**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра** **ВТ**

**КУРСОВАЯ РАБОТА**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Алгоритм Хаффмана»**

**Вариант 10**

|  |  |  |
| --- | --- | --- |
| Студент гр. 0322 |  | Руссу В.А. |
| Преподаватель |  | Пестерев Д. О. |

Санкт-Петербург

2022

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Руссу Василий Андреевич | | |
| Группа 0322 | | |
| Тема работы: алгоритм Хаффмана. | | |
| Исходные данные: реализовать сжатие символьных данных и обратную операцию с помощью Алгоритма Хаффмана. | | |
| Содержание пояснительной записки: «Содержание», «Описание программы», «Оценка сложности каждого метода», «Примеры работы программы», «Заключение», «Список использованных источников». | | |
| Предполагаемый объем пояснительной записки:  Не менее 10 страниц | | |
| **Дата выдачи задания: 15.11.2022** | | |
| Дата сдачи курсовой работы: 05.12.2022 | | |
| Дата защиты курсовой работы: 12.12.2022 | | |
| Студент |  | Руссу В.А. |
| Преподаватель |  | Пестерев Д. О. |

**Содержание**

Оглавление

[Описание программы 4](#_Toc121252397)

[Оценка сложности каждой функции 5](#_Toc121252398)

[Примеры работы программы 6](#_Toc121252399)

[Заключение 13](#_Toc121252400)

[Приложение 1. Листинг программного кода 14](#_Toc121252401)

[Приложение 2. Список использованных источников. 17](#_Toc121252402)

# Описание программы

Данная программа получает текст в виде символьных данных из файла (1.txt), далее кодирует данный текст при помощи алгоритма Хаффмана и записывает закодированный текст в другой файл (output.txt), после происходит декодирование текста, и нам выводится расшифрованный текст.

Функции, реализованные в программе:

* BuildTable – функция, которая ассоциирует символ с его кодом и записывает их в файл value\_freq.
* print – выводит дерево.

# Оценка сложности каждого метода.

BuildTable – функция, которая присваивает каждому символу свой уникальный код. O(N)

print – функция, которая выводит дерево. O(N)

# Примеры работы программы

Пример 1.

Для начала запишем нужный нам текст в файл 1.txt и запомним его размер (Рисунок 1).

Вводимый текст: “Huffman Coding is a technique of compressing data to reduce its size without losing any of the details. It was first developed by David Huffman.

Huffman Coding is generally useful to compress the data in which there are frequently occurring characters.”

Размер файла: 253 байта.

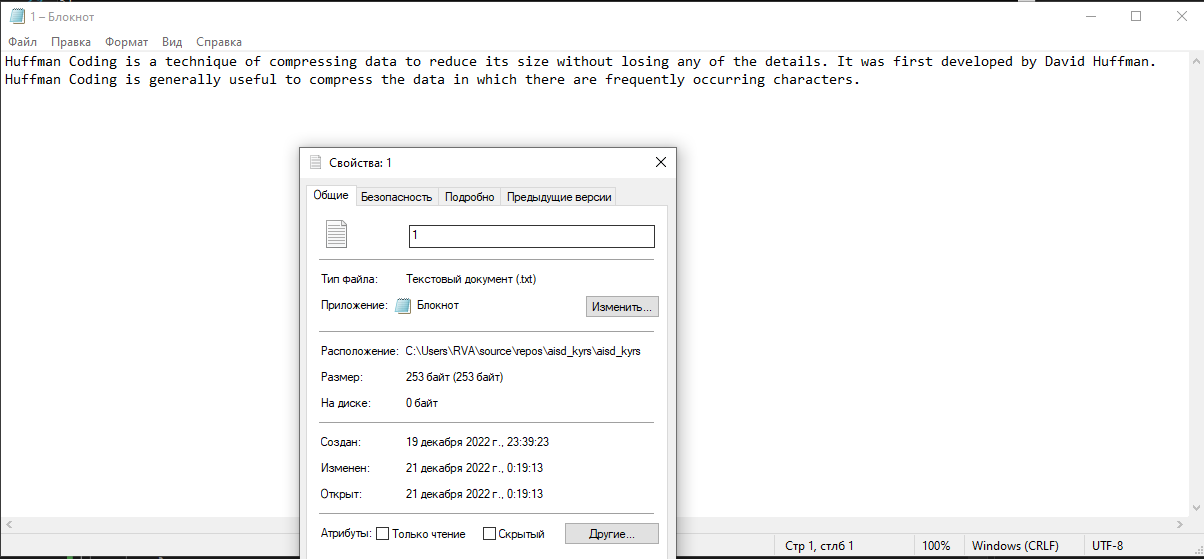


Рисунок 1

Далее запускаем программу. Смотрим на размер сжатого файла output.txt (Рисунок 2) и декодированную информацию, которая выводится в консоли (Рисунок 3).

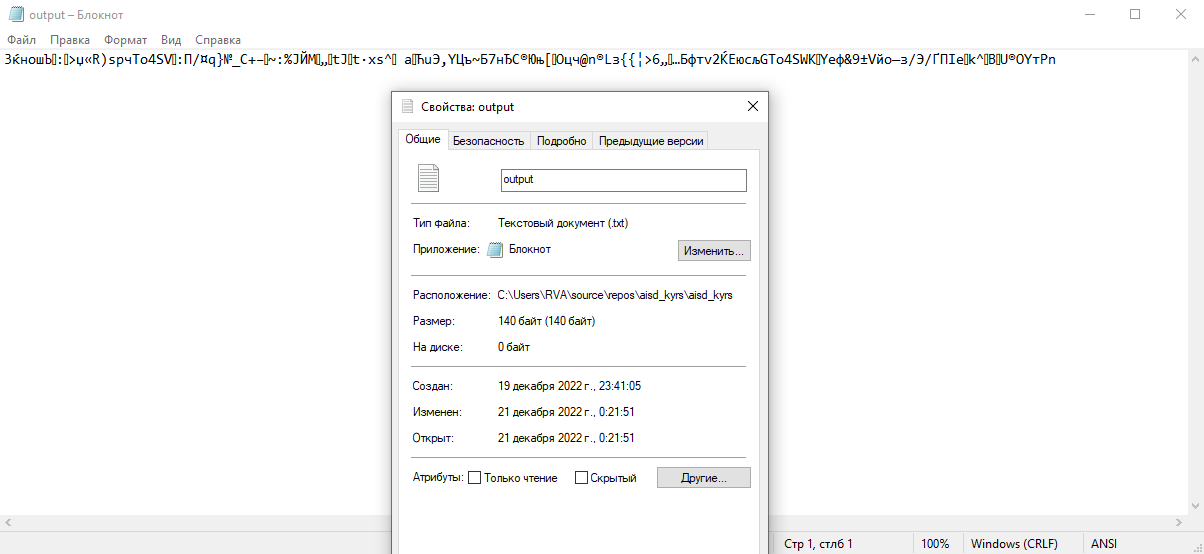


Рисунок 2

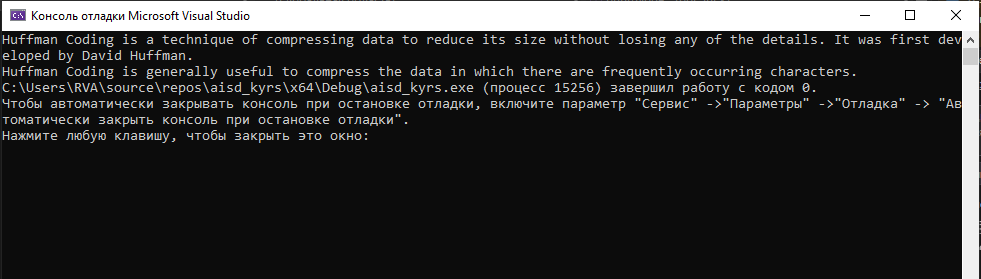


Рисунок 3

Как итог размер сжатого файла уменьшился с 253 (1.txt) до 140 (output.txt) байт и декодирование информации из файла output.txt прошло успешно.

Степень сжатия 55%

Пример 2.

Запишем нужный нам текст в файл 1.txt и запомним его размер (Рисунок 4).

Размер файла: 187 КБ.

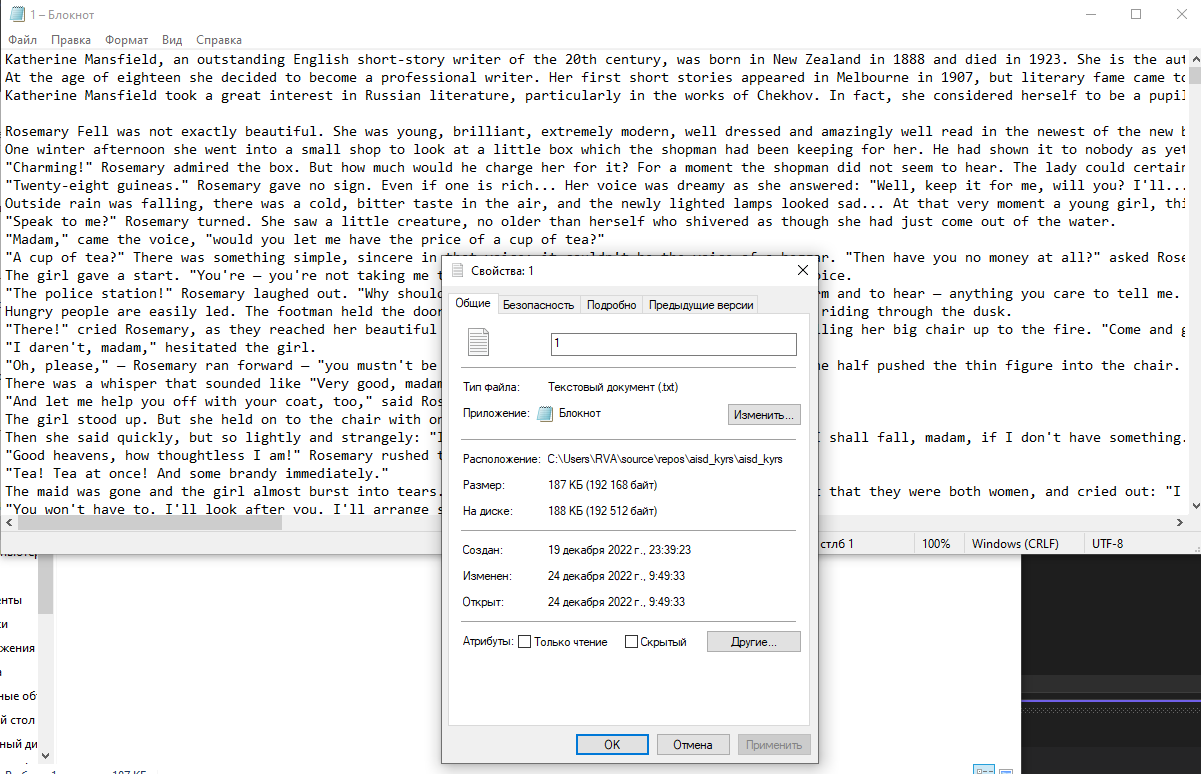


Рисунок 4

Далее запускаем программу. Смотрим на размер сжатого файла output.txt (Рисунок 5) и декодированную информацию, которая выводится в консоли (Рисунок 6).

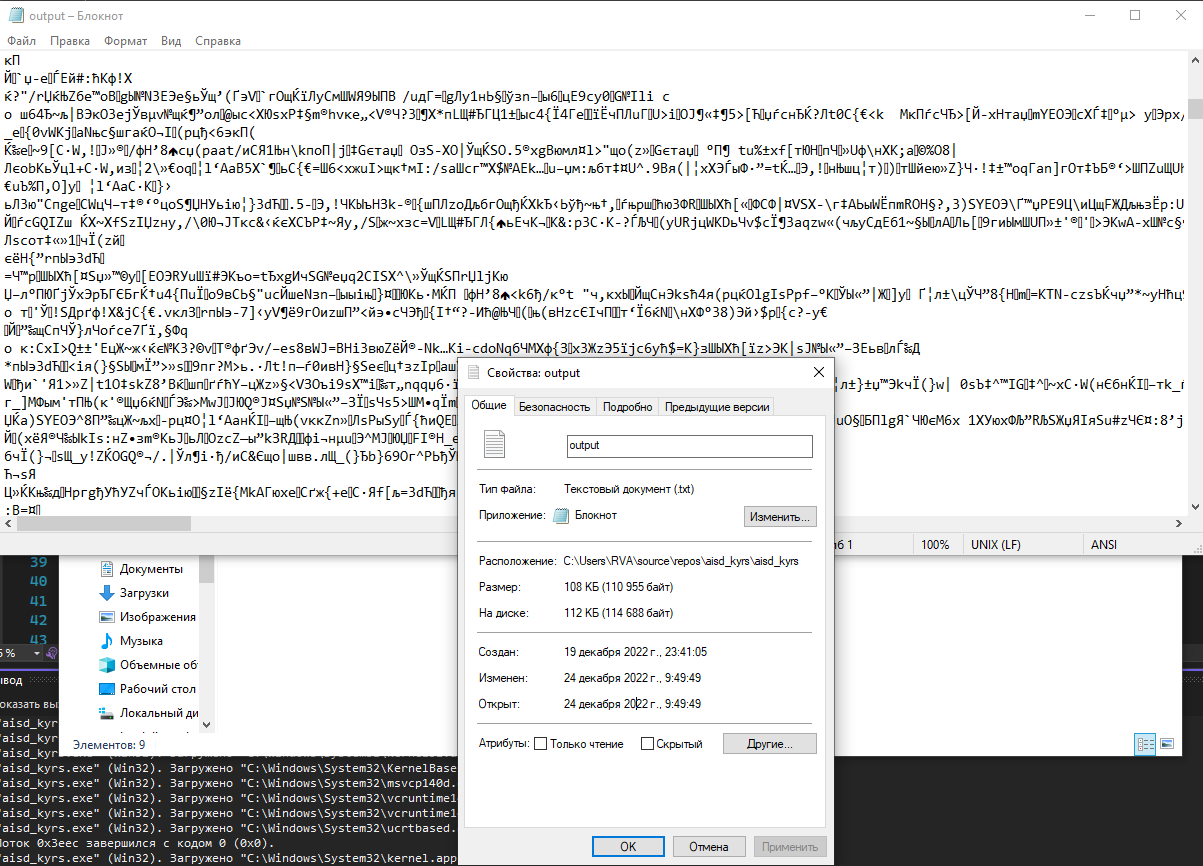


Рисунок 5

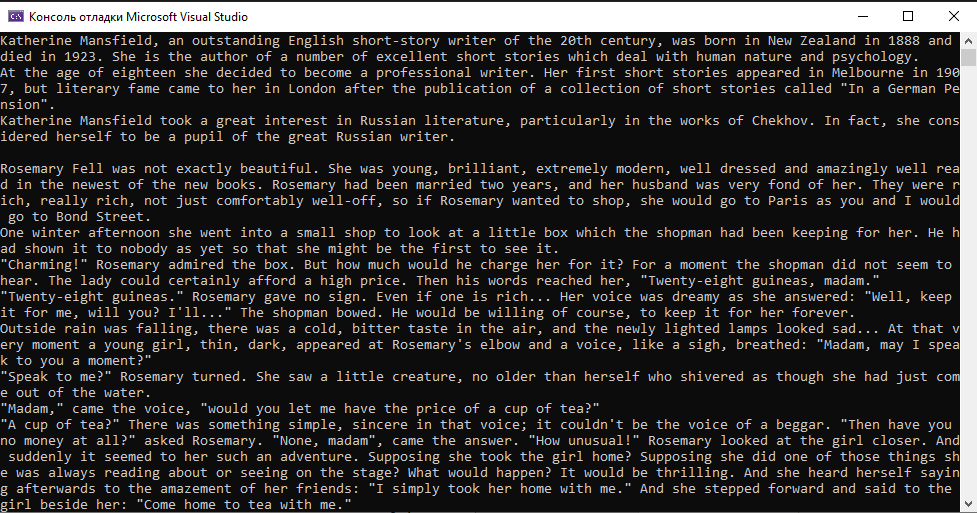


Рисунок 6

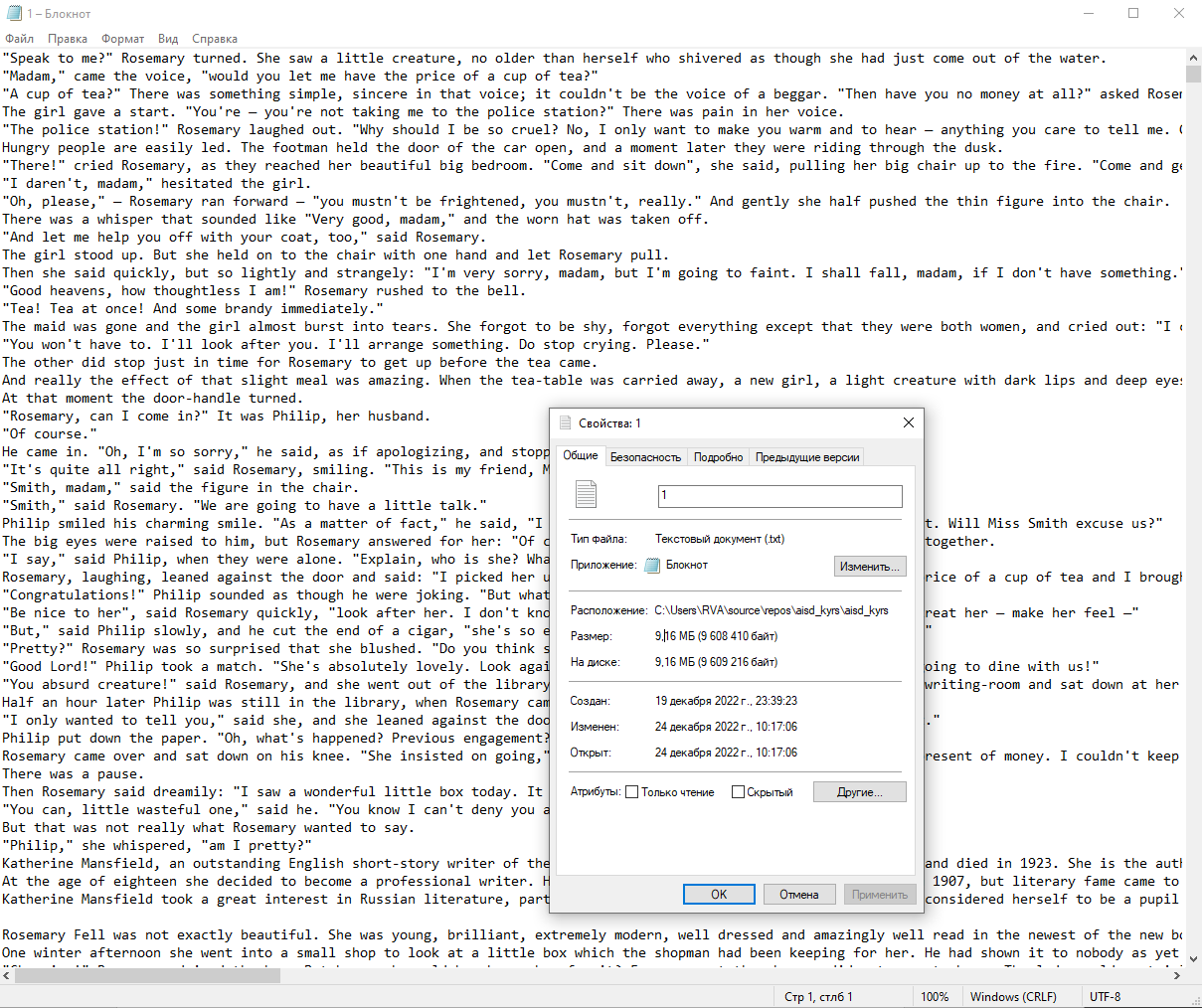
Как итог размер сжатого файла уменьшился с 187КБ (1.txt) до 108КБ (output.txt) и декодирование информации из файла output.txt прошло успешно.

Степень сжатия 57%.

Пример 3.

Запишем нужный нам текст в файл 1.txt и запомним его размер (Рисунок 7).

Размер файла: 9.16 МБ.

Рисунок 7

Далее запускаем программу. Смотрим на размер сжатого файла output.txt (Рисунок 8) и декодированную информацию, которая выводится в консоли (Рисунок 9).

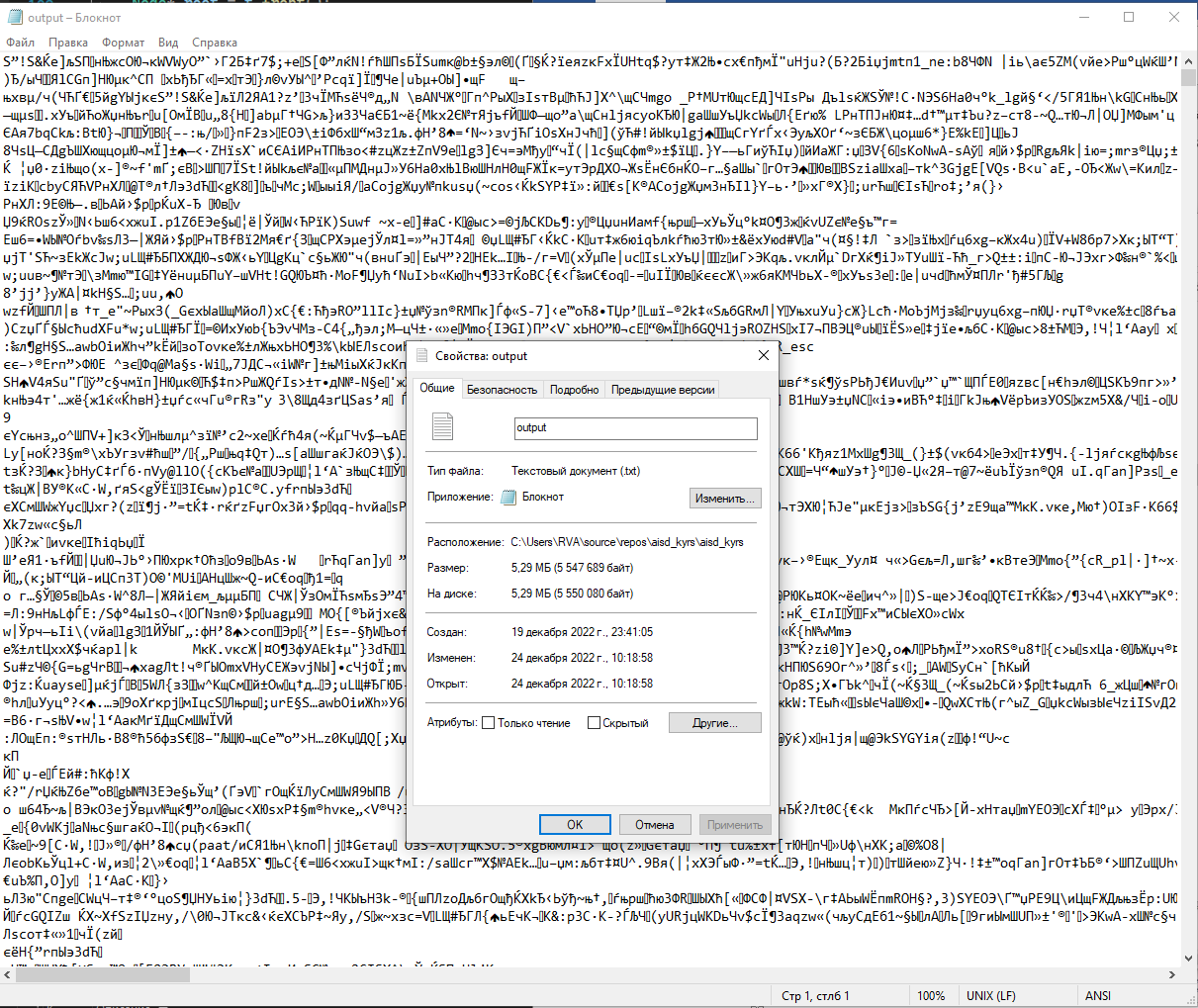


Рисунок 8

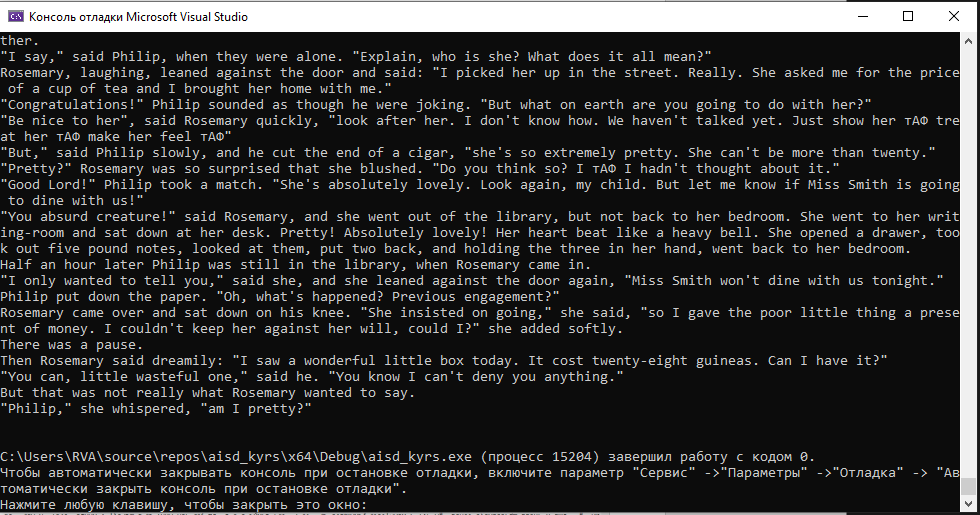


Рисунок 9

Как итог размер сжатого файла уменьшился с 9.16 МБ (1.txt) до 5.29 МБ (output.txt) и декодирование информации из файла output.txt прошло успешно.

Степень сжатия 57%.

Для того, чтобы декодировать код было возможно без исходника, в файле value\_freq была добавлена таблица символ – значение, где значение – это код символа в бинарном виде. Пример на рисунке 12.

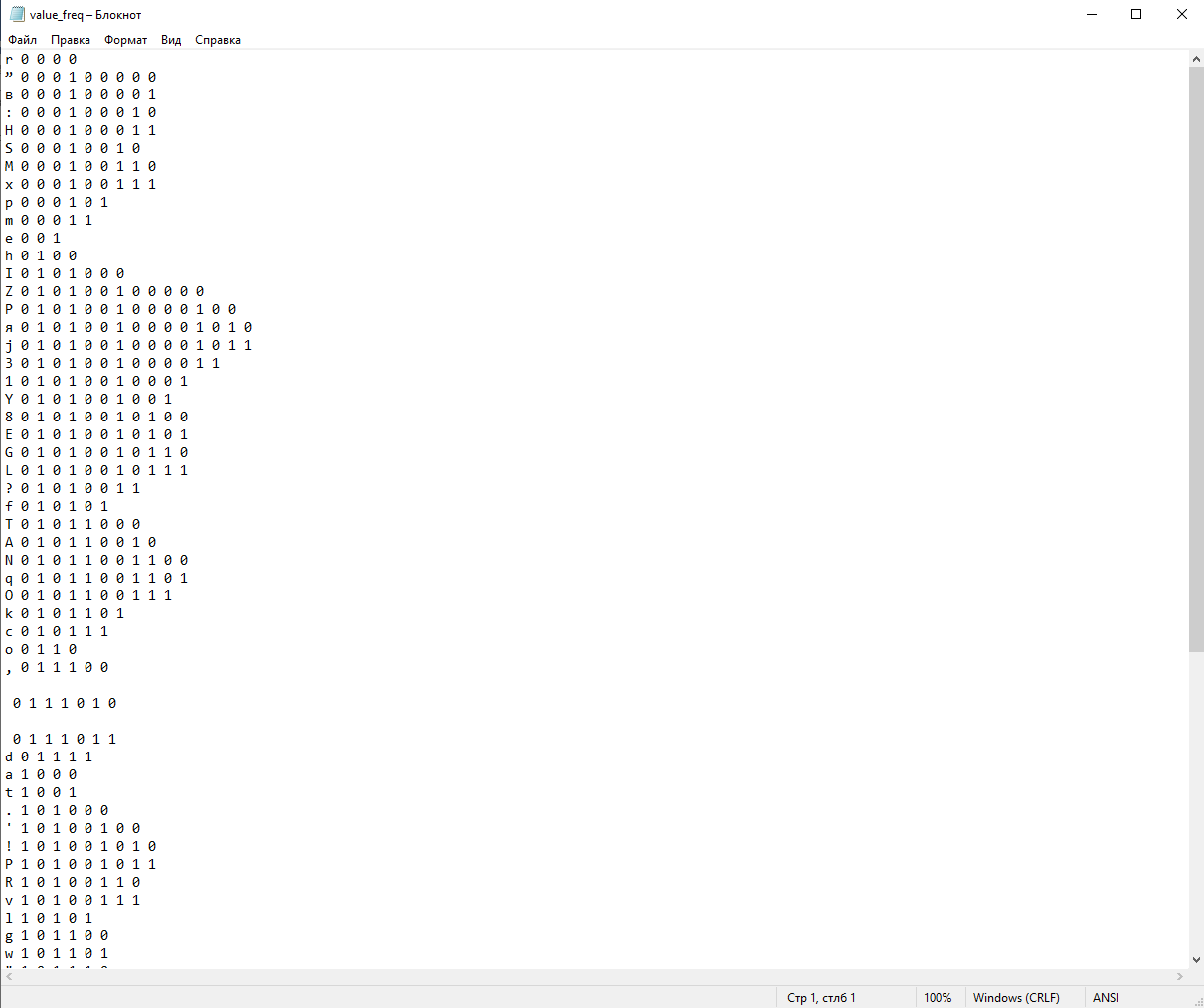


Рисунок 12

# Заключение

В ходе выполнения курсовой работы были приобретены навыки кодирования и декодирования информации при помощи алгоритма Хаффмана.

# Приложение 1. Листинг программного кода

<https://github.com/RVA-t/aisd_kyrs>

#include <iostream>

#include <vector>

#include <map>

#include <list>

#include <fstream>

using namespace std;

class Node

{

public:

int a;

char c;

Node\* left, \* right;

Node() { left = right = NULL; }

Node(Node\* L, Node\* R){

left = L;

right = R;

a = L->a + R->a;

}

};

struct MyCompare

{

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

};

vector<bool> code;

map<char, vector<bool> > table;

void BuildTable(Node\* root)

{

bool temp;

if (root->left != NULL){

code.push\_back(0);

BuildTable(root->left);

}

if (root->right != NULL){

code.push\_back(1);

BuildTable(root->right);

}

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

table[root->c] = code;

}

if (!code.empty()) code.pop\_back();

}

void print(Node\* root, int k){

if (root != NULL) {

print(root->left, k + 1);

for (int i = 0; i < k; i++) {

cout << " ";

}

if (root->c) {

cout << root->a << " (" << root->c << ")" << endl;

} else {

cout << root->a << endl;

}

print(root->right, k + 1);

}

}

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

{

ifstream f("1.txt", ios::out | ios::binary);

map<char, int> m;

while (!f.eof()){

char c = f.get();

m[c]++;

}

list<Node\*> t;

for (map<char, int>::iterator itr = m.begin(); itr != m.end(); ++itr){

Node\* p = new Node;

p->c = itr->first;

p->a = itr->second;

t.push\_back(p);

}

while (t.size() != 1){

t.sort(MyCompare());

Node\* SonL = t.front();

t.pop\_front();

Node\* SonR = t.front();

t.pop\_front();

Node\* parent = new Node(SonL, SonR);

t.push\_back(parent);

}

Node\* root = t.front();

//print(root, 0);

BuildTable(root);

f.clear(); f.seekg(0);

ofstream g("output.txt", ios::out | ios::binary);

int count = 0; char buf = 0;

while (!f.eof()){

char c = f.get();

vector<bool> x = table[c];

for (int n = 0; n < x.size(); n++){

buf = buf | x[n] << (7 - count);

count++;

if (count == 8) { count = 0; g << buf; buf = 0; }

}

}

f.close();

g.close();

//Расшифровка

ifstream F("output.txt", ios::in | ios::binary);

Node\* p = root;

count = 0; char byte;

byte = F.get();

while (!F.eof()){

bool b = byte & (1 << (7 - count));

if (b) p = p->right; else p = p->left;

if (p->left == NULL && p->right == NULL) { cout << p->c; p = root; }

count++;

if (count == 8) { count = 0; byte = F.get(); }

}

F.close();

return 0;

}

# Приложение 2. Список использованных источников.

Алгоритм сжатия Хаффмана [Электронный ресурс]. URL: https://habr.com/ru/company/otus/blog/497566 (дата обращения: 20.12.2022)

Huffman Coding [Электронный ресурс]. URL: https://www.programiz.com/dsa/huffman-coding (дата обращения: 20.12.2022)

Код Хаффмана [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Код\_Хаффмана (дата обращения: 20.12.2022)