**Тема 6. Сжатие данных**

### **Частотно-зависимое кодирование. Метод Шеннона-Фано**

* 1. **Словарное сжатие**

### **Частотно-зависимое кодирование. Метод Шеннона-Фано**

Следующими важнейшими задачами, рассматриваемыми в теории  кодирования, являются принципы наиболее экономичного кодирования  информации для уменьшения объема, а также разработка приемов,  обеспечивающих надежность передачи информации по каналам связи,  защита от помех.

Объём информации имеет большое значение для хранения и передачи  информации. Двоичное представление информации обычно имеет  некоторую избыточность, поэтому были разработаны специальные методы  (алгоритмы) сжатия информации, с помощью которых можно уменьшить

объем.

Существуют как универсальные алгоритмы, которые рассматривают  информацию, как набор битов, так и специализированные, которые  предназначенные для сжатия информации определённого типа, например,  графической. Цель сжатия - уменьшение количества бит, необходимых для  хранения или передачи данных. Методы сжатия можно разделить на  сжатие без потерь и с потерями. Рассмотрим некоторые общие методы без  потерь:

• частотно-зависимое (frequency-dependent encoding) кодирование;

• систему сжатия Лемпеля-Зива (Lempel-Ziv encoding), кодирование с адаптивным словарем (dictionary encoding, adaptive dictionary encoding).

•

До сих пор мы работали с алфавитным равномерным кодированием, в  этом случае двоичный код строится кодовыми словами равной длины, такое  кодирование используется при представлении информации в компьютере.  Для более экономичного представления можно использовать алфавитное  неравномерное кодирование, при этом знаки первичного алфавита  кодируются кодовыми словами различной длины. При таком кодировании  возникает проблема однозначности декодирования, различимости кодовых  слов. Один из способов для различимости кодовых слов – использование  префиксных кодов.

**Префиксные коды,** это коды, которые удовлетворяют условию Фано: неравномерный код может быть однозначно декодирован, если никакое  кодовое слово не совпадает с началом (префиксом) какого-либо другого  кодового слова.

Относительная частота, т.е. вероятность появления различных букв в  тексте (или сообщении) различна и при частотно-зависимом кодировании

длина кодового слова символа первичного алфавита обратно  пропорциональна частоте появления этого символа в тексте, т.е. символы,  которые чаще появляются в сообщении, кодируются словами меньшей  длины. Мы рассмотрим два таких кода.

Префисный код Шеннона-Фано был предложен в 1948-1949 гг.  независимо Р. Фано и К. Шенноном [2]. Кодирование осуществляется  следующим образом:

• символы в таблице располагаются в порядке убывания вероятностей (относительных частот) появления в тексте;

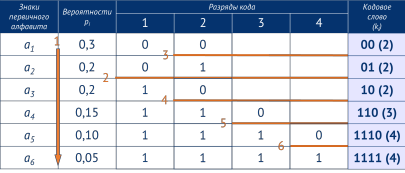
• знаки делятся на две группы таким образом, чтобы суммы  вероятностей в каждой из них были бы более близкими к равенству,  первой группе в соответствие ставится знак – 0, второй – 1;

• продолжаем деление каждой из групп на подгруппы по этой же схеме,  пока это возможно.

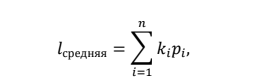
Рассмотрим алгоритм кодирования на примере. Первым шагом  расположим символы в таблице в порядке убывания вероятностей (*Рис. 1*).  Вторым шагом разделим знаки на две группы таким образом, чтобы суммы  вероятностей в каждой из них были бы более близкими к равенству. В  нашем случае – это группа из символов c суммой вероятностей

0,3 + 0,2 = 0,5 и символы с суммой вероятностей 0,2 + 0,15 + 0,1 + 0,05 = 0,5. Первой группе поставим в соответствие 0, второй 1.  Третьим шагом делим первую группу, в данном случае возможен  только один вариант.

Четвертым шагом делим вторую группу это символ a3 с вероятностью  0,2 и символы a4, a5, a6 с суммой вероятностей 0,15+0,1+0,05=0,3.

*Рис. 1*

Средняя длина кода рассчитывается по следующей формуле:



где – длина кодового слова с вероятностью появления в тексте 

Для нашего примера средняя длина:

При равномерном алфавитном двоичном кодировании (не  рассматриваем конкретное представление в компьютере), длина кодового  слова получается из соотношения 2k ≥ N, где *N* – количество символов в  первичном алфавите, *k* – наименьшее натуральное удовлетворяющее  неравенству, длина кодового слова. В случае равномерного кодирования  все длины в рассматриваемом примере равны 3, так как 23 ≥ 6 и средняя  длина – 3:

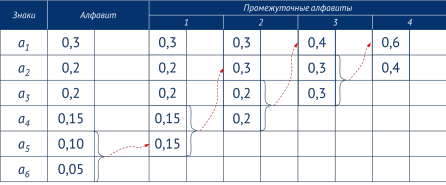
Lсредняя= 0,3 ∗ 3 + 0,2 ∗ 3 + 0,2 ∗ 3 + 0,15 ∗ 3 + 0,1 ∗ 3 + 0,05 ∗ 3 = 3 Префиксный код Хаффмана был предложен Дэвидом Хаффманом в  1952 году. Кодирование осуществляется следующим образом: • символы в таблице располагаются в порядке убывания вероятностей (относительных частот) появления в тексте;

• создается новый вспомогательный алфавит объединением двух  знаков с наименьшей вероятностью и заменой их одним знаком с вероятностью равной сумме, затем знаки переупорядочиваются по  убыванию;

• аналогичным образом продолжается создание новых алфавитов, пока  в алфавите не останется 2 знака;

• в обратном направлении проводится кодирование.

Рассмотрим кодирование по схеме на том же алфавите. Понадобится  четыре шага до получения алфавита из двух знаков (см. Рис. 2). Для  упорядочивания промежуточного алфавита, в случае равенства  вероятностей, новый символ добавляем последним.

*Рис. 2*

Кодируем в обратном направлении, последним символам ставим в  соответствие 0 – символу с большей вероятностью и 1 – с меньшей,  переходя к предыдущему алфавиту, к кодовому слову для объединенных  символов добавляем справа 0 и 1 сверху вниз (Рис. 3).

*Рис. 3*

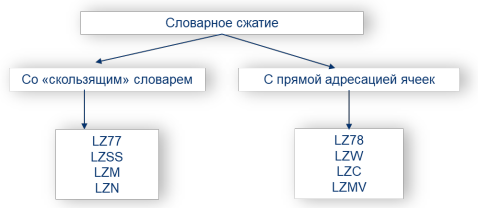
Рассчитаем среднюю длину кода:

Lсредняя = 0,3 ∗ 2 + 0,2 ∗ 2 + 0,2 ∗ 2 + 0,15 ∗ 3 + 0,1 ∗ 4 + 0,05 ∗ 4 = 2,45 Коды, получаемые по схеме Фано-Шеннона и Хаффмена,  удовлетворяют условию Фано по построению.

Не смотря на то, что в предложенном примере средние длины кодов  получились равными, в общем случае, код Хаффмана является самым  экономным, т.е. ни для какого метода алфавитного кодирования средняя  длина кода не может оказаться меньше, чем у кода Хаффмана [2].

***Словарное сжатие***

Словарные методы обеспечивают сжатие в основном за счет замены  повторяющихся цепочек знаков более короткими ссылками на ранее  встретившиеся цепочки, которые хранятся в специальном «словаре» [3].



Алгоритм LZ77 был опубликован в 1977 г. Разработан израильскими  математиками Якобом Зивом (Ziv) и Авраамом Лемпелом (Lempel). В  дальнейшем на основе этого алгоритма были созданы другие алгоритмы,  которые объединили в группу алгоритмов со «скользящим» словарем (Рис.  4).

Данные алгоритмы используют в архиваторах и различных форматов,  некоторые из них приведены в Таблица 1.

*Таблица 1*

|  |  |
| --- | --- |
| Программы | Алгоритмы кодирования |
| Zip (zip) | LZW, LZ77, Хаффмана |
| Gzip (gz) | LZ77, Хаффмана |
| 7-zip | LZMA, Хаффмана |
| Форматы | Алгоритмы кодирования |
| gif | LZW |
| png | LZ77, Хаффмана |
| tiff | LZW |

Одним из этой группы является алгоритм LZSS разработанный в 1982  году Лемпелем, Зивом, Сторером и Шаманским. LZSS использует часть уже  просмотренной части сообщения как словарь. Чтобы добиться сжатия, он  пытается заменить очередной символ или фрагмент сообщения кодом,

который указывает на такой же символ или фрагмент, хранящийся в словаре, если он есть.

LZSS использует «скользящее» по сообщению окно, разделенное на  две неравные части. Первая, большая по размеру (словарь), включает уже  просмотренную часть сообщения.

Вторая, намного меньшая, является буфером, содержащим еще  незакодированные символы входного потока. Поиск повторяющихся  цепочек ведется только внутри рамки словаря. Код начинается с  однобитного префикса для отличия кода от символа. Если префикс x равен  1, то далее идет код, который состоит из пары [d,l]: смещение от начала  словаря и длина фрагмента. Окно сдвигается ровно на длину найденной  подстроки или на 1, если не найдено вхождение подстроки из буфера в  словарь.

Закодируем по алгоритму LZSS следующую строку (Рис. 5): https://lh4.googleusercontent.com/kwFIVxTr8WaRWWbSEKePZP_1MLwgIMJIqI1iAE9Naf6PaS-MBT8N-3u1_aotFrTckHnqUjDsA4Th6qHoBpwBzqa_qPeVO_tetoOdxCk0PtMabEvvd4yTlwLeW_wmOcAokrlEgIePioj8Y4MVhxT55Yk

*Рис. 5*

При сжатии с LZSS используем словарь из 8 знаков и буфер из 5  знаков. В этом случае максимальное смещение от начала словаря – 7,  следовательно, для кодирования смещения необходимо 3 бита,  максимальная длина цепочки не может превышать размер буфера,  следовательно, 5, для кодирования потребуется 3 бита (22<5<23), для  кодирования битового префикса – 1 бит. Длина кода цепочки x[d,l]: 1+3+3=7  битов.

https://lh3.googleusercontent.com/B4WvPih-aTc3Q0gZLAiqAtv63d6FpE63Hv_iNeWinU72gkikIOrzAofTgm4CfdTCgALUgkjzyANd0wmI97zXtVZ9pD_SWtXiHZyFyVU8kQKLhDzcLuVQbMHOiIPfn41v1cNCMx-l-4J_69gH3c0iVnA

*Рис. 6*

После седьмого шага выполнения алгоритма сжатия положение  буфера и словаря будут иметь положения, как показано на рисунке Рис. 6.  Код сообщения: 0к+0о+0с+0а+0р+0ь+0\_, а его длина в битах равна  9+9+9+9+9+9+9=63.

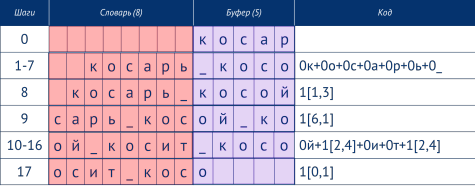
На восьмом шаге в словаре возникает цепочка, как и в буфере: кос  (Рис. 7). К коду добавляется 1[1,3], и длина сообщения становится равна  9+9+9+9+9+9+9+7=70.

https://lh3.googleusercontent.com/dT4X4T3oY-xElNyilPqkdWQdjdrg2D9p5QllG9zf4g6Cl_CXzOH1UzOVN2a8A1W2gUoCSw-DN6ZfSU4jw7R5dTALlXGhHpG58yzMW6NSG9GYJCaye3N_KldoVJ9rEqaNmMRBKltf2s7SxOUOZs1XL6k

*Рис. 7*

После обработки всего сообщения (Рис. 8) получаем код:

0к+0о+0с+0а+0р+0ь+0\_+1[1,3]+1[6,1]+0й+1[2,4]+0и+0т+1[2,4]+1[0,1] Длина кода в битах равна 9+9+9+9+9+9+9+7+7+9+7+9+9+7+7=125.

*Рис. 8*

Сообщение состоит из 23 символов, при представлении в компьютере,  если символ кодируется 8 битами, например, с использованием ASCII, то  длина сообщения в битах равна 23 \* 8 = 184. Сжатое сообщение – 125  битов. Длина сжатого кода составляет 68% от 8-битного представления.

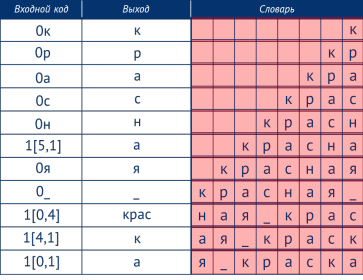
Пример декодирования следующего закодированного сообщения: 0к+0р+0а+0с+0н+1[5,1]+0я+0\_+1[0,4]+1[4,1]+1[0,1] показано на рисунке Рис. 9.

LZ77 и LZSS обладают следующими очевидными недостатками: • работают медленно;

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

• длина подстроки, которую можно закодировать, ограничена размером  буфера.

В 1978 г. авторами алгоритма LZ77 был разработан алгоритм LZ78,  лишенный недостатков предыдущих: не использует «скользящее» окно,  хранит словарь из уже просмотренных фраз, в словаре нет одинаковых  фраз, каждый код состоит из номера (индекса) фразы в словаре или  префикса 0 и символа, от размера словаря зависит размер получаемых  кодов. А в 1984 г. Уэлчем (Welch) путем модификации LZ78 был  создан алгоритм LZW. Алгоритм запатентован.

*Рис. 9*

Алгоритм динамически создает таблицу для преобразования символов.  Таблица инициализируется всеми односимвольными фразами (обычно 256  символами ASCII), кроме того для очистки и кода конца информации  зарезервированы значения 256 и 257.

При заполнении таблицы определенными последовательностям  символов ставится в соответствие код фиксированной длины, этот код  является индексом создаваемой таблицы. Длина кода от 9 до 12 бит, и  тогда под коды выделяют значения от 258 до 4095.

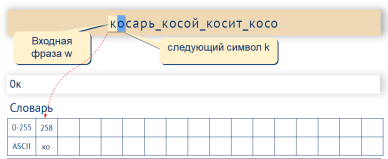
Алгоритм кодирования:

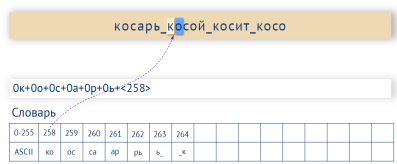
1 шаг. Инициализация входной фразы w первым символом сообщения. 2 шаг. Считывается очередной символ K из кодируемого сообщения. 3 шаг. Если конец сообщения, то выдать код для w и завершить. 4 шаг. Если фраза wK уже есть в словаре, то

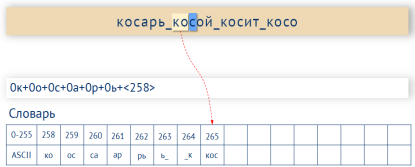
присвоить входной фразе значение wK и перейти к шагу 2, иначе выдать  код w, добавить wK в словарь, присвоить входной фразе w значение K,  перейти к шагу 2.

Рассмотрим на примере алгоритм LZW, закодируем фразу на рисунке  Рис. 5. Размер словаря – 500 фраз, код – 9 бит.

Первый проход изображен на рисунке Рис. 10.

*Рис. 10*

*Рис. 11*

*Рис. 12*

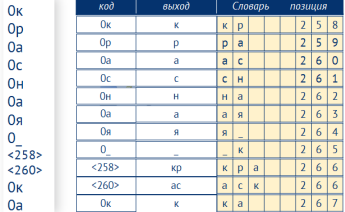
Восьмой и девятый проход – на рисунках Рис. 11 и Рис. 12. Мы видим,  что во входной фразе хранится w = «k», считывается K = «о» (Рис. 11).  Фраза «ко» есть в словаре под индексом 256, присваиваем входной фразе  w = «ко» и считываем следующий символ K = «с», выдаем код 256 (Рис. 12),  присваиваем w = «с» и переходим ко второму шагу алгоритма.

Все шаги изображены на Рис. 13. В результате работы алгоритма  получаем код:

0к+0о+0с+0а+0p+0ь+0\_+<258>+0с+0о+0й+<264>+<259>+0и+0т+<269>+<266>

Сообщение, если символ кодируется 8 битами (например, ASCII) – 184  бита. Сжатое сообщение – 153 бита. Длина сжатого кода составляет 83% от  8-битного представления.

*Рис. 13*

**

*Рис. 14*

Пример декодирования сообщения:

0к+0р+0а+0с+0н+0а+0я+0\_+<258>+ +<260>+0к+0а

изображено на рисунке Рис. 14.