Лабораторная работа № 10
СЖАТИЕ/РАСПАКОВКА ДАННЫХ МЕТОДОМ ЛЕМПЕЛЯ – ЗИВА

Цель: приобретение практических навыков использования метод Лемпеля – Зива (Lempel-Ziv) 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.

10.1. Теоретические сведения

10.1.1. Описание и основные свойства и особенности реализации метода Лемпеля – Зива

В 1977 г. Авраам Лемпель и Якоб Зив выдвинули идею формирования «словаря» общих последовательностей анализируемых (сжимаемых) данных. При этом сжатие данных осуществляется за счет замены записей соответствующими кодами из словаря. Классический алгоритм Лемпеля – Зива – LZ77, названный так по году представления метода, формулируется следующим образом: «*если в проанализированном (сжатом) ранее выходном потоке уже встречалась подобная последовательность байт, причем запись о ее длине и смещении от текущей позиции короче, чем сама эта последовательность, то в выходной файл записывается ссылка (смещение, длина), а не сама последовательность*» (оригинальную статью см. по ссылке [33]).

Известный метод сжатия RLE, который заключается в записи вместо последовательности одинаковых символов одного символа и их количества, является подклассом LZ77.

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 $n_1 = n - n_2$ уже закодированных символов (словарь);
- упреждающего буфера (буфера предварительного просмотра, *lookahead*) длиной n_2 – *буфера кодирования.*

Пусть к текущему моменту времени закодировано t символов: S_1, S_2, ..., S_t. Тогда словарем будут являться n_1 предшествующих символов: $S_{t-(n1-1)}$, $S_{t-(n1-1)+1}$, …, S_t.

В буфере находятся ожидающие кодирования (сжатия) символы S_{t+1}, S_{t+2}, …, S_{t+n2}. Если $n_2 \geq t$, то словарем будет являться вся уже обработанная часть входной последовательности.

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

Эти фразы могут начинаться с любого символа $S_{t-(n1-1)}$, $S_{t-(n1-1)+1}$, …, S_t, 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 $S_{t-(p-1)}$, $S_{t-(p-1)+1}$, $S_{t-(p-1)+(q-1)}$ 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

$$l(c) = \log_N n_1 + \log_N n_2 + 1, \qquad (10.1)$$

где N – мощность алфавита.

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.

10.1.2. Примеры реализации метода

Пример 1. Используется алфавит $A = \{0,1,2,3\}$, $N = 4$, принимаем: длина словаря $n_1 = 15$, длина буфера данных (кодирования) $n_2 = 13$; для обозначения p и q используется четверичная система счисления. Тогда длина кодовой комбинации на каждом шаге составляет

$$l(c) = \log_N n_1 + \log_N n_2 + 1 = 2 + 2 + 1 = 5. \qquad (10.2)$$

Входной поток $S = 2000302013020130313031303130313333333$.

Вспомним соответствие между числами в десятичной и четверичной системах счисления:

$0_{10} = 00_4$, $1_{10} = 01_4$, $2_{10} = 02_4$, $3_{10} = 03_4$, $4_{10} = 10_4$, $5_{10} = 11_4$ и т. д.

Прямое преобразование. Шаг 1: состояние буфера отображает рис. 10.1.

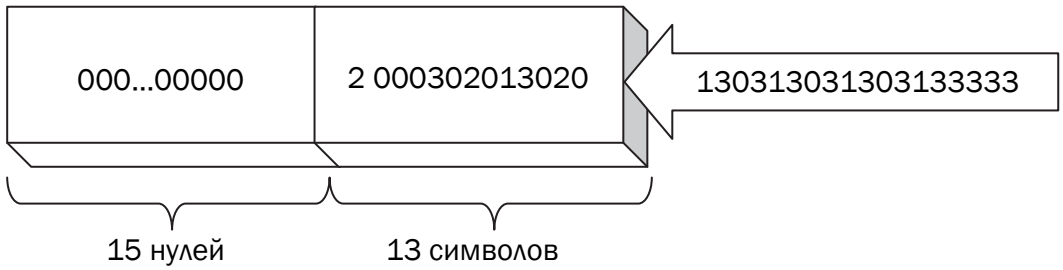

Рис. 10.1. Состояние буфера на первом шаге сжатия сообщения S

Анализируем 1-й символ в буфере кодирования на предмет соответствия (наличия) такого же символа или нескольких символов в словаре. Таких символов нет, следовательно, $p_1 = 0$. Длина повторения $q_1 = 0$. Таким образом, имеем следующую триаду: $(p_1, q_1, s_1) = (00\ 00\ 2)_4$ (нижний индекс справа от знака равенства означает основание системы счисления).

Шаг 2: состояние буфера отображает рис. 10.2.

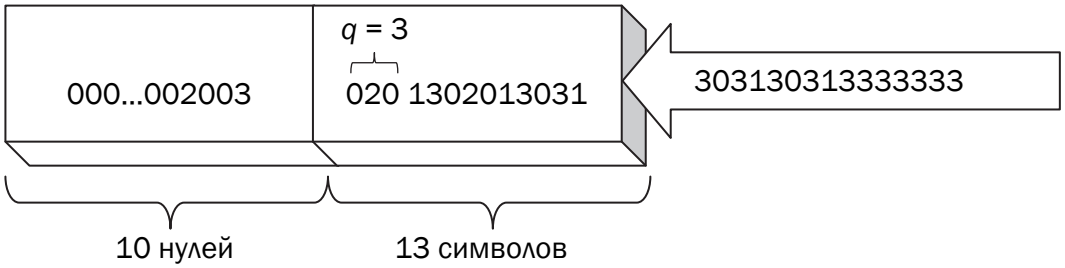

Рис. 10.2. Состояние буфера на втором шаге сжатия сообщения S

Находим повторение (000), длина этого повторения $q_2 = 3$. Поскольку в словаре нулевые символы записываются с 1-й по 14-ю позицию (для упрощения индексация ведется слева направо), то индексом p_2 (началом повторения) может быть выбрано любое число от 1 до 12: $1 \le p_2 \le 12$; выбираем $p_2 = 6$. Итак, получим следующую триаду: $(p_2, q_2, s_2) = (6, 3, 3) = (12\ 03\ 3)_4$.

Передвигаем сообщение в окне на $q_2 + 1 = 3 + 1 = 4$ позиции.

Шаг 3: состояние буфера отображает рис. 10.3.

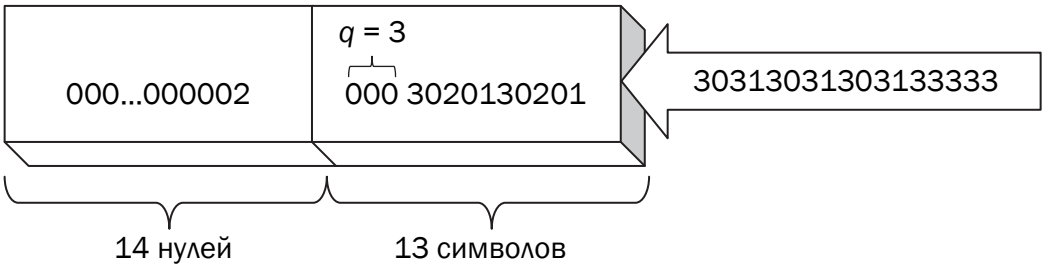

Рис. 10.3. Состояние буфера на третьем шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 020, т. е. $q_3 = 3$, $p_3 = 10$. Передвигаем сообщение в окне на $q_3 + 1 = 4$ позиции. И получаем триаду: $(p_3, q_3, s_3) = (10, 3, 1) = (22\ 03\ 1)_4$.

Шаг 4: состояние буфера отображает рис. 10.4.

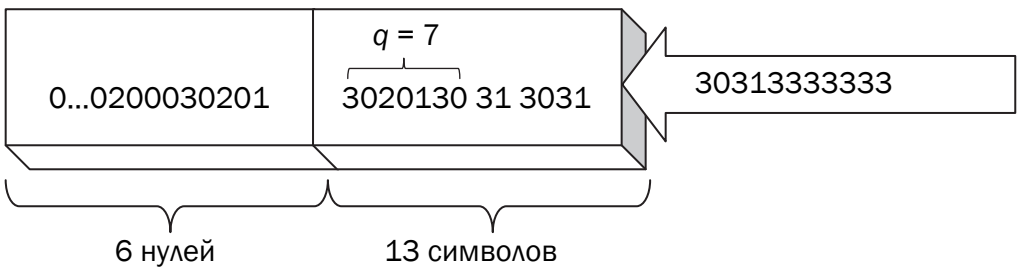

Рис. 10.4. Состояние буфера на четвертом шаге сжатия сообщения S

В этой ситуации наиболее длинный повтор – 3020130. Значит, $q_4 = 7$, $p_4 = 11$. Передвигаемся на $q_4 + 1 = 8$ и получаем триаду: $(p_4, q_4, s_4) = (11, 7, 3) = (23\ 13\ 3)_4$. 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, $l(c) = 5$): 00002, следовательно, $q_1 = 0$ и $p_1 = 0$. В результате в младший (крайний справа) разряд окна записывается лишь символ 3 ($s_1 = 3$).

Шаг 2: анализируем следующие 5 символов: 12033, т. е. $p_2 = 6$, $q_2 = 3$. 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 ($s_2 = 3$). 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, $p_3 = 10$, $q_3 = 3$. Разряды с десятого ($p_3 = 10$) по двенадцатый ($q_3 = 3$) окна содержат символы 020. Эти разряды допишутся справа к существующим символам окна и дополнительно к ним допишется символ 1 ($s_3 = 1$). Таким образом, в окне после этого будет записано: 000000200030201. И т. д.

Пример 2. Рассмотрим преобразование входного потока бинарного типа: S = 100100111011001010110011 0. Размер словаря и буфера данных равны 16.

Прямое преобразование. Буфер словаря изначально заполнен нулями. Буфер данных содержит первые 16 символов исходного сообщения. Так как размер буфера словаря и буфера данных имеет двухзначное число, то при формировании триады необходимо для записи значений p и q отводить 2 знака.

Шаг 1:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000000000	1001001110110010	101100110	00,00,1

Из буфера данных берем первый слева символ «1» и осуществляем поиск данного символа в буфере словаря. Символ в словаре отсутствует. Формируем триаду с началом повторяющей последовательности (p), равным 0, длиной повторяющихся символов (q), равной 0, и следующим за повторяющейся последовательностью символом (c) «1». Сдвигаем окна на $q + 1$ позицию вправо.

Шаг 2:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000000001	0010011101100101	01100110	14,03,0

Из буфера данных берем символ «0» и осуществляем поиск данного символа в буфере словаря. Символ в словаре имеется. Увеличиваем длину анализируемой последовательности на 1. Осуществляем поиск «00» в буфере словаря. Такая последовательность имеется. Увеличиваем длину последовательности на 1. Осуществляем поиск «001» в буфере словаря. Вновь последовательность имеется в буфере словаря. Опять увеличиваем длину последовательности на 1. Осуществляем поиск «0010» в буфере словаря. Такого сочетания символов в словаре не обнаружено. Формируем триаду с началом повторяющей последовательности (p), равным 14, длиной повторяющихся символов (q), равной 3, и следующим за повторяющейся последовательностью символом (c) «0». Сдвигаем окна на $q + 1$ позицию вправо.

Шаг 3:

Буфер словаря	Буфер данных	Сообщение	Код
0000000000010010	0111011001010110	0110	11,02,1

Шаг 4:

Буфер словаря	Буфер данных	Сообщение	Код
0000000010010011	1011001010110011	0	09,02,1

Конечным результатом сжатия будет набор полученных триад (кодов):

(00,00,1)(14,03,0)(11,02,1)(09,02,1)(06,05,1)(09,06,1)(02,01, …).

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.

Шаг 1:

Сообщение (распакованное)	Буфер словаря	Код
	0000000000000000	00,00,1

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.

Шаг 2:

Сообщение (распакованное)	Буфер словаря	Код
1	0000000000000001	14,03,0

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.

Шаг 3:

Сообщение (распакованное)	Буфер словаря	Код
10010	0000000000010010	11,02,1

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.

Алгоритмы LZ-формата используются в тех случаях, когда требуется универсальное сжатие. Например, в известном стандарте модема V.42bis, протоколе передачи данных ZModem, форматах GIF, TIFF, ARC и других прикладных программах. Некоторые алгоритмы данного класса используются в дисковых утилитах сжатия типа DoubleSpace и Stacker, графических форматах типа PNG, а также в универсальных утилитах архивирования и сжатия, включая ZIP, GZIP и LHA.

10.2. Практическое задание

1. Разработать авторское приложение в соответствии с целью лабораторной работы. При этом предусмотреть возможность оперативного изменения размеров окон (n_1, n_2).

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

3. Изменяя размеры окон, оценить скорость и эффективность (используя соотношения на с. 76) выполнения операций сжатия/распаковки.

4. Результаты оформить в виде отчета по установленным правилам.

ВОПРОСЫ ДЛЯ КОНТРОЛЯ И САМОКОНТРОЛЯ

1. К какому классу методов сжатия относится метод LZ и почему?

2. В чем заключается, на Ваш взгляд, оптимизация алгоритма LZ?

3. Какое значение имеет последний символ в сжатом текстовом сообщении методом LZ?

4. Какие модификации метода LZ77 Вам известны? В чем заключается особенность модификаций?

5. Пояснить физический смысл соотношения (10.1).

6. Как будет выглядеть сжатое сообщение, состоящее из символа моноалфавита при определенных значениях n_1, n_2?