодов

C – матрица элементарных кодов
Length – массив длин элементарных кодов
S_L – сумма элементов первой части массива
S_R – сумма элементов второй части массива.

```
Fano (L,R,k)
IF (L<R)
        k:=k+1
        m:=Med (L,R)
        DO (i=L,…,R)
                IF (i≤m) C [i,k]:=0, Length [i]:= Length [i]+1
                ELSE C [i,k]:=1, Length [i]:= Length [i]+1
                FI
        OD
        Fano (L,m,k)
        Fano (m+1,R,k)
FI
```

Функция Med находит медиану части массива P, т.е. такой индекс $L \le m \le R$, что величина $\left| \sum_{i=L}^{m} p_i - \sum_{i=m+1}^{R} p_i \right|$ минимальна.

```
Med (L,R)
S_L:= 0
DO (i=L,…,R-1)
        S_L:=S_L+ P [i]
OD
S_R:=P [R]
m:= R
DO (S_L ≥ S_R)
        m:=m-1
        S_L:=S_L - P [m]
        S_R:=S_R+ P [m]
OD
Med:= m
```

6.3 Алфавитный код Гилберта – Мура

Е. Н. Гилбертом и Э. Ф. Муром был предложен метод построения алфавитного кода, для которого $L_{cp} < H(p_1,...,p_n) + 2$.

Пусть символы алфавита некоторым образом упорядочены, например, $a_1 \le a_2 \le ... \le a_n$. Код σ называется *алфавитным*, если кодовые слова лексикографически упорядочены, т.е. $\sigma(a_1) \le \sigma(a_2) \le ... \le \sigma(a_n)$.

Процесс построения кода происходит следующим образом.

1. Вычислим величины Q_i, $i=1,n$:

$$Q_1=p_1/2,$$
$$Q_2=p_1+p_2/2,$$
$$Q_3=p_1+p_2+p_3/2,$$
$$\ldots$$
$$Q_n=p_1+p_2+\ldots+p_{n-1}+p_n/2.$$

2. Представим суммы Q_i в двоичном виде.
3. В качестве кодовых слов возьмем $\lceil -\log p_i \rceil + 1$ младших бит в двоичном представлении Q_i, $i=1,...,n$.

Пример. Пусть дан алфавит A=$\{a_1, a_2, a_3, a_4, a_5, a_6\}$ с вероятностями p_1=0.36, p_2=0.18, p_3=0.18, p_4=0.12, p_5=0.09, p_6=0.07. Построенный код приведен в таблице 8.

Таблица 8 Код *Гилберта-Мура*

a_i	P_i	Q_i	L_i	кодовое слово
a_1	$1/2^3 \le 0.18$	0.09	4	0001
a_2	$1/2^3 \le 0.18 < 1/2^2$	0.27	4	0100
a_3	$1/2^2 \le 0.36 < 1/2^1$	0.54	3	100
a_4	$1/2^4 \le 0.07$	0.755	5	11000
a_5	$1/2^4 \le 0.09$	0.835	5	11010
a_6	$1/2^4 \le 0.12$	0.94	5	11110

Средняя длина кодового слова не превышает значения энтропии плюс 2. Действительно,

$$L_{cp}=4\cdot0.18+4\cdot0.18+3\cdot0.36+5\cdot0.07+5\cdot0.09+5\cdot0.12=3.92<2.37+2$$

Алгоритм на псевдокоде

Построение кода Гилберта-Мура

Обозначим
n – количество символов исходного алфавита
P – массив вероятностей символов
Q – массив для величин Q_i
L – массив длин кодовых слов
C – матрица элементарных кодов

```
pr:=0
DO (i=1,…,n)
        Q [i]:= pr+ P[i] /2
```

```
        pr:=pr+ P[i]
        L [i]:= -⌈log2P[i] ⌉+1          (использовать соотношение из п. 6.1)
OD
DO (i=1,…,n)
        DO (j=1,…,L[i])
        Q [i]:=Q [i] *2                          (формирование кодового слова
        C [i,j]:=⌊Q [i]⌋                              в двоичном виде)
        IF (Q [i] >1) Q [i]:=Q [i] - 1 FI
        OD
OD
```

7. АРИФМЕТИЧЕСКИЙ КОД

Идея арифметического кодирования была впервые предложена П. Элиасом. В арифметическом коде кодируемое сообщение разбивается на блоки постоянной длины, которые затем кодируются отдельно. При этом при увеличении длины блока средняя длина кодового слова стремится к энтропии, однако возрастает сложность реализации алгоритма и уменьшается скорость кодирования и декодирования. Таким образом, арифметическое кодирование позволяет получить произвольно малую избыточность при кодировании достаточно больших блоков входного сообщения.

Рассмотрим общую идею арифметического кодирования. Пусть дан источник, порождающий буквы из алфавита $A=\{a_1,a_2,\ldots,a_n\}$ с вероятностями $p_i=P(a_i),\ i=1,\ldots,n$. Необходимо закодировать последовательность символов данного источника $X=x_1x_2x_3x_4$.

1) Вычислим кумулятивные вероятности $Q_0, Q_1,\ldots,Q_n$:

$$Q_0=0$$

$$Q_1=p_1$$

$$Q_2=p_1+p_2$$

$$Q_3=p_1+p_2+p_3$$

...

$$Q_n=p_1+p_2+\ldots+p_n=1$$

2) Разобьем интервал $[Q_0,Q_n)$ (т.е. интервал [0,1)) так, чтобы каждой букве исходного алфавита соответствовал свой интервал, равный ее вероятности (см. рис. 7):

a_1 $[Q_0,Q_1)$

a_2 $[Q_1,Q_2)$

a_3 $[Q_2,Q_3)$

a_4 $[Q_3,Q_4)$

...

a_n $[Q_{n-1},Q_n)$

3) В процессе кодирования будем выбирать интервал, соответствующий текущей букве исходного сообщения, и снова разбивать его пропорционально вероятностям исходных букв алфавита. Постепенно происходит сужение интервала до тех пор, пока не будет закодирован последний символ кодируемого

сообщения. Двоичное представление любой точки, расположенной внутри интервала, и будет кодом исходного сообщения.

На рисунке 7 показан этот процесс для кодирования последовательности $a_3a_2a_3\ldots$

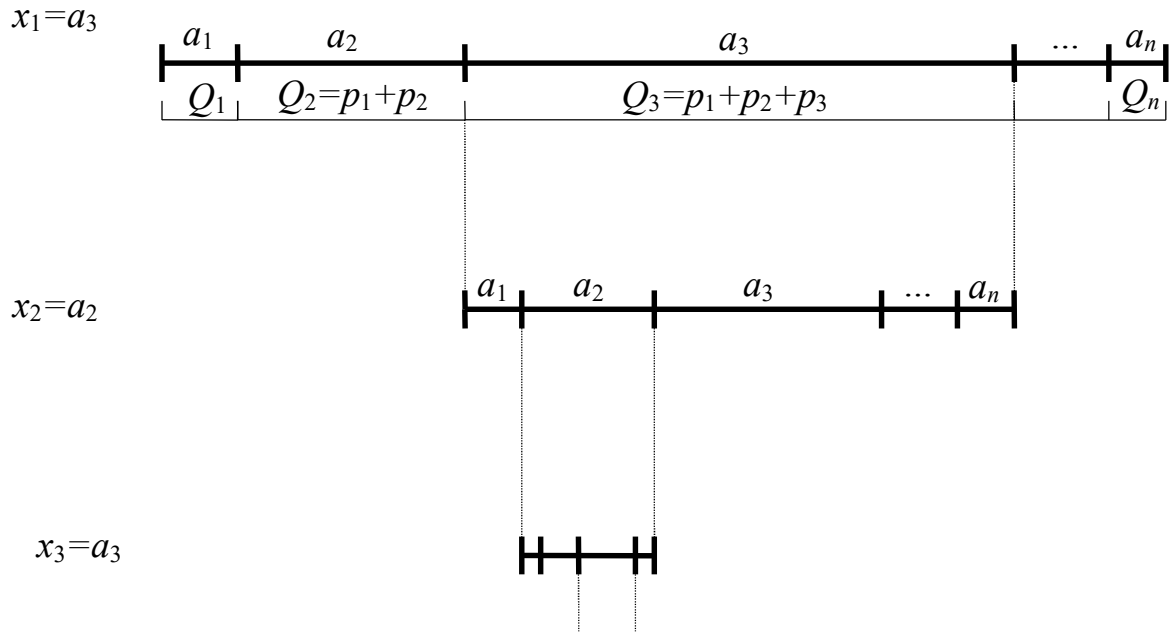


Рисунок 7 Схема арифметического кодирования

Для удобства вычислений введем следующие обозначения:

l_i – нижняя граница отрезка, соответствующего i-той букве исходного сообщения;

h_i – верхняя граница этого отрезка;

r_i – длина отрезка $[l_i, h_i)$, т.е. $r_i = h_i - l_i$.

Возьмем начальные значения этих величин

$$l_0 = Q_0 = 0,\ h_0 = Q_k = 1,\ r_0 = h_0 - l_0 = 1$$

и далее будем вычислять границы интервала, соответствующего кодируемой букве по формулам:

$$l_i = l_{i-1} + r_{i-1} \cdot Q_{m-1},\ h_i = l_{i-1} + r_{i-1} \cdot Q_m,$$

где m – порядковый номер кодируемой буквы в алфавите источника, $m=1,...,n$.

Таким образом, окончательная длина интервала равна произведению вероятностей всех встретившихся символов, а начало интервала зависит от порядка расположения символов в кодируемой последовательности.

Для однозначного декодирования исходной последовательности достаточно взять $\lceil \log(r_k) \rceil$ разрядов двоичной записи любой точки из

интервала $[l_i,h_i)$, где r_k – длина интервала после кодирования k символов источника.

Пример. Рассмотрим кодирование бесконечной последовательности $X=a_3a_2a_3a_1a_4...$ в алфавите $A=\{a_1, a_2, a_3, a_4\}$ с помощью арифметического кода. Пусть вероятности букв исходного алфавита равны соответственно

$$p_1 = 0.1,\ p_2 = 0.4,\ p_3 = 0.2,\ p_4 = 0.3.$$

Вычислим кумулятивные вероятности Q_i :

$$Q_0 = 0,$$
$$Q_1 = p_1 = 0.1,$$
$$Q_2 = p_1 + p_2 = 0.5,$$
$$Q_3 = p_1 + p_2 + p_3 = 0.7,$$
$$Q_4 = p_1 + p_2 + p_3 + p_4 = 1.$$

Получим границы интервала, соответствующего первому символу кодируемого сообщения a_3:

$$l_1 = l_0 + r_0 \cdot Q_2 = 0 + 1 \cdot 0.5 = 0.5,$$
$$h_1 = l_0 + r_0 \cdot Q_3 = 0 + 1 \cdot 0.7 = 0.7,$$
$$r_1 = h_1 - l_1 = 0.7 - 0.5 = 0.2.$$

Для второго символа кодируемого сообщения a_2 границы интервала будут следующие:

$$l_2 = l_1 + r_1 \cdot Q_1 = 0.5 + 0.2 \cdot 0.1 = 0.52,$$
$$h_2 = l_1 + r_1 \cdot Q_2 = 0.5 + 0.2 \cdot 0.5 = 0.6,$$
$$r_2 = h_2 - l_2 = 0.6 - 0.52 = 0.08 \text{ и т.д.}$$

В результате всех вычислений получаем следующую последовательность интервалов для сообщения $a_3a_2a_3a_1a_4$

В начале	[0.0, 1.0)
После просмотра a_3	[0.5, 0.7)
После просмотра a_2	[0.52, 0.6)
После просмотра a_3	[0.56, 0.576)
После просмотра a_1	[0.56, 0.5616)
После просмотра a_4	[0.56112, 0.5616)

Кодом последовательности $a_3a_2a_3a_1a_4$ будет двоичная запись любой точки из интервала [0.56112, 0.5616), например, 0.56112. Для однозначного декодирования возьмем $\lceil \log_2(r_5) \rceil = \lceil \log_2(0.00048) \rceil = 12$ разрядов, получим 100011111010.

Таким образом, при арифметическом кодировании сообщение представляется вещественными числами в интервале [0,1). По мере кодирования сообщения отображающий его интервал уменьшается, а количество битов для представления интервала возрастает. Очередные символы сообщения сокращают величину интервала в зависимости от значений их вероятностей. Более вероятные символы делают это в меньшей степени, чем менее вероятные, и следовательно, добавляют меньше битов к результату.

Алгоритм на псевдокоде

Арифметическое кодирование

Обозначим

l_i – нижняя граница отрезка, соответствующего *i*-той букве исходного сообщения

h_i – верхняя граница этого отрезка

r_i – длина отрезка $[l_i, h_i)$

m – порядковый номер кодируемой буквы в алфавите источника

Q – массив для величин Q_i.

l_0:=0; h_0:=1; r_0:=1; i:=0
DO (not EOF)
    C:=Read( ) (читаем следующий символ из файла)
    i:=i+1
    DO (j=1,…,n)
        IF (C= a_j) m:=j FI
    OD
    $l_i = l_{i-1} + r_{i-1} \cdot Q[m-1]$
    $h_i = l_{i-1} + r_{i-1} \cdot Q[m]$
    $r_i = h_i - l_i$
OD

В начале декодирования известен конечный интервал, например, [0.56112; 0.5616) или любое число из этого интервала, например, 0.56112. Сразу можно определить, что первым закодированным символом был a_3, т. к. число 0.56112 лежит в интервале [0.5; 0.7), выделенном символу a_3. Затем в качестве интервала берется [0.5; 0.7) и в нем определяется диапазон, соответствующий числу 0.56112. Это интервал [0.52, 0.6), выделенный символу a_2 и т.д. Для декодирования необходимо знать количество закодированных символов и исходные вероятности символов.

Алгоритм на псевдокоде

Арифметическое декодирование

Обозначим

l_i – нижняя граница отрезка, соответствующего *i*-той букве исходного сообщения

h_i – верхняя граница этого отрезка

r_i – длина отрезка $[l_i, h_i)$

Q – массив для величин Q_i

length – количество закодированных символов,

value – значение из входного файла.

l_0:=0; h_0:=1; r_0:=1;
value:=ReadCode(); (читаем код из файла)
DO (i=1,…,length)
    DO (j=1,…,n)
        $l_i = l_{i-1} + r_{i-1} \cdot Q[j-1]$
        $h_i = l_{i-1} + r_{i-1} \cdot Q[j]$
        $r_i = h_i - l_i$
        IF ((l_i <= value) и (value< h_i) OD FI
    OD
    Write(a[i]) (пишем символ в выходной файл)
OD

При реализации арифметического кодирования возникают две проблемы:

- необходима арифметика с плавающей точкой теоретически неограниченной точности;
- результат кодирования становится известен только после окончания входного потока.

Для решения этих проблем реальные алгоритмы работают с целыми числами и оперируют с дробями, числитель и знаменатель которых являются целыми числами (например, знаменатель равен 10000h=65536, l_0=0, h_0=65535). При этом с потерей точности можно бороться, отслеживая сближение l_i и h_i и умножая числитель и знаменатель представляющей их дроби на одно и то же число, например на 2. С переполнением сверху можно бороться, записывая старшие биты l_i и h_i в файл только тогда, когда они перестают меняться (т.е. уже не участвуют в дальнейшем уточнении интервала), когда l_i и h_i одновременно находятся в верхней или нижней половине интервала.

8. АДАПТИВНЫЕ МЕТОДЫ КОДИРОВАНИЯ

При решении реальных задач истинные значения вероятностей источника, как правило, неизвестны или могут изменяться с течением времени, поэтому для кодирования сообщений применяют *адаптивные* модификации методов кодирования.

Большинство адаптивных методов для учета изменений статистики исходных данных используют так называемое *окно*. *Окном* называют последовательность символов, предшествующих кодируемой букве, а *длиной окна* – количество символов в окне.

Обычно окно имеет фиксированную длину и после кодирования каждой буквы текста окно передвигается на один символ вправо. Таким образом, код для очередной буквы строится с учетом информации, хранящейся в данный момент в окне (см. рис. 8).

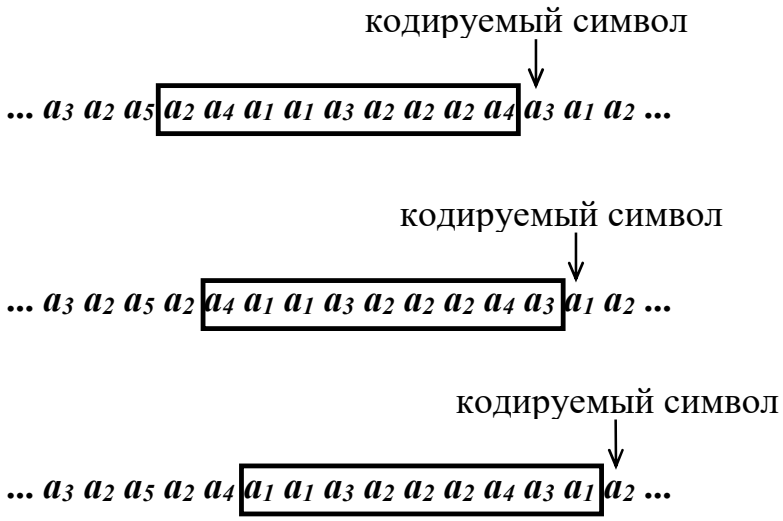


Рисунок 8 Схема перемещения окна при кодировании

При декодировании окно передвигается по тексту аналогичным образом. Информация, содержащаяся в окне, позволяет однозначно декодировать очередной символ.

Оценка избыточности при адаптивном кодировании является достаточно сложной математической задачей, поскольку общая избыточность складывается из двух составляющих: избыточность кодирования и избыточность, возникающая при оценке вероятностей появления символов. Поэтому эффективность методов адаптивного кодирования зачастую оценивают экспериментальным путем.

Однако для всех методов адаптивного кодирования, которые приводятся в этой главе, справедлива следующая теорема:

Теорема. *Величина средней длины кодового слова при адаптивном кодировании удовлетворяет неравенству*

$$L_{ср} \le H + C\,,$$

где H – энтропия источника информации, C – константа, зависящая от размера алфавита источника и длины окна.

8.1 Адаптивный код Хаффмана

В 1978 году Р. Галлагер предложил метод кодирования источников с неизвестной или меняющейся статистикой, основанный на коде Хаффмана, и поэтому названный адаптивным кодом Хаффмана.

Адаптивный код Хаффмана используется как составная часть во многих методах сжатия данных. В нем кодирование осуществляется на основе информации, содержащейся в окне длины *W*. Алгоритм такого кодирования заключается в выполнении следующих действий:

1. Перед кодированием очередной буквы подсчитываются частоты появления в окне всех символов исходного алфавита $A=\{a_1, a_2, ..., a_n\}$. Обозначим эти частоты как $q(a_1), q(a_2), ..., q(a_n)$. Вероятности символов исходного алфавита оцениваются на основе значений частот символов в окне

$$P(a_1)= q(a_1)/W,\ P(a_2) =q(a_2)/W,\ ...,\ P(a_n)= q(a_n)/W.$$

2. По полученному распределению вероятностей строится код Хаффмана для алфавита *A*.
3. Очередная буква кодируется при помощи построенного кода.
4. Окно передвигается на один символ вправо, вновь подсчитываются частоты встреч в окне букв алфавита, строится новый код для очередного символа, и так далее, пока не будет получен код всего сообщения.

Пример. Рассмотрим пример адаптивного кодирования с помощью метода Хаффмана для алфавита $A=\{a_1, a_2, a_3, a_4\}$ и длины окна W=6 (см. рис. 9).

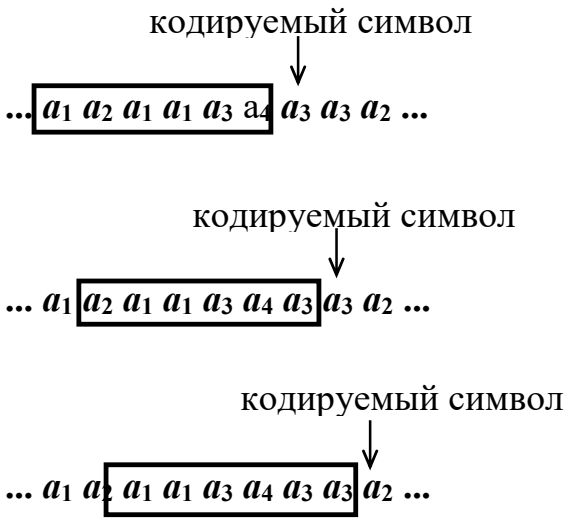


Рисунок 9 Кодирование адаптивным кодом Хаффмана

Рассмотрим подробно этапы кодирования сообщения.

При кодировании буквы a_3 получаем следующие частоты встреч символов в окне: $q(a_1)=3$, $q(a_2)=1$, $q(a_3)=1$, $q(a_4)=1$. Тогда вероятности символов алфавита A оцениваются так

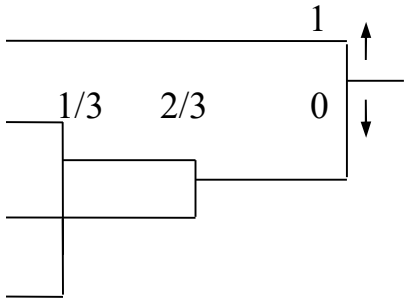
$$P(a_1)=\frac{1}{2},\ P(a_2)=\frac{1}{6},\ P(a_3)=\frac{1}{6},\ P(a_4)=\frac{1}{6}.$$

Строим код Хаффмана для полученного распределения вероятностей (см. рис. 10 (а)) и кодируем символ a_3 кодовым символом 001.

а)

Символы	Вероятности	Построение кода	Кодовые слова
a_1	1/2	1	1
a_2	1/6	1/2 0	01
a_3	1/6	1/3	001
a_4	1/6		000

б)

Символы	Вероятности	Построение кода	Кодовые слова
a_1	1/3		1
a_2	1/6		011
a_3	1/3		00
a_4	1/6		010

в)

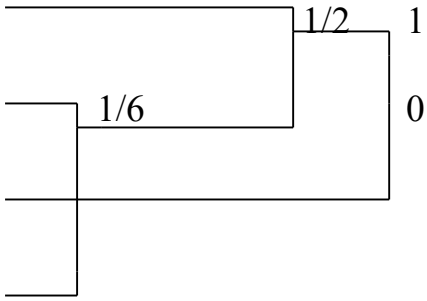
Символы	Вероятности	Построение кода	Кодовые слова
a_1	1/3		11
a_2	0		101
a_3	1/2		0
a_4	1/6		100

Рисунок 10 Построение адаптивного кода Хаффмана

Далее передвигаем окно на один символ вправо и снова подсчитываем частоты встреч символов в окне $q(a_1)=2$, $q(a_2)=1$, $q(a_3)=2$, $q(a_4)=1$ и оцениваем вероятности:

$$P(a_1)=\frac{1}{3},\ P(a_2)=\frac{1}{6},\ P(a_3)=\frac{1}{3},\ P(a_4)=\frac{1}{6}.$$

Строим код на основе полученных оценок вероятностного распределения (см. рис. 10 (б)) и кодируем очередной символ a_3 другим кодовым словом – 00. В этом случае частота встречи в окне символа a_3 увеличилась, соответственно уменьшилась длина его кодового слова.

Снова передвигаем окно на один символ вправо, подсчитываем частоты встреч символов в окне $q(a_1)=2$, $q(a_2)=0$, $q(a_3)=3$, $q(a_4)=1$ и оцениваем вероятности:

$$P(a_1)=\frac{1}{3},\ P(a_2)=0,\ P(a_3)=\frac{1}{2},\ P(a_4)=\frac{1}{6}.$$

Строим код на основе полученных оценок вероятностного распределения (см. рис. 10 (в)) и кодируем очередной символ a_2 кодовым

словом – 101. Таким образом, после кодирования символов $a_3a_3a_2$ получаем кодовую последовательность 00100101.

Алгоритм на псевдокоде

Адаптивный код Хаффмана

Обозначим
w – окно
mas – массив вероятностей символов и кодовых слов

```
DO (i=1,…n) w[i]:=a[i] OD                    (заполняем окно символами алфавита)
DO (not EOF)                                 (пока не конец входного файла)
      DO (i=1,…,n) mas[i].p:=0 OD
      DO (i=1,…,n) mas[w[i]].p:= mas[w[i]].p +1 OD     (частоты встречи
                                                        символов в окне)
      DO (i=1,…,n) mas[w[i]].p:= mas[w[i]].p/n OD      (вероятности символов
                                                        в окне)
      Сортировка(mas)
      DO (i=1,…,n)                                (определяем количество
            IF (mas[i].p=0) OD                     ненулевых вероятностей)
      OD
      Huffman(i,P)                           (строим код Хаффмана)
      C:=Read( )                             (читаем следующий символ из файла)
      Write(mas[C].code)                     (код символа – в выходной файл)
      DO (i=1,…,n-1) w[i]:= w[i+1] OD
      w[n]:=C                                (включаем в окно текущий символ)
OD
```

8.2 Код «Стопка книг»

Этот метод был предложен Б. Я. Рябко в 1980 году. Название метод получил по аналогии со стопкой книг, лежащей на столе. Обычно сверху стопки находятся книги, которые недавно использовались, а внизу стопки – книги, которые использовались давно, и после каждого обращения к стопке использованная книга кладется сверху стопки.

До начала кодирования буквы исходного алфавита упорядочены произвольным образом и каждой позиции в стопке присвоено свое кодовое слово, причем первой позиции стопки соответствует самое короткое кодовое слово, а последней позиции – самое длинное кодовое слово. Очередной символ кодируется кодовым словом, соответствующим номеру его позиции в стопке, и переставляется на первую позицию в стопке.

Пример. Приведем описание кода на примере алфавита $A=\{a_1,a_2,a_3,a_4\}$. Пусть кодируется сообщение $a_3a_3a_4a_4a_3\ldots$

- Символ a_3 находится в третьей позиции стопки, кодируется кодовым словом 110 и перемещается на первую позицию в стопке, при этом символы a_1 и a_2 смещаются на одну позицию вниз.
- Следующий символ a_3 уже находится в первой позиции стопки, кодируется кодовым словом 0 и стопка не изменяется.
- Символ a_4 находится в последней позиции стопки, кодируется кодовым словом 111 и перемещается на первую позицию в стопке, при этом символы a_1, a_2, a_3 смещаются на одну позицию вниз.
- Следующий символ a_4 уже находится в первой позиции стопки, кодируется кодовым словом 0 и стопка не изменяется.
- Символ a_3 находится во второй позиции стопки, кодируется кодовым словом 10 и перемещается на первую позицию в стопке, при этом символ a_4 смещается на одну позицию вниз.

№	Кодовое слово	Начальная «стопка»	Преобразования «стопки»				
1	0	a_1	a_3	a_3	a_4	A_4	a_3
2	10	a_2	a_1	a_1	a_3	A_3	a_4
3	110	a_3	a_2	a_2	a_1	A_1	a_1
4	111	a_4	a_4	a_4	a_2	A_2	a_2

Рисунок 11 Кодирование методом «стопка книг»

Таким образом, закодированное сообщение имеет следующий вид:

$$\underbrace{110}_{a_3}\ \underbrace{0}_{a_3}\ \underbrace{111}_{a_4}\ \underbrace{0}_{a_4}\ \underbrace{10}_{a_3}\ldots$$

Рассмотренный метод сжимает сообщение за счет того, что часто встречающиеся символы будут находиться в верхних позициях «стопки книг» и кодироваться более короткими кодовыми словами. Эффективность метода особенно заметна при кодировании серий одинаковых символов. В этом случае все символы серии, начиная со второго, будут кодироваться самым коротким кодовым словом, соответствующим первой позиции «стопки книг».

При декодировании используется такая же «стопка книг», находящаяся первоначально в том же состоянии. Над «стопкой» проводятся такие же преобразования, что и при кодировании. Это гарантирует однозначное восстановление исходной последовательности.

Приведение описание данного алгоритма на псевдокоде.

Алгоритм на псевдокоде

Кодирование кодом «Стопка книг»

Обозначим
code – массив кодовых слов для позиции «стопки»
s_in – строка для кодирования
s_out – результат кодирования
S – строка, содержащая исходный алфавит

```
l:=<длина s_in>
DO (i=1,…l)
        T:=<номер символа s_in[i] в строке S>
        s_out:=s_out+code[t]
        stmp:=S[t]
        delete(S,t,1)                    (преобразование
        insert(stmp,S,1)                      строки S)
OD
```

8.3 Интервальный код

Интервальный код был предложен П. Элиасом в 1987 году. В нем используется окно длины W, т.е. при кодировании символа x_i исходной последовательности учитываются W предыдущих символов:

$$\ldots \underbrace{x_{i-W} \ldots x_{i-2} x_{i-1}} x_i x_{i+1} x_{i+2} \ldots$$

При кодировании символа x_i определяется расстояние (интервал) до его предыдущей встречи в окне длины W. Обозначим это расстояние $\lambda(x_i)$. Если символ есть x_i в окне, то значение $\lambda(x_i)$ равно номеру позиции символа в окне. Позиции в окне нумеруются справа налево. Если символа x_i нет в окне, то $\lambda(x_i)$ присваивается значение $W+1$, а x_i кодируется словом, состоящим из $\lambda(x_i)$ и самого символа x_i. После кодирования очередного символа окно сдвигается вправо на один символ.

Пример. Рассмотрим описание кода для исходного алфавита $A=\{a_1,a_2,a_3,a_4\}$, пусть длина окна $W=3$. Возьмем некоторый префиксный код σ для записи числа $\lambda(X_i)$:

$\lambda(x_i)$	1	2	3	4
σ_i	0	10	110	111

Пусть кодируется последовательность $a_1a_1a_2a_3a_2a_2\ldots$ (см. рис. 12)

1. Сначала символа a_1 нет в окне, поэтому на выход кодера передается 111 и восьмибитовый ASCII-код символа a_1 Затем окно сдвигается на одну позицию вправо.

2. При кодировании следующего символа a_1 на выход кодера передается 0, так как теперь символ a_1 находится в первой позиции окна. Далее окно снова сдвигается на одну позицию вправо.
3. Следующий символ a_2 кодируется комбинацией 111 и восьмибитовым ASCII-кодом символа a_2, так как символа a_2 нет в окне. Окно снова сдвигается на одну позицию вправо.
4. Следующий символ a_3 кодируется комбинацией 111 и восьмибитовым ASCII-кодом символа a_3 , так как символа a_3 нет в окне. Окно снова сдвигается на одну позицию вправо.
5. Следующий кодируемый символ a_2 находим во второй позиции окна и поэтому кодируем комбинацией 10. Окно снова сдвигается на одну позицию вправо.
6. И последний символ a_2 находим во первой позиции окна и поэтому кодируем комбинацией 0. Таким образом, закодированное сообщение имеет следующий вид:

$$111\ a_1\ \ 0\ \ 111\ a_2\ 111\ \ a_3\ \ 10\ \ 0$$

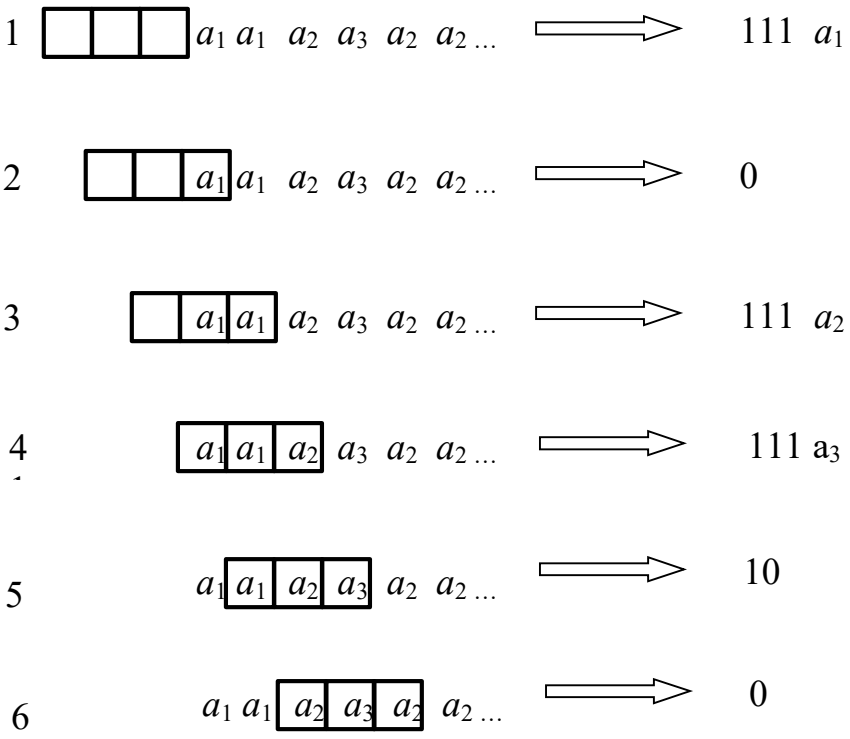


Рисунок 12 Кодирование интервальным кодом

При декодировании используется такое же окно, как и при кодировании. По принятому кодовому слову можно определить, в какой позиции окна находится данный символ. Если поступает код 111, то декодер считывает следующий за ним символ. Каждая декодированная буква полностью включается в окно.

Сжатие данных интервальным кодом достигается за счет малых расстояний между встречами более вероятных букв, что позволяет получить более короткие кодовые слова.

Алгоритм на псевдокоде

Кодирование интервальным кодом

Обозначим
code – массив кодовых слов для записи числа $\lambda(X_i)$
s_in – строка для кодирования
s_out – результат кодирования

```
l:=<длина s_in>
DO (i=1,…l)
        t:=0
        found:=нет
        DO (j=i-1,…,i-W)
                t:=t+1
                IF (j>0) и (s_in[i]=s_in[j])
                        s_out:=s_out+code[t]
                        found:=да
                        OD
                FI
        OD
        IF (not found) s_out:=s_out+code[n]+s_in[i] FI
OD
```

8.4 Частотный код

В 1990 году Б. Я. Рябко предложил алгоритм кодирования, использующий алфавитный код Гилберта-Мура, и названный частотным. Частотный код относится к адаптивным методам сжатия с постоянной избыточностью. Средняя длина кодового слова для этого метода определяется только длиной окна, по которому оценивается статистика кодируемых данных, к тому же частотный код имеет достаточно высокую скорость кодирования и декодирования.

Рассмотрим алгоритм построения частотного кода для источника с алфавитом $A=\{a_1, a_2, \ldots, a_n\}$. Пусть используется окно длины W, т.е. при кодировании символа x_i исходной последовательности учитываются W предыдущих символов:

$$\ldots \underbrace{x_{i-W} \ldots x_{i-2} x_{i-1}}_{\text{окно}} x_i x_{i+1} x_{i+2} \ldots$$

Возьмем размер окна такой, что $W=(2^r-1)\cdot n$, *где* $n=2^k$ – размер исходного алфавита, r, k – целые числа.

Порядок построения кодовой последовательности следующий:

1. Сначала оценивается число встреч в окне $x_{i-W}\ldots x_{i-1}$ всех букв исходного алфавита. Обозначим эти величины через $P(a_j)$, $j=1,\ldots,n$
2. $P(a_j)$ увеличивается на единицу и обозначается как $\hat{P}(a_j)=P(a_j)+1$.
3. Вычисляются суммы Q_i, $i=1,\ldots,n$

$$
\begin{aligned}
Q_1 &= \hat{P}(a_1),\\
Q_2 &= \hat{P}(a_1)+\frac{\hat{P}(a_2)}{2},\\
Q_3 &= \hat{P}(a_1)+\hat{P}(a_2)+\frac{\hat{P}(a_3)}{2},\\
&\ldots\\
Q_n &= \hat{P}(a_1)+\ldots+\hat{P}(a_{n-1})+\frac{\hat{P}(a_n)}{2}.
\end{aligned}
$$

4. Для кодового слова символа a_j берется k знаков от двоичного разложения Q_j, где $k=1+\log(n+W)-\lfloor\log\hat{P}(a_j)\rfloor$.
5. Далее окно сдвигается на один символ вправо и для кодирования следующего символа алгоритм вновь повторяется.

Пример. Пусть $A=\{a_1, a_2, a_3, a_4\}$, длина окна $W=4$. Необходимо закодировать последовательность символов

$$\ldots\underbrace{a_3a_3a_3a_4}_{W=4}a_3\ldots$$

Построим кодовое слово для символа a_3.

1. Оценим частоты встреч в текущем "окне" всех символов алфавита:

$$\hat{P}(a_1)=\hat{P}(a_2)=1,\ \hat{P}(a_3)=4,\ \hat{P}(a_4)=2.$$

2. Вычислим суммы Q_j:

$$
\begin{aligned}
Q_1 &= 1\\
Q_2 &= 1+0.5=1.5\\
Q_3 &= 1+1+2=4\\
Q_4 &= 1+1+4+1=7
\end{aligned}
$$

3. Определим длину кодового слова для a_3:

$$k=1+\log(4+4)-\lfloor\log 4\rfloor=1+3-2=2$$

4. Двоичное разложение $Q_3 = 100_2$, берем первые 2 знака. Таким образом, для текущего символа a_3 кодовое слово 10.

Алгоритм на псевдокоде

Частотный код

Обозначим
w – окно
W – размер окна
P – массив частот символов
Q – массив для величин Q_i.

```
DO (i=1,…n) w[i]:=a[i] OD                (заполняем окно символами алфавита)
DO (not EOF)                             (пока не конец входного файла)
    DO (i=1,…,n) P [i]:=1 OD
    DO (i=1,…,n) P [w[i]]:= P [w[i]] +1 OD        (частоты встречи
                                                   символов в окне)
    pr:=0
    DO (i=1,…,n)
        Q [i] := pr+ P [i] /2                     (вычисляем суммы)
        pr:=pr+ P [i]
    OD
    C:=Read( )                           (читаем следующий символ из файла)
    DO (j=1,…,n)
        IF (C= a[i]) m:=j FI             (определяем номер символа в алфавите)
    OD
    k = 1 + log2 (n+W) – ⌊log2 P[m] ⌋                (длина кодового слова)
    DO (j=1,…,k)                          (формирование кодового слова
        Q [m]:=Q [m] *2                              в двоичном виде)
        code:=⌊Q [m]⌋
        IF (Q [m] >1) Q [m]:= Q [m] - 1 FI
    OD
    Write(code)                          (код символа – в выходной файл)
    DO (i=0,…,n-1) w[i]:= w[i+1] OD
    w[n]:=C                          (включаем в окно текущий символ)
OD
```

9. СЛОВАРНЫЕ КОДЫ КЛАССА LZ

Словарные коды класса LZ широко используются в практических задачах. На их основе реализовано множество программ-архиваторов. Эти методы также используются при сжатии изображений в модемах и других цифровых устройствах передачи и хранения информации.

Словарные методы сжатия данных позволяют эффективно кодировать источники с неизвестной или меняющейся статистикой. Важными свойствами этих методов являются высокая скорость кодирования и декодирования, а также относительно небольшая сложность реализации. Кроме того, LZ-методы обладают способностью быстро адаптироваться к изменению статистической структуры сообщений.

Словарные коды интенсивно исследуются и конструируются, начиная с 1977 года, когда появилось описание первого алгоритма, предложенного А. Лемпелом и Я. Зивом. В настоящее время существует множество методов, объединенных в класс LZ-кодов, которые представляют собой различные модификации метода Лемпела-Зива.

Общая схема кодирования, используемая в LZ-методах, заключается в следующем. При кодировании сообщение разбивается на слова переменной длины. При обработке очередного слова ведется поиск ему подобного в ранее закодированной части сообщения. Если слово найдено, то передается соответствующий ему код. Если слово не найдено, то передается специальный символ, обозначающий его отсутствие, и новое обозначение этого слова. Каждое новое слово, не встречавшееся ранее, запоминается, и ему присваивается индивидуальный код.

При декодировании по принятому коду определяется закодированное слово. В случае получения специального символа, сигнализирующего о передаче нового слова, принятое слово запоминается, и ему присваивается такой же, как и при кодировании, код. Таким образом, декодирование является однозначным, т.к. каждому слову соответствует свой собственный код.

По способу организации хранения и поиска слов словарные методы можно разделить на две большие группы:

- алгоритмы, осуществляющие поиск слов в какой-либо части ранее закодированного текста, называемой окном;
- алгоритмы, использующие адаптивный словарь, который включает ранее встретившиеся слова. Если словарь заполняется до окончания процесса кодирования, то в некоторых методах он обновляется (на место ранее встретившихся слов записываются новые), а в некоторых кодирование продолжается без обновления словаря.

Алгоритмы класса LZ отличаются размерами окна, способами кодирования слов, алгоритмами обновления словаря и т.п. Все указанные факторы влияют и на характеристики этих методов: скорость кодирования,

объем требуемой памяти и степень сжатия данных, различные для разных алгоритмов. Однако в целом методы из класса LZ представляют значительный практический интерес и позволяют достаточно эффективно сжимать данные с неизвестной статистикой.

9.1 Кодирование с использованием скользящего окна

Рассмотрим основные этапы кодирования сообщения $X=x_1x_2x_3x_4\ldots$, которое порождается некоторым источником информации с алфавитом A. Пусть используется окно длины W, т.е. при кодировании символа x_i исходной последовательности учитываются W предыдущих символов:

$$\ldots\underbrace{x_{i-W}\ldots x_{i-2}x_{i-1}}_{\text{окно}}x_ix_{i+1}x_{i+2}\ldots$$

Сначала осуществляется поиск в окне символа x_1. Если символ не найден, то в качестве кода передается 0 как признак того, что этого символа нет в окне и двоичное представление $\underline{x_1}$.

Если символ x_1 найден, то осуществляется поиск в окне слова x_1x_2, начинающегося с этого символа. Если слово x_1x_2 есть в окне, то производится поиск слова $x_1x_2x_3$, затем $x_1x_2x_3x_4$ и так далее, пока не будет найдено слово, состоящее из наибольшего количества входных символов в порядке их поступления. В этом случае в качестве кода передается 1 и пара чисел (i, j), указывающая положение найденного слова в окне (i – номер позиции окна, с которой начинается это слово, j – длина этого слова, позиции в окне нумеруются справа налево). Затем окно сдвигается на j символов вправо по тексту и кодирование продолжается.

Для кодирования чисел (i, j) могут быть использованы рассмотренные ранее коды целых чисел.

Пример. Пусть алфавит источника $A=\{a, b, c\}$, длина окна W=6. Необходимо закодировать исходное сообщение *bababaabacabac.* (см. рис. 13)

После кодирования первых шести букв окно будет иметь вид *bababa*.

- Далее проверяем наличие в окне буквы *a.* Она найдена, добавляем к ней *b*, ищем в окне *ab.* Эта пара есть в окне, добавляем букву *a*, ищем *aba.* Это слово есть в окне, добавляем букву *c*, ищем *abac*. Этого слова нет в окне, тогда кодируем *aba* кодовой комбинацией (1,3,3), где 1– признак того, что слово есть в «окне», 3– номер позиции в окне, с которой начинается это слово, 3– длина этого слова.
- Далее окно сдвигаем на 3 символа вправо и ищем в окне букву *c*. Ее нет в окне, поэтому кодируем комбинацией (0, «*c*»), где 0– признак того, что буквы нет в окне, «*c*»– двоичное представление буквы. Окно сдвигаем на 1 символ вправо.
- Ищем в окне букву *a*, она найдена, добавляем к ней *b*, ищем в окне *ab.* Эта пара есть в окне, добавляем букву *a*, ищем *aba.* Это

слово есть в окне, добавляем букву *c*, ищем *abac*. Это слово есть в окне, тогда кодируем *abac* кодовой комбинацией (1,4,4), где 1 – признак того, что слово есть в окне, 4 –номер позиции в окне, с которой начинается это слово, 4 – длина этого слова.

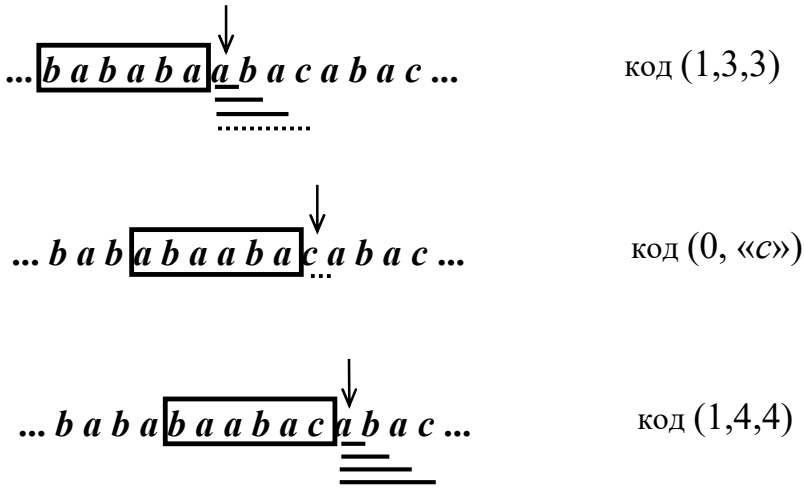


Рисунок 13 Кодирование последовательности *bababaabacabac*

9.2 Кодирование с использованием адаптивного словаря

В этих словарных методах для хранения слов используется адаптивный словарь, заполняющийся в процессе кодирования. Перед началом кодирования в словарь включаются все символы алфавита источника. При кодировании сообщения $X=x_1x_2x_3x_4\ldots$ сначала читаются два символа x_1x_2, поскольку это слово отсутствует в словаре, то передается код символа x_1 (обычно это номер строки, содержащей найденный символ или слово). В свободную строку таблицы записывается слово x_1x_2, далее читается символ x_3 и осуществляется поиск в словаре слова x_2x_3. Если это слово есть в словаре, то проверяется наличие слова $x_2x_3x_4$ и так далее. Таким образом, слово из наибольшего количества входных символов, найденное в словаре, кодируется как номер строки, его содержащей.

Пример. Пусть алфавит источника $A=\{a,\ b,\ c\}$, размер словаря $V=8$. Необходимо закодировать исходное сообщение *abababaabacabac*.

1. Запишем символы алфавита A в словарь, каждому символу припишем кодовое слово длины $L=\lceil \log_2 V \rceil = \lceil \log_2 8 \rceil = 3$.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	–	
4	–	
5	–	
6	–	
7	–	

2. Читаем первые две буквы *ab*, ищем слово *ab* в словаре. Этого слова нет, поэтому поместим слово *ab* в свободную 3-ю строку словаря, а букву *a* закодируем кодом 000.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	–	
5	–	
6	–	
7	–	

3. Далее читаем букву *a* и ищем в словаре слово *ba.* Этого слова нет, поэтому запишем в 4-ю строку словаря слово *ba,* букву *b* закодируем кодом 001.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	*ba*	100
5	–	
6	–	
7	–	

4. Читаем букву *b*, ищем в словаре слово *ab.* Это слово есть в словаре в строке 3. Читаем следующую букву *a*, получим слово *aba*, его нет в словаре. Запишем слово *aba* в 5-ю строку словаря, и закодируем *ab* кодом 011.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	*ba*	100
5	*aba*	011
6	–	
7	–	

5. Читаем букву *b,* ищем в словаре слово *ab.* Это слово есть в словаре в строке 3. Читаем следующую букву *a*, получим слово *aba.* Это слово есть в словаре в строке 5. Читаем букву *a*, получим слово *abaa,* его нет в словаре. Запишем слово *abaa* в 6-ю строку словаря, и закодируем *aba* кодом 101.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	*ba*	100
5	*aba*	011
6	*abaa*	101
7	–	

6. Читаем букву *b,* ищем в словаре слово *ab.* Это слово есть в словаре в строке 3. Читаем следующую букву *a*, получим слово *aba.* Это слово есть в словаре в строке 5. Читаем букву *c*, получим слово *abac,* его нет в словаре. Запишем слово *abac* в 7-ю строку словаря, и закодируем *aba* кодом 101. Если словарь заполняется до окончания кодирования, то можно записывать новые слова в словарь, начиная со строки с наибольшим номером, удаляя ранее записанные там слова.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	*ba*	100
5	*aba*	101
6	*abaa*	110
7	*abac*	111

7. Читаем букву *a,* ищем в словаре слово *ca.* Этого слова нет в словаре, поэтому запишем слово *ca* в 7-ю строку словаря, удалив слово *abac,* и закодируем букву *c* кодом 010.

Номер строки	Слово	Код
0	*a*	000
1	*b*	001
2	*c*	010
3	*ab*	011
4	*ba*	100
5	*aba*	101
6	*abaa*	110
7	~~*abac*~~ *ca*	111

8. Читаем букву *b,* ищем в словаре слово *ab.* Это слово есть в словаре в строке 3. Читаем следующую букву *a*, получим слово *aba.* Это слово есть в словаре в строке 5. Читаем букву *c*, получим слово *abac,* его нет в словаре. Запишем слово *abac* в 6-ю строку словаря, и закодируем *aba* кодом 101.

Номер строки	Слово	Код
0	a	000
1	b	001
2	c	010
3	ab	011
4	ba	100
5	aba	101
6	~~abaa~~ abac	110
7	ca	111

9. Закодируем букву *c* кодом 010. Конец входной последовательности.

Таким образом, входное сообщение будет закодировано так

Вход кодера:	*a*	*b*	*ab*	*aba*	*aba*	*c*	*aba*	*c*
Выход кодера:	000	001	011	101	101	010	101	010

Алгоритм на псевдокоде

Кодирование с адаптивным словарем

```
Обозначим
CurCode – текущий код
PrevCode – предыдущий код
M – массив, содержащий текущую последовательность
L – длина текущей последовательности
C – словарь (массив строк)
S – текущая длина кода
DicPos – количество последовательностей в словаре

<Инициализация словаря символами исходного алфавита>
S:=9; L:=0; DicPos:=257                 (256+конец сжатия)
DO (not EOF)
        CurCode:=Read( )                (читаем следующий байт из файла)
        M[L]:=CurCode;   L:=L+1
        IF (текущая последовательность найдена в словаре)
                CurCode:=номер найденной последовательности
        ELSE
                C[DicPos]:=M;  DicPos:=DicPos+1
                IF (log(DicPos)+1)>S  S:=S+1 FI (использовать соотношение п.6.1)
                Write(PrevCode,S)       (пишем в выходной файл S бит PrevCode)
                M[0]:=CurCode;  L:=1
        FI
        PrevCode:=CurCode
OD
Write(PrevCode,S)                       (сохраняем оставшийся код)
Write(256,S)                            (конец сжатия)
```

Рассмотрим теперь на примере ранее закодированного сообщения *abababaabacabac* (алфавит источника $A=\{a, b, c\}$, размер словаря V=8) процесс декодирования. Закодированная последовательность имела такой вид

000001011101101010101010

1. Запишем символы алфавита A в словарь, каждому символу припишем кодовое слово длины $L = \lceil \log_2 V \rceil = \lceil \log_2 8 \rceil = 3$. (Процесс заполнения словаря будет таким же, как и при кодировании.)

2. Читаем первые три бита кодовой последовательности (код 000), по коду найдем в словаре букву *a*.

3. Читаем следующий код 001, по коду найдем в словаре букву *b*. Получим новое слово *ab*, которого нет в словаре, поместим слово *ab* в свободную 3-ю строку словаря. На выход декодера передаем букву *a*, букву *b* запоминаем.

4. Читаем код 011, по коду находим в словаре слово *ab*. Добавляем первую букву *a* к предыдущему декодированному слову *b*, получим слово *ba*, его нет в словаре. Поместим слово *ba* в свободную 4-ю строку словаря. На выход декодера передаем букву *b*, слово *ab* запоминаем.

5. Читаем код 101, такого кода нет в словаре. Тогда добавляем к слову *ab*

первую букву этой же последовательности – *a*, получим слово *aba*, его нет в словаре. Поместим слово *aba* в свободную 5-ю строку словаря. На выход декодера передаем слово *ab*, слово *aba* запоминаем.

6. Читаем код 101, по коду находим в словаре слово *aba*, добавляем первую букву *a* к предыдущему декодированному слову *aba*, получим *abaa.* Добавим слово *abaa* в словарь в свободную 6-ю строку. На выход декодера передаем слово *aba* , и слово *aba* запоминаем.

7. Читаем код 010, по коду находим в словаре букву *c*, добавляем букву *c* к предыдущему декодированному слову *aba*, получим *abac.* Добавим слово *abac* в словарь в свободную 7-ю строку. На выход декодера передаем слово *aba* , букву *c* запоминаем.

8. Читаем код 101, по коду находим в словаре слово *aba*, добавляем первую букву *a* к предыдущей декодированной букве *c*, получим слово *ca.*

Так как словарь заполнился до окончания декодирования, то будем записывать новые слова в словарь, начиная со строки с наибольшим номером, удаляя ранее записанные там слова. Добавим слово *ca* в 7-ю строку словаря вместо слова *abac*. На выход декодера передаем букву *c*, слово *aba* запоминаем.

9. Читаем код 010, по коду находим в словаре букву *c*, добавляем букву *c* к предыдущему декодированному слову *aba*, получим *abac.* Добавим слово *abac* в 6-ю строку словаря вместо слова *abaa*. На выход декодера передаем слово *aba,* букву *c* запоминаем.

10. На выход декодера передаем букву *c*.

В результате декодирования получим исходное сообщение

Вход декодера:	000	001	011	101	101	010	101	010
Выход декодера:	*a*	*b*	*ab*	*aba*	*aba*	*c*	*aba*	*c*

Алгоритм на псевдокоде

Декодирование с адаптивным словарем

```
Обозначим
CurCode – текущий код
PrevCode – предыдущий код
M – массив, содержащий текущую последовательность
L – длина текущей последовательности
C – словарь (массив строк)
S – текущая длина кода
DicPos – количество последовательностей в словаре

<Инициализация словаря символами исходного алфавита>
S:=9;  L:=0;   DicPos:=257
DO
        CurCode:=Read(S)                    (читаем из файла S бит)
        IF  (CurCode=256)  break  FI
        IF (C[CurCode]<>0)    (в словаре найдена послед-ть с номером CurCode)
                M[L]:=C[CurCode][0]         (в конец текущей последовательности
                     приписываем первый символ найденной последовательности)
                L:=L+1
        ELSE M[L]:=M[0];  L:=L+1
        FI
        IF (текущая последовательность M не найдена в словаре C)
                Write(C[PrevCode])
                C[DicPos]:=M                (добавляем текущую послед-ть в словарь)
                DicPos:=DicPos+1
                IF  (log DicPos+1)>S  S:=S+1 FI  (использовать соотношение п.6.1)
                M:=C[CurCode]               (в текущую послед-ть заносим слово
                L=длина слова                         с номером CurCode)
        FI
        PrevCode:=CurCode
OD
Write(C[PrevCode])
```

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторные работы выполняются на языках высокого уровня (Паскаль, Си). По согласованию с преподавателем допускается выполнения

лабораторных работ в средах Delphi, Builder C++, Visual C++. Для зачета по лабораторной работе студенту необходимо представить

- Исходные тексты программ с подробными комментариями;
- Исполняемые файлы;
- Отчет по лабораторной работе.

Отчет должен включать в себя следующие разделы

- Формулировку задания
- Описание основных методов, используемых в лабораторной работе;
- Результаты работы программы (в виде файла или в виде скриншота);
- Анализ результатов.

Лабораторная работа №1

Кодирование целых чисел

Порядок выполнения работы

1. Изучить теоретический материал гл. 2.
2. Реализовать построение таблиц кодовых слов для рассмотренных кодов целых чисел: кода класса Fixed+Variable и кодов класса Variable +Variable (гамма-кода Элиаса и омега-кода Элиаса) и заполнить следующую таблицу.

Число	Кодовое слово		
	Fixed+Variable	γ-код Элиаса	ω-код Элиаса
0			
1			
2			
…			

3. Запрограммировать процедуру кодирования методом длин серий.
4. Создать файл (не менее 1 Кб), содержащий последовательность из нулей и единиц, причем $P(0) >> P(1)$. Сравнить степени сжатия этого файла методом длин серий при использовании трех кодов целых чисел (Fixed+Variable, γ-код Элиаса, ω-код Элиаса).
5. Коэффициент сжатия определять как процентное отношение длины закодированного файла к длине исходного файла.
6. Результаты оформить в виде таблицы

Размер файла	Коэффициент сжатия файла		
	Fixed+Variable	γ-код Элиаса	ω-код Элиаса

Контрольные вопросы

1. В чем состоит основная идея кодирования целых чисел?
2. Чем отличаются коды Fixed+Variable и Variable+Variable?
3. Как формируются кодовые слова гамма-кода Элиаса?
4. Как строится омега-код Элиаса?
5. В чем заключается кодирование длин серий?

Лабораторная работа №2

Оптимальный код Хаффмана

Порядок выполнения работы

1. Изучить теоретический материал гл. 3 и гл.4.
2. Реализовать процедуру построения оптимального кода Хаффмана.
3. Построить код Хаффмана для текста на английском языке, использовать файл не менее 1 Кб. Распечатать полученную кодовую таблицу в виде:

Символ	Вероятность	Кодовое слово	Длина кодового слова

4. Проверить выполнение неравенства Крафта-МакМиллана для полученного кода
5. Вычислить энтропию исходного файла и сравнить со средней длиной кодового слова.

Контрольные вопросы

1. Какой код называется разделимым? Префиксным?
2. В чем заключается теорема Крафта? Теорема МакМиллана?
3. Что такое энтропия дискретного вероятностного источника?
4. Какова основная характеристика неравномерного кода?
5. Что такое избыточность кода?
6. Почему код Хаффмана называется оптимальным?

Лабораторная работа №3

Почти оптимальное алфавитное кодирование

Порядок выполнения работы

1. Изучить теоретический материал гл. 5
2. Реализовать процедуры построения кодов Шеннона, Фано и Гилберта-Мура.
3. Построить коды Шеннона, Фано и Гилберта-Мура для текста на английском языке (использовать файл не менее 1 Кб). Распечатать полученные кодовые таблицы в виде:

Символ	Вероятность	Кодовое слово	Длина кодового слова

4. Сравнить средние длины кодового слова с энтропией исходного файла для всех построенных статических кодов. Полученные результаты оформить в виде таблицы:

Энтропия исходного текста	Средняя длина кодового слова			
	Код Хаффмена	Код Шеннона	Код Фано	Код Гилберта-Мура

Контрольные вопросы

1. Почему коды Шеннона, Фано и Гилберта-Мура считаются почти оптимальными?
2. На сколько средняя длина кодового слова превосходит энтропию источника для кода Шеннона? Для кода Фано?
3. Какой код называется алфавитным?
4. Являются ли алфавитными коды Шеннона, Фано и Гилберта-Мура?
5. Почему код Гилберта-Мура имеет наибольшую избыточность среди рассмотренных почти оптимальных кодов?

Лабораторная работа №4

Арифметическое кодирование

Порядок выполнения работы

1. Изучить теоретический материал гл. 6
2. Закодировать арифметическим кодом текст на английском языке, использовать файл не менее 1 Кб.
3. Вычислить коэффициент сжатия данных как процентное отношение длины закодированного файла к длине исходного файла.
4. Определить зависимость коэффициента сжатия данных от длины блока при арифметическом кодировании.
5. Декодировать файл, закодированный арифметическим кодом, и сравнить полученный файл с исходным текстом на английском языке.

Контрольные вопросы

1. Как зависит средняя длина кодового слова от длины блока при арифметическом кодировании?
2. Как формируется кодовое слово для последовательности символов при арифметическом кодировании?
3. Сколько двоичных разрядов необходимо взять для арифметического кодирования k исходных символов, чтобы декодирование было возможным?
4. Как осуществляется арифметическое декодирование?
5. Какие проблемы возникают при реализации арифметического кодирования?

Лабораторная работа №5

Адаптивное кодирование

Порядок выполнения работы

1. Изучить теоретический материал гл. 7
2. Закодировать текст на английском языке (использовать файл не менее 1 Кб) с помощью адаптивного кода Хаффмана, кода «Стопка книг», интервального и частотного кодов.
3. Вычислить коэффициенты сжатия данных как процентное отношение длины закодированного файла к длине исходного файла.
4. Сравнить полученные коэффициенты сжатия данных, построить таблицу вида:

Размер исходного файла	Коэффициент сжатия данных			
	Адаптивный код Хаффмана	Код «Стопка книг»	Интервальный код	Частотный код

Контрольные вопросы

1. Как адаптивные методы учитывают изменение статистики исходных данных?
2. В чем заключается адаптивный код Хаффмана?
3. Какова основная идея кода «Стопка книг»?
4. За счет чего достигается сжатие данных при кодировании интервальным кодом?
5. Чем отличается частотный код от кода Гилберта-Мура?

Лабораторная работа №6

Словарные коды

Порядок выполнения работы

1. Изучить теоретический материал гл. 8
2. Закодировать словарным кодом с использованием адаптивного словаря текст на английском языке, текст на русском языке и текст программы на языке С (использовать файлы не менее 1 Кб).
3. Вычислить коэффициенты сжатия данных как процентное отношение длины закодированного файла к длине исходного файла, построить таблицу вида:

Размер исходного файла	Коэффициент сжатия данных		
	Текст на английском языке	Текст на русском языке	Текст программы на языке С

4. Декодировать файлы, закодированные словарным кодом, и сравнить полученные файлы с исходными текстами.

Контрольные вопросы

1. Какова общая схема кодирования, используемая в LZ-методах?
2. Чем отличаются друг от друга алгоритмы класса LZ?
3. В чем заключается метод кодирования с использованием скользящего окна?
4. Как используется в LZ-кодах адаптивный словарь?
5. Как осуществляется декодирование в методах с адаптивным словарем?

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы. - М.: Издательский дом "Вильямс", 2000. - 384 с.
2. Березин Б.И., Березин С.Б. Начальный курс С и С++. М: «Диалог-МИФИ», 2001
3. Ватолин Д., Ратушняк А., Смирнов М., Юкин В. Методы сжатия данных. – М.: «Диалог-МИФИ», 2002.
4. Галлагер Р. Теория информации и надежная связь. - М.: Советское радио, 1974. - 719 с.
5. Кричевский Р.Е. Сжатие и поиск информации. - М.: Радио и связь, 1989. - 168 с.
6. Марченко А.И., Марченко Л.А. Программирование в среде Turbo PASCAL 7.0. Базовый курс. Киев: «ВЕК+», 2003.

ПРИЛОЖЕНИЕ А

Псевдокод для записи алгоритмов

Для записи алгоритма будем использовать специальный язык – псевдокод. Алгоритм на псевдокоде записывается на естественном языке с использованием двух конструкций: ветвления и повтора. В круглых скобках будем писать комментарии. В треугольных скобках будем описывать действия, алгоритм выполнения которых не требует детализации, например, <обнулить массив>.

: = Операция присваивания значений.

⟷ Операция обмена значениями.

Конструкции ветвления.

1. IF (условие) Если выполняется условие,
<действие> то выполнить действие
FI FI указывает на конец этих действий.

2. IF (условие)
<действия 1>
ELSE <действия 2> Действия 2 выполняются,
FI если неверно условие.

3. IF (условие1)
<действия1>
ELSEIF (условие2) Действия 2 выполняются,
<действия2> если неверно условие1 и верно условие 2
…FI

Конструкции повтора.

1. Цикл с предусловием.
DO (условие) Действия повторяются
<действия> пока условие истинно.
OD OD указывает на конец цикла.

2. Цикл с постусловием.
DO <действия>
OD (условие выполнения)

3. Цикл с параметром.
DO (i=1, 2, ... n) Действия выполняются для значений
<действия> параметра из списка
OD

4. Бесконечный цикл.
DO
<действия>
OD

5. Принудительный выход из цикла.
DO
...IF (условие) OD Если условие истинно, то выйти из цикла.
OD

Елена Викторовна Курапова
Елена Павловна Мачикина

ОСНОВНЫЕ МЕТОДЫ КОДИРОВАНИЯ ДАННЫХ

Методические указания

Редактор Фионов А. Н.
Корректор:.....

Подписано в печать.....
Формат бумаги 62 х 84/16, отпечатано на ризографе, шрифт № 10,
изд. л.....,заказ №.....,тираж – экз., СибГУТИ.
630102, г. Новосибирск, ул. Кирова, 86.