

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

**РТУ МИРЭА**

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №8**

**по дисциплине**

**«Структуры и алгоритмы обработки данных»**

Тема. Кодирование и сжатие данных методами без потерь

|  |  |  |
| --- | --- | --- |
| Выполнил студент группы ИКБО-04-22 |  | Кликушин В.И. |
| Принял старший преподаватель |  | Скворцова Л.А. |

Москва 2023

ОГЛАВЛЕНИЕ

[1 ЗАДАНИЕ №1 3](#_Toc153886481)

[1.1 ОПИСАНИЕ АЛГОРИТМА 3](#_Toc153886482)

[1.2 ПРИМЕР КОДИРОВАНИЯ 3](#_Toc153886483)

[1.3 УСОВЕРШЕНСТВОВАННЫЙ МЕТОД RLE 3](#_Toc153886484)

[1.4 КОД АЛГОРИТМА 4](#_Toc153886485)

[2 ЗАДАНИЕ №2 6](#_Toc153886486)

[2.1 МЕТОД LZ77 6](#_Toc153886487)

[2.1.1 Описание алгоритма 6](#_Toc153886488)

[2.1.2 Задание варианта 6](#_Toc153886489)

[2.1.3 Процесс декодирования 6](#_Toc153886490)

[2.2 МЕТОД LZ78 7](#_Toc153886491)

[2.2.1 Описание алгоритма 7](#_Toc153886492)

[2.2.2 Задание варианта 7](#_Toc153886493)

[2.2.3 Процесс декодирования 8](#_Toc153886494)

[3 ЗАДАНИЕ№3 9](#_Toc153886495)

[3.1 МЕТОД ШЕННОНА-ФАНО 9](#_Toc153886496)

[3.1.1 Описание алгоритма 9](#_Toc153886497)

[3.1.2 Задание варианта 9](#_Toc153886498)

[3.1.3 Процесс декодирования 9](#_Toc153886499)

[3.1.4 Таблица формирования кода 10](#_Toc153886500)

[3.1.5 Префиксное дерево 11](#_Toc153886501)

[3.1.6 Коэффициент сжатия 11](#_Toc153886502)

[3.2 МЕТОД ХАФФМАНА 11](#_Toc153886503)

[3.2.1 Описание алгоритма 11](#_Toc153886504)

[3.2.2 Задание варианта 12](#_Toc153886505)

[3.2.3 Процесс декодирования 12](#_Toc153886506)

[3.2.4 Таблицы частот символов 12](#_Toc153886507)

[3.2.5 Префиксное дерево Хаффмана 13](#_Toc153886508)

[3.2.6 Кодирование 14](#_Toc153886509)

[4 ВЫВОДЫ 15](#_Toc153886510)

[5 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ 16](#_Toc153886511)

# 1 ЗАДАНИЕ №1

## 1.1 ОПИСАНИЕ АЛГОРИТМА

Рассматриваемый алгоритм является алгоритмом сжатия с потерями. Суть алгоритма заключается в следующем: группы подряд идущих одинаковых байтов (символов) в поступающей последовательности символов, заменяют на пары вида, где счетчик определяет количество одинаковых символов (значений) в группе. Этот подход к сжатию уменьшает избыточность данных.

## 1.2 ПРИМЕР КОДИРОВАНИЯ

Придумаем текст, в котором есть длинные (в разумных пределах) серии из повторяющихся символов. Выполним его сжатие и рассчитаем коэффициент сжатия. Рассмотрим произвольную строку длиной 43 символа.

Исходный текст: bbbbbbbeeeeeeeeeeeaabbbbbbbbbaaaaaaaaaaeeee

Посчитаем количество символов:

16 символов b; 15 символов e; 12 символов a

Итого найдено 6 серий. Заменим серии на число повторов и сам повторяющийся символ:

7b11e2a9b10a4e

Получилась последовательность из 14 символов. Исходная последовательность состояла из 43 символов. Данные были сжаты в 43/14≈3.07 раза.

## 1.3 УСОВЕРШЕНСТВОВАННЫЙ МЕТОД RLE

Возьмём строку, состоящую из большого количества неповторяющихся символов: XYZXYZXYZXYZVVVVVVZYZYZY

После сжатия методом RLE такая строка будет выглядеть так:

1X1Y1Z1X1Y1Z1X1Y1Z1X1Y1Z6V1Z1Y1Z1Y1Z1Y

Простой метод кодирования RLE будет неэффективен, он увеличит исходную строку в 38/24 = 1.58 раза. Чтобы после сжатия размер данных не увеличивался, алфавит, в котором записаны длины серий, делят на две части (обычно равные). Например, алфавит целых чисел можно разделить на две части: положительные и отрицательные числа. Положительные числа используют для записи количества повторов одного символа, а отрицательные — для записи количества неодинаковых символов, следующих друг за другом.

При таком кодировании новая строка будет иметь следующий вид:

-12XYZXYZXYZXYZ6V-5ZYZYZ

Коэффициент сжатия в этом случае равен 24/24 = 1

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1 – Сравнение двух алгоритмов RLE

## 1.4 КОД АЛГОРИТМА

#include <iostream>

#include <string>

using namespace std;

string RLE(string str)

{

string encoding = "";

int count;

for (int i = 0; str[i]; i++)

{

count = 1;

while (str[i] == str[i + 1]) {

count++, i++;

}

encoding += to\_string(count) + str[i];

}

return encoding;

}

string SUPER\_RLE(string s)

{

string res = "", group = s.substr(0, 1);

int count = 1;

for (int i = 1; i < s.size(); i++) {

if (s[i] == s[i - 1]) {

if (group.size() > 1) {

res += to\_string(count + 1) + group.substr(0, -count - 1);

count = 1;

}

group = "";

count++;

}

else {

if (count >= 2) {

res += to\_string(count) + s[i - 1];

count = 1;

}

else if (count == 1) {

count = -2;

}

else {

count--;

}

group += s[i];

}

}

if (group.size() > 1) {

res += to\_string(count + 1) + group.substr(0, -count - 1);

}

else {

res += to\_string(count) + s[s.size() - 1];

}

return res;

}

int main()

{

system("chcp 1251");

string test = "XYZXYZXYZXYZVVVVVVZYZYZY";

cout << "Исходная строка: " << test << endl;

cout << "Результат кодированяи методов RLE: " << RLE(test) << endl;

cout << "Продвинутый метод RLE: " << SUPER\_RLE(test) << endl;

cout << " Длина исходной строки: " << size(test) << endl;

cout << " Длина при кодировании RLE: " << size(RLE(test)) << endl;

cout << " Длина при кодировании продвинутым RLE: " << size(SUPER\_RLE(test)) << endl;

}

# 2 ЗАДАНИЕ №2

## 2.1 МЕТОД LZ77

### 2.1.1 Описание алгоритма

Идея алгоритма чрезвычайно проста: когда ранее прочитанная подстрока встречается повторно, алгоритм пишет вместо неё ссылку на её предыдущее вхождение. Алгоритм LZ77 (Зив и Лемпель [1977]): на каждом шаге формируется тройка вида (d, l, s), что означает, повторяется подстрока длины l, ранее встречавшаяся - d символов назад, а потом идёт следующий символ текста. Программа кодирования, как бы, создает окно для входного файла и двигает его справа налево в виде строки символов, требующих сжатие. Таким образом, считается, что метод основан на скользящем окне.

### 2.1.2 Задание варианта

Согласно варианту №13, необходимо закодировать следующую строку:

01011011001010101011

|  |  |  |
| --- | --- | --- |
| Содержимое окна поиска | Содержимое упреждающего буфера | Код последовательности |
|  | 01011011001010101011 |  |
| 0 | 1011011001010101011 | 0 |
| 10 | 11011001010101011 | 10 |
| 11 | 011001010101011 | 11 |
| 01 | 1001010101011 | 100 |
| 100 | 1010101011 | 101 |
| 101 | 0101011 | 110 |
| 010 | 1011 | 111 |
| 1011 | - | 1000 |

|  |  |
| --- | --- |
| Исходный текст | 01011011001010101011 |
| LZ-код | 0.10.011.001.0100.0101.1000.1101 |
| R | 2 3 4 |
| Вводимые коды | - 10 11 100 101 110 111 1000 |

### 2.1.3 Процесс декодирования

Алгоритм построен таким образом, что мы в состоянии восстановить таблицу строк, пользуясь только потоком кодов. Первые две группы полностью являются кодами последовательностей, а для следующих групп процесс декодирования следующий: кодом являются все цифры, кроме последней, поэтому в декодированный текст вставляется последовательность, соответствующая коду, и потом приписывается последний символ группы.

## 2.2 МЕТОД LZ78

### 2.2.1 Описание алгоритма

В отличие от LZ77, работающего с уже полученными данными, LZ78 ориентируется на данные, которые только будут получены (LZ78 не использует скользящее окно), он хранит словарь из уже просмотренных фраз. Алгоритм считывает символы сообщения до тех пор, пока накапливаемая подстрока входит целиком в одну из фраз словаря. Как только эта строка перестанет соответствовать хотя бы одной фразе словаря, алгоритм генерирует код, состоящий из индекса строки в словаре, которая до последнего введенного символа содержала входную строку, и символа, нарушившего совпадение. Затем в словарь добавляется введенная подстрока. Если словарь уже заполнен, то из него предварительно удаляют менее всех используемую в сравнениях фразу. Если в конце алгоритма мы не находим символ, нарушивший совпадения, то тогда мы выдаем код в виде (индекс строки в словаре без последнего символа, последний символ).

### 2.2.2 Задание варианта

Согласно варианту №13, необходимо закодировать следующую строку:

weaweraweaerweaerwea

|  |  |  |  |
| --- | --- | --- | --- |
| Содержимое словаря | Номер позиции | Содержимое считанной строки | Код |
|  | 1 | w | <0,w> |
| w, | 2 | e | <0,e> |
| w,e | 3 | a | <0,a> |
| w,e,a | 4 | we | <1,e> |
| w,e,a,we | 5 | r | <0,r> |
| w,e,a,we,r | 6 | aw | <3,w> |
| w,e,a,we,r,aw | 7 | ea | <2,a> |
| w,e,a,we,r,aw,ea | 8 | er | <2,r> |
| w,e,a,we,r,aw,ea,er | 9 | wea | <4,a> |
| w,e,a,we,r,aw,ea,er,wea | 10 | erw | <8,w> |
| w,e,a,we,r,aw,ea,er,wea, erw | 11 | ea | <2,a> |

В результате сжатия получаем следующую строку: 0w0e0a1e0r3w2a2r4a8w2a

Коэффициент сжатия в этом случае равен 20/22 = 0.90

### 2.2.3 Процесс декодирования

Закодированная строка состоит из групп, где есть число, являющееся кодом последовательности символов в словаре, и символ, идущий после этой последовательности. Соответственно, процесс декодирования проходит заменой чисел на соответствующие последовательности и приписыванием к ним соответствующих символов.

# 3 ЗАДАНИЕ№3

## 3.1 МЕТОД ШЕННОНА-ФАНО

### 3.1.1 Описание алгоритма

На вход поступает сообщение (текст), который требуется сжать.

1. Шаг 1. Формируется таблица частот символов, составляющих алфавит текста. Таблица состоит из строк вида: символ, частота появления символа в тексте, упорядоченных по убыванию частот.

2. Шаг 2. Находится середина таблицы, которая делит алфавит примерно на две равные части. Середина определяется так: отсчитываются строки таблицы сверху вниз, так чтобы и сверху, и снизу суммы частот были равны. Получаем как бы две части таблицы: левая (верхняя часть таблицы), правая (нижняя часть таблицы). Для символов левой части присваивается «0», для правой «1».

3. Шаг 2 повторяется до тех пор, пока мы не получим единственный элемент последовательности.

Полученная таблица хранит бинарное дерево, сформированное по алгоритму Фано. Содержание таблицы можно отобразить в форме бинарного дерева, представляя левые части таблицы как левое поддерево, а правые как правое поддерево. Записывая на левых ребрах код 0, а на правых ребрах код 1.

### 3.1.2 Задание варианта

Согласно варианту №13, необходимо закодировать следующую строку:

Эне, бене, лики, паки, Цуль, буль-буль, Калики-цваки, Эусбеус, кликмадеус, бокс.

### 3.1.3 Процесс декодирования

В полученной таблице хранятся коды всех символов, поэтому для кодирования все символы в тексте заменяются на свои коды. Для декодирования нужно считывать символы в закодированной до тех пор, пока не получится код какого-либо символа из исходного текста. Такой алгоритм гарантированно приведёт к корректному декодированию, поскольку коды являются префиксными, и ни один код символа не содержится в коде другого символа как префикс.

### 3.1.4 Таблица формирования кода

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Символ | К-во повторений | 1-я цифра | 2-я цифра | 3-я цифра | 4-я цифра | 5-я цифра | 6-я цифра | 7-я цифра | Код | Кол-во бит |
| « » | 9 | 0 | 0 | 0 |  |  |  |  | 000 | 3 |
| , | 9 | 0 | 0 | 1 |  |  |  |  | 001 | 3 |
| к | 8 | 0 | 1 | 0 |  |  |  |  | 010 | 3 |
| и | 7 | 0 | 1 | 1 | 0 |  |  |  | 0110 | 4 |
| у | 6 | 0 | 1 | 1 | 1 |  |  |  | 0111 | 4 |
| л | 6 | 1 | 0 | 0 | 0 |  |  |  | 1000 | 4 |
| б | 5 | 1 | 0 | 0 | 1 |  |  |  | 1001 | 4 |
| е | 5 | 1 | 0 | 1 | 0 |  |  |  | 1010 | 4 |
| с | 4 | 1 | 0 | 1 | 1 |  |  |  | 1011 | 4 |
| а | 4 | 1 | 1 | 0 | 0 | 0 |  |  | 11000 | 5 |
| ь | 3 | 1 | 1 | 0 | 0 | 1 |  |  | 11001 | 5 |
| э | 2 | 1 | 1 | 0 | 1 | 0 |  |  | 11010 | 5 |
| ц | 2 | 1 | 1 | 0 | 1 | 1 |  |  | 11011 | 5 |
| - | 2 | 1 | 1 | 1 | 0 | 0 |  |  | 11100 | 5 |
| н | 2 | 1 | 1 | 1 | 0 | 1 |  |  | 11101 | 5 |
| п | 1 | 1 | 1 | 1 | 1 | 0 | 0 |  | 111100 | 6 |
| в | 1 | 1 | 1 | 1 | 1 | 0 | 1 |  | 111101 | 6 |
| д | 1 | 1 | 1 | 1 | 1 | 1 | 0 |  | 111110 | 6 |
| о | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1111110 | 7 |
| м | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1111111 | 7 |

### 3.1.5 Префиксное дерево

Изображение выглядит как текст, рисунок, зарисовка, чернила

Автоматически созданное описание

Рисунок 2 – Префиксное дерево

### 3.1.6 Коэффициент сжатия

Объем исходной фразы: 8\*79=632 бита. После кодирования: кол-во повторений умножаем на кол-во бит для кодировки каждого символа. Получаем: 317 бит. Коэффициент сжатия: 632/317=1.99

## 3.2 МЕТОД ХАФФМАНА

### 3.2.1 Описание алгоритма

Метод основывается на создании бинарных деревьев. В нем узел может быть либо листом, либо внутренним. Изначально все узлы считаются листьями, которые представляют сам символ и его вес (то есть частоту появления). Внутренние узлы содержат вес символа и ссылаются на два узла-наследника. По общему соглашению, бит «0» представляет следование по левой ветви, а «1» — по правой. В полном бинарном дереве N листьев и N-1 внутренних узлов. Рекомендуется, чтобы при построении дерева Хаффмана отбрасывались неиспользуемые символы для получения кодов оптимальной длины. Кодовое дерево (дерево кодирования Хаффмана, Н-дерево) – это бинарное дерево, у которого: листья помечены символами, для которых разрабатывается кодировка; узлы (в том числе корень) помечены суммой вероятностей появления всех символов, соответствующих листьям поддерева, корнем которого является соответствующий узел.

### 3.2.2 Задание варианта

Согласно варианту №13, необходимо закодировать следующую строку:

«кликушин владислав игоревич»

### 3.2.3 Процесс декодирования

Аналогичен с предыдущим методом

### 3.2.4 Таблицы частот символов

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | к | л | и | у | ш | н | « » | в | а | д | с | г | о | р | е | ч |
| Кол-во вхожд. | 2 | 3 | 5 | 1 | 1 | 1 | 2 | 3 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Вер-сть | 0.07 | 0.11 | 0.18 | 0.03 | 0.03 | 0.03 | 0.07 | 0.11 | 0.07 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 |

Упорядочим таблицу по количеству вхождений:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Алфавит | и | л | в | « » | а | к | у | ш | н | д | с | г | о | р | е | ч |
| Кол-во вхожд. | 5 | 3 | 3 | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Вер-сть | 0.18 | 0.11 | 0.11 | 0.07 | 0.07 | 0.07 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 |

### 3.2.5 Префиксное дерево Хаффмана

Изображение выглядит как текст, зарисовка, рисунок, бумага

Автоматически созданное описание

Рисунок 3 - Префиксное дерево Хаффмана

Изображение выглядит как текст, рукописный текст, бумага, чернила

Автоматически созданное описание

Рисунок 4 - Префиксное дерево Хаффмана с кодами

### 3.2.6 Кодирование

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| к | л | и | к | у | ш | и | н | « » | в | л | а | д | и |
| 1010 | 010 | 00 | 1010 | 10110 | 10111 | 00 | 11000 | 1000 | 011 | 010 | 1001 | 11001 | 00 |
| с | л | а | в | « » | и | г | о | р | е | в | и | ч |  |
| 11010 | 010 | 1001 | 011 | 1000 | 00 | 11011 | 11100 | 11101 | 11110 | 011 | 00 | 11111 |  |

Объем исходной фразы: 8\*27=216 бит. Объем закодированной фразы: 102 бита. Коэффициент сжатия: 216/102=2.14. Средняя длина кода: 3.78

# 4 ВЫВОДЫ

В ходе выполнения практической работы были получены навыки работы с методами кодирования информации.

# 5 ИСПОЛЬЗУЕМЫЕ ИСТОЧНИКИ

1. Учебно-методическое пособие СиАОД (часть 2)
2. Приложение к практическим работам – СДО (online-edu.mirea.ru)