Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Сжатие/распаковка данных методом Лемпеля-Зива**

Студент: Дрозд А. И.

ФИТ 3 курс 2 группа

Преподаватель:

Нистюк Ольга Александровна

Минск 2024

**Цель**: приобретение практических навыков использования статистических методов Шеннона − Фано и Хаффмана (ShannonFano and Huffman coding) для сжатия/распаковки данных.

Задачи:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов сжатия/распаковки (архивации/ разархивации) данных на основе метода Лемпеля-Зива.
2. Разработать приложение для реализации метода Лемпеля-Зива.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# **Теоретические сведения**

Суть метода LZ77 (как и последующих его модификаций) состоит в следующем: упаковщик постоянно хранит некоторое количество последних обработанных символов в буфере. По мере обработки входного потока вновь поступившие символы попадают в конец буфера, сдвигая предшествующие символы и вытесняя самые старые. Размеры этого буфера, называемого также скользящим словарем (англ. sliding dictionary), варьируются в разных реализациях систем сжатия. Скользящее окно имеет длину n, т. е. в него помещается n символов, и состоит из двух частей:

• последовательности длины n1 = n − n2 уже закодированных символов (словарь);

• упреждающего буфера (буфера предварительного просмотра, lookahead) длиной n2 – буфера кодирования.

Пусть к текущему моменту времени закодировано t символов: S1, S2, ..., St. Тогда словарем будут являться n1 предшествующих символов: St − (n1 − 1), St − (n1 − 1)+1, …, St.

В буфере находятся ожидающие кодирования (сжатия) символы St+1, St+2, …, St+n2. Если n2 ≥ t, то словарем будет являться вся уже обработанная часть входной последовательности.

Нужно найти самое длинное совпадение между строкой буфера кодирования, начинающейся с символа St + 1, и всеми фразами словаря.

Эти фразы могут начинаться с любого символа St − (n1 − 1), St − (n1 − 1) + 1, …, St, выходить за пределы словаря, вторгаясь в область буфера, но должны лежать в окне. Буфер не может сравниваться сам с собой. Длина совпадения не должна превышать размера буфера. Полученная в результате поиска фраза St − (р − 1), St − (р − 1) + 1, St − (р − 1) + (q − 1) кодируется с помощью двух чисел:

1) смещения (англ. offset) от начала буфера p;

2) длины соответствия, или совпадения (англ. match length) q.

Ссылки (p и q − указатели) однозначно определяют фразу. Дополнительно в выходной поток записывается символ s, следующий за совпавшей строкой буфера.

Длина кодовой комбинации (триады – p, q, s) на каждом шаге определяется соотношением представленном на рисунке 1.1:



Рисунок 1.1 – Длина кодовой комбинации

После каждого шага окно смещается на q + 1 символов вправо и осуществляется переход к новому циклу кодирования. Величина сдвига объясняется тем, что мы реально закодировали именно q + 1 символов: q – с помощью указателя и 1 − с помощью тривиального копирования.

Передача одного символа в явном виде (s) позволяет разрешить проблему обработки еще ни разу не встречавшихся символов, но существенно увеличивает размер сжатого блока.

Пример 1. Используется алфавит А = {0,1,2,3}, N = 4, принимаем: длина словаря n1 = 15, длина буфера данных (кодирования) n2 = 13; для обозначения p и q используется четверичная система счисления. Тогда формула для вычисления длины кодовой комбинации изображена на рисунке 1.2.



Рисунок 1.2 – Формула вычисления кодовой комбинации

Входной поток S = 2000302013020130313031303130313333333. Вспомним соответствие между числами в десятичной и четверичной системах счисления: 010 = 004, 110 = 014, 210 = 024, 310 = 034, 410 = 104, 510 = 114 и т. д.

Прямое преобразование. Шаг 1: состояние буфера отображает рис. 1.3.

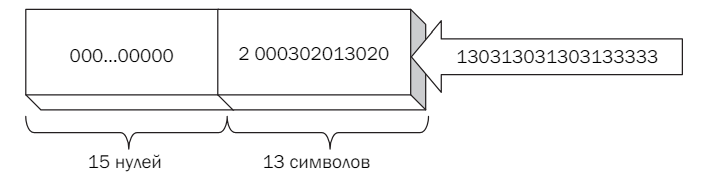


Рисунок 1.3 – Первый шаг

Анализируем 1-й символ в буфере кодирования на предмет соответствия (наличия) такого же символа или нескольких символов в словаре. Таких символов нет, следовательно, p1 = 0. Длина повторения q1 = 0. Таким образом, имеем следующую триаду: (p1, q1, s1) = (00 00 2)4 (нижний индекс справа от знака равенства означает основание системы счисления).

Шаг 2: состояние буфера отображает рис. 1.4.

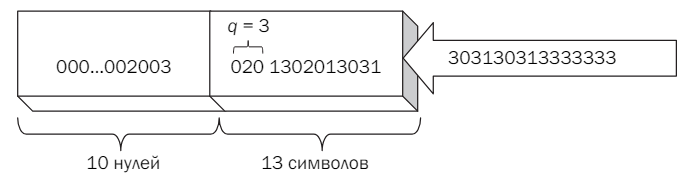


Рисунок 1.4 – Второй шаг

Находим повторение (000), длина этого повторения q2 = 3. Поскольку в словаре нулевые символы записываются с 1-й по 14-ю позицию (для упрощения индексация ведется слева направо), то индексом р2 (началом повторения) может быть выбрано любое число от 1 до 12: 1 ≤ р2 ≤ 12; выбираем р2 = 6. Итак, получим следующую триаду: (p2, q2, s2) = (6, 3, 3) = (12 03 3)4.

Передвигаем сообщение в окне на q2 + 1 = 3 + 1 = 4 позиции.

Шаг 3: состояние буфера отображает рис. 1.5.

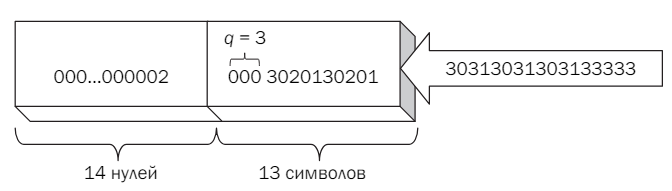


Рисунок 1.5 – Третий шаг

В этой ситуации наиболее длинный повтор – 020, т. е. q3 = 3, р3 = 10. Передвигаем сообщение в окне на q3 + 1 = 4 позиции. И получаем триаду: (p3, q3, s3) = (10, 3, 1) = (22 03 1)4.

Шаг 4: состояние буфера отображает рис. 1.6.

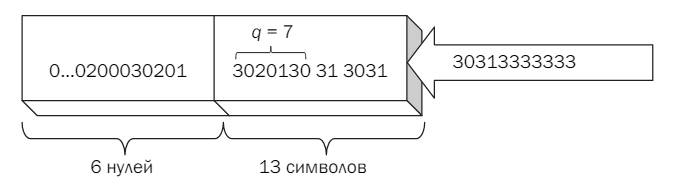


Рисунок 1.6 – Четвертый шаг

В этой ситуации наиболее длинный повтор – 3020130. Значит, q4 = 7, р4 = 11. Передвигаемся на q4 + 1 = 8 и получаем триаду: (p4, q4, s4) = (11, 7, 3) = (23 13 3)4. И т. д.

Обратное преобразование. Имеем на входе «запакованное» сообщение: 00002 12033 22031 23133 30301 02013 32103… .

В исходном состоянии в окне (используется одно скользящее окно) записывают 15 нулей (такой выбрана длина окна). Результат анализа – символы исходного сообщения на основе выбранного алфавита.

Шаг 1: анализируем первые 5 символов (вспомним, l(с) = 5): 00002, следовательно, q1 = 0 и р1 = 0. В результате в младший (крайний справа) разряд окна записывается лишь символ 3 (s1 = 3).

Шаг 2: анализируем следующие 5 символов: 12033, т. е. р2 = 6, q2 = 3. Поскольку в окне содержатся только нули, за исключением последнего разряда (15, индексация ведется как и в случае прямого преобразования), то мы можем сделать вывод, что наш повтор – это следующие три символа: 000. К ним дописывается еще последний символ из анализируемой триады (s2 = 3). Таким образом, после двух шагов начальные символы распакованного сообщения будут такими: 20003. В остальных (начальных: с первого по десятый) разрядах окна будут нули.

Шаг 3: анализируем третью триаду сжатого сообщения: 22031. Следовательно, р3 = 10, q3 = 3. Разряды с десятого (р3 = 10) по двенадцатый (q3 = 3) окна содержат символы 020. Эти разряды допишутся справа к существующим символам окна и дополнительно к ним допишется символ 1 (s3 = 1). Таким образом, в окне после этого будет записано: 000000200030201. И т. д.

Обратное преобразование. В процессе восстановления используются только набор триад (кодов) и буфер. Принимаем размер буфера и его начальное содержимое такое же, как и при сжатии. Первый шаг изображён на рисунке 1.7.

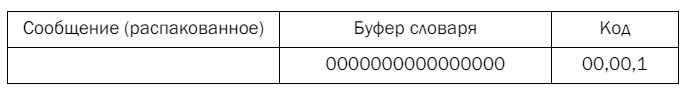


Рисунок 1.7 – Первый шаг обратного преобразования

Анализируется триада (00,00,1). Начало повторяющейся последовательности (p) и ее длина (q) равны 0 – повторений нет. Добавляем к словарю символ «1» и сдвигаем окно словаря на q + 1 позицию вправо. Второй шаг изображён на рисунке 1.8.

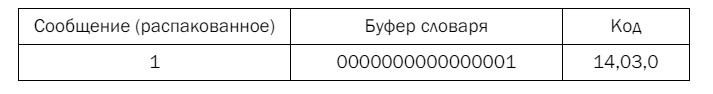


Рисунок 1.8 – Второй шаг обратного преобразования

Анализируется триада (14,03,0). Повторяющаяся последовательность (p) начинается с четырнадцатой позиции и длиной (q), равной 3. Добавляем к словарю повторяющуюся последовательность «001» и символ «0». Сдвигаем окно словаря на q + 1 позицию вправо. Третий шаг изображён на рисунке 1.9.

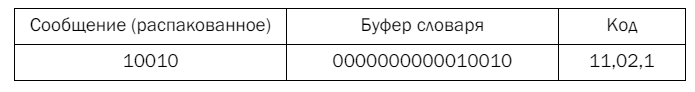


Рисунок 1.9 – Третий шаг обратного преобразования

Анализируется триада (11,02,1). Повторяющаяся последовательность (p) начинается с одиннадцатой позиции и длиной (q), равной 2. Добавляем к словарю повторяющуюся последовательность «01» и символ «1». Сдвигаем окно словаря на q + 1 позицию вправо. И т. д.

Как видно из приведенных примеров, алгоритм LZ77 обладает следующими очевидными недостатками:

1) невозможностью кодирования подстрок, отстоящих одна от другой на расстояние, большее длины словаря;

2) длина кодируемой подстроки ограничена размером буфера. Однако если увеличивать размер словаря, то это приведет к увеличению времени преобразования и длины кодовых последовательностей.

В 1978 г. создателями алгоритма LZ77 был разработан усовершенствованный алгоритм: LZ78, лишенный названных недостатков. LZ78 не использует «скользящее» окно. Он хранит словарь из уже просмотренных подстрок. При старте алгоритма этот словарь содержит только одну пустую строку (строку длины нуль). Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта подстрока перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу.

**Практическое задание**

Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n1, n2).

С помощью приложения выполнить прямое и обратное преобразования произвольного текста длиной несколько килобайт. Формат представления параметров p и q выбрать по указанию преподавателя.

Изменяя размеры окон, оценить скорость и эффективность выполнения операций сжатия/ распаковки.

Первоначально приложению необходимо передать исходное сообщение, к которому будет применен метод сжатия Лемпеля-Зива. Также, помимо сообщения, требуется ввести два параметра – размер словаря и размер буфера, на основе которых и будет применен метод. Данный процесс показан на рисунке 2.1.

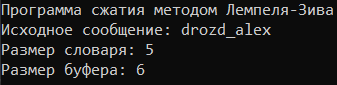


Рисунок 2.1 – Входные параметры

Далее к исходному сообщению будет применен метод Лемпеля-Зива, суть которого была описана в теоретическом введении. На первом шаге наш словарь пуст, поэтому значения для q и p будут нулевыми. Соответственно итогом первого шага будет триада (0, 0, d). Результат показан на рисунке 2.2.

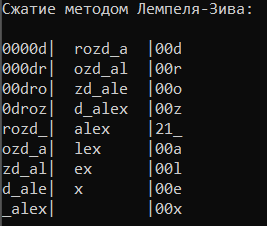


Рисунок 2.2 – Применение метода Лемпеля-Зива

Итогом сжатия является сообщение, состоящее из всех триад, полученных на последовательных шагах сжатия. Итоговое сообщение представлено на рисунке 2.3.



Рисунок 2.3 – Сжатое сообщение

После этого необходимо произвести обратное преобразование (распаковку) сообщения к первоначальному виду. Процесс обратного преобразования был также подробно описан в теоретическом введении. Обратное преобразование показано на рисунке 2.4.

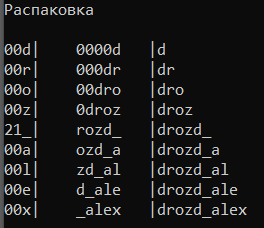


Рисунок 2.4 – Обратное преобразование сообщения

В итоге необходимо рассчитать время, затраченное на прямо и обратное преобразование, а также эффективность этих преобразований при различных входных данных. Результат представлен на рисунке 2.5.

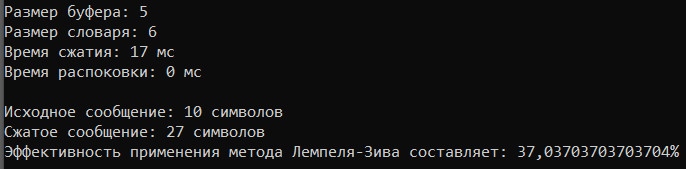


Рисунок 2.5 – Вычисление времени преобразований и их эффективности

На рисунке 2.6 будут показаны еще два результата различных вариаци1 размерности буфера и словаря.

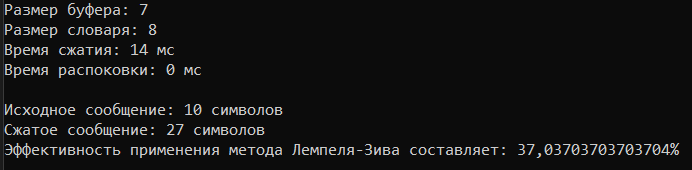
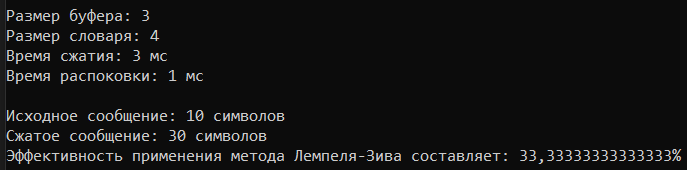


Рисунок 2.6 – Вариации размерности буфера и словаря

Из предоставленной информации можно сделать вывод, что при увеличении размера окон увеличивается время, затрачиваемое на сжатие, а также увеличивается эффективность применения метода Лемпеля-Зива.

**Вывод:**

Алгоритм Лемпеля-Зива (LZ77) – это метод сжатия данных, который оперирует словарем для замены повторяющихся символьных последовательностей. Он работает путем записи ссылок на уже закодированные фрагменты данных вместо их повторного записывания. Основная идея алгоритма заключается в разделении данных на словарь и буфер предварительного просмотра. Словарь содержит закодированные символы, а буфер - еще не закодированные. Постепенно алгоритм обрабатывает данные, ищет совпадения в словаре для символов из буфера и записывает ссылки на найденные фрагменты данных. Это позволяет эффективно сжимать данные за счет замены повторов ссылками на уже известные фрагменты, что уменьшает объем передаваемой информации.