Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Институт радиоэлектроники и информационных технологий – РтФ

Школа профессионального и академического образования

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель: Пономарёва Ольга Алексеевна

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчёт по лабораторной №3

по дисциплине «Теория информации»

по теме: Методы упаковки данных.

Студент: Ляпустина Дарья Денисовна \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Группа: РИ-321056

Студент: Медведев Сергей Владиславович \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись)

Группа: РИ-321056

Екатеринбург

2024

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc183782646)

[АЛГОРИТМ СЖАТИЯ ДАННЫХ 4](#_Toc183782647)

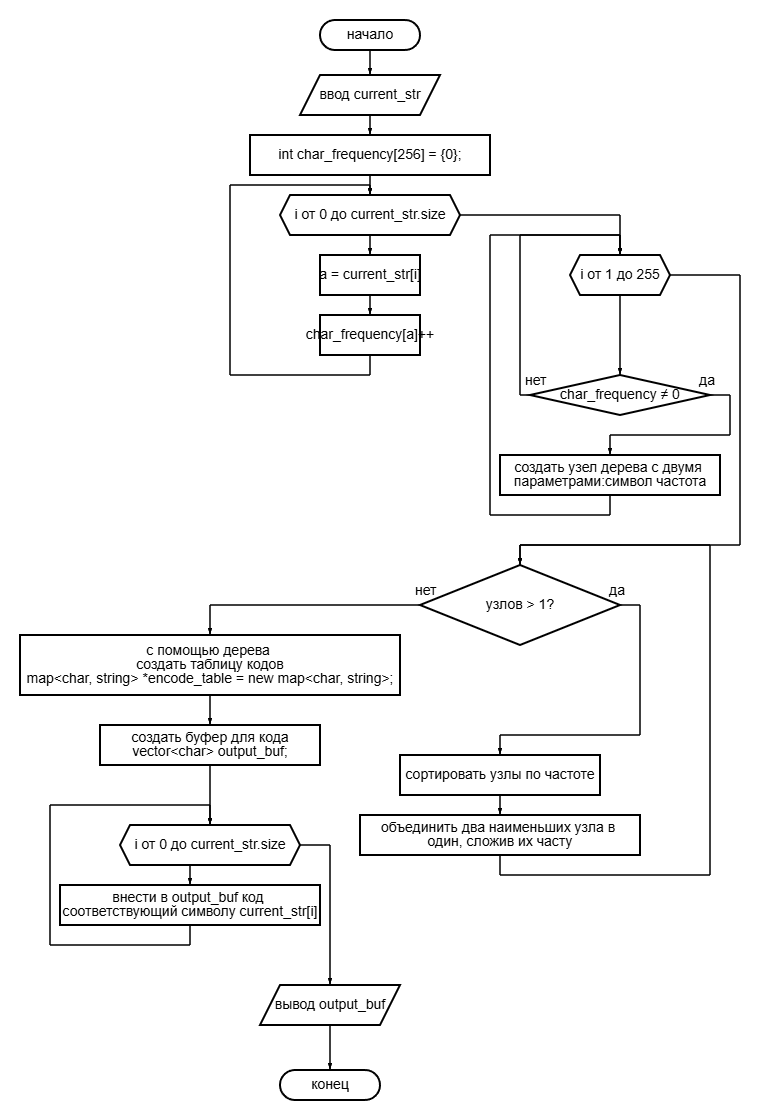
[РЕАЛИЗАЦИЯ МЕТОДОВ С ПОМОЩЬЮ КОДА 5](#_Toc183782648)

[ЗАКЛЮЧЕНИЕ 10](#_Toc183782649)

# ВВЕДЕНИЕ

Цель данной работы заключается в закреплении теоретических знаний по сжатию информации, а также программной реализации метода Хаффмена для сжатия информации, а именно, создание архиватора и дезархиватора на языке программирования C++.

# АЛГОРИТМ СЖАТИЯ ДАННЫХ



# РЕАЛИЗАЦИЯ МЕТОДОВ С ПОМОЩЬЮ КОДА

Листинг 1.1.

Методика Хаффмена. Архиватор и дезархиватор.

#include <iostream>

#include <vector>

#include <string>

#include <list>

#include <map>

using namespace std;

struct Node{

    char symbol;

    uint32\_t frq;

    Node \*left, \*right;

    bool operator<(Node anoter\_one) const{

        return frq < anoter\_one.frq;

    }

    Node(char x, uint32\_t y){symbol = x; frq = y; left=right=NULL;}

    Node(char x, uint32\_t y, Node \*L, Node \*R){symbol = x; frq = y; left= L; right=R;}

};

struct MyCompare

{

    bool operator()(const Node\* l, const Node\* r) const { return l->frq < r->frq; }

};

void mktables\_from\_tree(Node \*root, map<char, string> \*table, map<string, char> \*table2, string code = ""){

    if(root->symbol == static\_cast<char>(0x00) && root->left != NULL && root->right != NULL){

        mktables\_from\_tree(root->left, table, table2, code + "0");

        mktables\_from\_tree(root->right, table, table2, code + "1");

        return;

    }

    else{

        table[0][root->symbol] = code;

        table2[0][code] = root->symbol;

        cout << root->symbol << ":" << code << endl;

        return;

    }

}

void delete\_tree(Node \*root){

    if (root->left != NULL && root->right != NULL){

        delete\_tree(root->left);

        delete\_tree(root->right);

    }

    else {delete root;}

    return;

}

int main(int argc, char\* argv[])

{

    setlocale(LC\_ALL, "Russian");

    string current\_str;

    if (argc == 1){

        current\_str = "abracadabra";

    }

    else if((argc == 2)){

        current\_str = argv[1];

    }

    //Вывод кодируемго сообщения

    cout << "Input message:" << endl;

    cout << current\_str << endl;

    //Запись частоты каждого байта в список

    int char\_frequency[256] = {0};

    for (int j = 0; j < current\_str.size(); j++){

        char current\_byte = current\_str[j];

        char\_frequency[static\_cast<unsigned char>(current\_byte)]++;

    }

    //Создание и заполнение списка ненулевыми частотами с соответствующими символами

    list<Node\*> char\_frq\_list;

    for(int i = 0; i < 256; i++){

        if(char\_frequency[static\_cast<unsigned char>(i)] != 0){

                //Запись ненулевых частот в list

                char\_frq\_list.push\_back(new Node(static\_cast<char>(i), char\_frequency[i]));

        }

    }

    //Построение дерева

    while (char\_frq\_list.size() != 1){

        char\_frq\_list.sort(MyCompare());

        Node \*LeftNode= char\_frq\_list.front();

        char\_frq\_list.pop\_front();

        Node \*RightNode = char\_frq\_list.front();

        char\_frq\_list.pop\_front();

        char\_frq\_list.push\_front(new Node(static\_cast<char>(0x00), (LeftNode->frq + RightNode->frq), LeftNode, RightNode));

    }

    //Создание таблицы кодирования и декодирования

    Node \*root = char\_frq\_list.front();

    map<char, string> \*encode\_table = new map<char, string>;

    map<string, char> \*decode\_table = new map<string, char>;

    cout << "Encoding table:" << endl;

    mktables\_from\_tree(root,encode\_table, decode\_table);

    //Запись кода в буфер

    vector<char> output\_buf;

    for (int i = 0; i < current\_str.size(); i++){

        string code\_to\_write = encode\_table[0][current\_str[i]];

        for (int j = 0; j < code\_to\_write.size(); j++){

            output\_buf.push\_back(code\_to\_write[j]);

        }

    }

    //Вывод буфера с кодом в консоль

    cout << "Encoded text:" << endl;

    for (int i = 0; i < output\_buf.size(); i++){

        cout << output\_buf[i];

    }

    cout << endl;

    //Создание и заполнение буфера декодированного текста

    vector<char> decoder\_buf;

    string current\_encode = "";

    for (int i = 0; i < output\_buf.size(); i++){

        current\_encode = current\_encode + output\_buf[i];

        if (decode\_table->find(current\_encode) == decode\_table->end()){

            continue;

        }

        else {

            decoder\_buf.push\_back(decode\_table[0][current\_encode]);

            current\_encode = "";

        }

    }

    //Вывод буфера с декодированным текстом в консоль

    cout << "Decoded text:" << endl;

    for (int i = 0; i < decoder\_buf.size(); i++){

        cout << decoder\_buf[i];

    }

    cout << endl;

    //Освобождение памяти и завершение программы

    delete\_tree(root);

    delete encode\_table;

    delete decode\_table;

    return 0;

}

Вывод кода, представленного выше, выглядит следующим образом:

Input message:

abracadabra

Encoding table:

a:0

r:10

c:1100

d:1101

b:111

Encoded text:

01111001100011010111100

Decoded text:

abracadabra

Второй пример:

Input message:

Очень сложный TEXT

12345

!@#$%^&\*()

Encoding table:

2:00000

3:00001

\*:00010

1:00011

@:00100

E:00101

4:00110

5:00111

#:01000

$:01001

:01010

!:01011

(:01100

):01101

%:01110

&:01111

T:1000

н:1001

:1010

:1011

ы:11000

ь:11001

с:11010

ч:11011

О:111000

е:111001

X:111010

^:111011

л:111100

о:111101

ж:111110

й:111111

Encoded text:

111000110111110011001110011011110101111001111011111101001110001111111011100000101111010100010100101000011000000000100110001111010010110010001000010010111011101101111000100110001101

Decoded text:

Очень сложный TEXT

12345

!@#$%^&\*()

# ЗАКЛЮЧЕНИЕ

В ходе данной лабораторной работы были реализованы архиватор и дезархиватор, использующие методику сжатия Хаффмена. Архиватор и дезархиватор были реализованы с помощью кода на языке C++.

Алгоритм Хаффмана позволил эффективно сжимать данные, используя частоты появления символов для формирования префиксного кодирования, где более часто встречающиеся символы получают более короткие коды, а менее частые — более длинные. Это позволило снизить среднюю длину кода по сравнению с равномерным кодированием.