МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Программирование»

Тема: Статическое кодирование и декодирование текстового файла методами Хаффмана и Шеннона-Фано

Студент гр. 7303 _____ Петров С.А.

Преподаватель _____ Балтрашевич В.Э.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Петров С.А.		
Группа 7303		
Тема работы: Кодирование и декодирование текстового файла методами Хаффмана и Шеннона-Фано		
Исходные данные (цель): Программа должна реализовывать кодирование и декодирование текстового файла с помощью алгоритма Хаффмана и алгоритма Шеннона-Фано.		
Содержание пояснительной записки: 1. Содержание 2. Введение 3. Класс Coder 4. Описание приложения на Qt 5. Краткое описание работы программы 6. Примеры работы программы 7. Заключение 8. Список использованных источников 9. Приложение 1. Примеры работы программы 10. Приложение А. Исходный код		
Дата выдачи задания: 05.09.2018		
Дата сдачи реферата:		
Дата защиты реферата:		
Студент		Петров С.А.
Преподаватель		Балтрашевич В.Э.

АННОТАЦИЯ

В данной курсовой работе представлена программа для кодирования и декодирования текстовых файлов, реализованная в виде оконного графического приложения с использованием C++ фреймворка Qt. Программа содержит: функции для загрузки и сохранения файлов, кодирования и декодирования с помощью алгоритмов Хаффмана и Шеннона-Фано. В данной курсовой работе представлены: полный исходный код программы, описание работы представленных функций и примеры работы программы: кодирования и декодирования текстовых файлов различного размера с помощью алгоритмов Хаффмана и Шеннона-Фано.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

При создании шаблона оконного приложения, фреймворк Qt создает класс MainWindow с графической формой, а также файл, в котором происходит создание объекта этого класса и запуск приложения. Для выполнения цели данной курсовой работы было решено создать отдельный класс, содержащий функции для кодирования и декодирования текстовых файлов методами Хаффмана и Шеннона-Фано, представление словаря кодируемых символов, дерево для построения беспрефиксного кода, а в главном классе MainWindow реализовать пользовательский интерфейс.

Для хранения дерева для построения беспрефиксного кода была заведена стуктура, хранящая информацию о символах, частоте их встречаемости в тексте, строку с кодом данных символов, указатели на левое и правое поддерево. Была также заведена структура для сравнения элементов дерева.

Вид структур для построения дерева кодов и сравнения элементов дерева приведены ниже:

```
struct Node {
    Node(std::string _chars, int frequence) {
        chars = _chars;
        freq = frequence;
    }
    Node(Node* _left, Node* _right){
        chars[0] = '\0';
        freq = _left->freq + _right->freq;
        left = _left;
        right = _right;
    }
    std::string chars;
    int freq = 0;
    Node *left = nullptr, *right = nullptr;
    std::string code = "";
    ~Node(){
        delete left;
        delete right;
    }
};
    struct HuffCompare{
        bool operator()(const Node* first, const Node* second) const{
            return first->freq >= second->freq;
        }
    };
    struct ShanCompare{
        bool operator()(const Node* first, const Node* second) const{
            return first->freq <= second->freq;
        }
  };
```

1. КЛАСС CODER

1.1. Интерфейс и реализация класса Coder

Помимо самого класса, были также объявлены вышеприведенные структуры.

Были обьявлен и реализованы: конструктор по умолчанию и деструктор, удаляющий построенное дерево и очищающий выделенную под него память. Были заведены следующие поля: _root – указатель на корень дерева, _map – словарь, который будет однозначно сопоставлять символу его код, _text – строка, содержащая кодируемый текст. Так же были объявлены методы: shannon_encode() – метод, осуществляющий построение беспрефикского кода для символьной последовательности, используя алгоритм Шеннона-Фано; huffman_encode() – метод, осуществляющий построение беспрефикского кода для символьной последовательности, используя алгоритм Хаффмана; encode_file() – метод, осуществляющий кодирование текстовой последовательности на основе построенного кода и запись закодированных данных в файл; decode_file() – метод, осуществляющий декодирование закодированного файла и вывод декодированной информации; get_map() – метод, возвращающий построенный словарь кодов; makeShannonCodes() – метод, осуществляющий построение дерева кодов в соответствии с алгоритмом Шеннона-Фано; makeHuffmanCodes() – метод, осуществляющий построение дерева кодов в соответствии с алгоритмом Хаффмана; clear() – метод, очищающий словарь и дерево для построения нового кода; write_tree () и read_tree() – методы, осуществляющие запись дерева в файл с закодированными данными и извлечение дерева, для декодирования информации. Полный код класса приведен в приложении А:

1.1.1. Конструктор и деструктор

Были заведены: конструктор по умолчанию и деструктор, удаляющий построенное дерево и очищающий выделенную под него память. Код конструктора и деструктора представлен на следующей странице:

```
Coder(){}

~Coder(){
    delete _root;
}
```

1.1.2. Метод shannon_encode()

Функция считывает информацию из файла, осуществляет подсчет частот символов (количества вхождений в текст), осуществляет сортировку символов в порядке невозрастания, после чего осуществляющий построение словаря и дерева кодов с помощью метода makeShannonCodes(). Функция возврщает полученный словарь. Код функции представлен ниже:

```
void shannon_encode(std::string const& file_name){
    clear();
    std::ifstream file;
    file.open(file_name, std::ios::in | std::ios::binary);
    std::string buf;
    while(file.good()){
        std::getline(file, buf);
        _text += buf;
    }
    std::map<char, int> freq;
    for(auto& ch : _text){
        freq[ch]++;
    }
    std::priority_queue<Node*, std::vector<Node*>, ShanCompare> sortedChars;
    for(auto& node : freq){
        sortedChars.push(new Node(std::string(1, node.first), node.second));
    }

    std::map<char, std::string> res;
    if(sortedChars.size() == 1){
        auto el = sortedChars.top();
        res[el->chars[0]] = "0";
        delete sortedChars.top();
        _map = res;
        return;
    }

    std::string start;
    while(sortedChars.size() > 0){
        auto el = sortedChars.top();
        start += el->chars;
        sortedChars.pop();
    }
    _root = new Node(start, _text.size());
    makeShannonCodes(_root, freq, res);
    _map = res;
}
```

1.1.3. Метод makeShannonCodes()

Функция принимает указатель на узел дерева, содержащее текущий алфавит, словарь частот, словарь кодов, который необходимо составить, и текущий код. Функция делит алфавит на две части, суммарные вероятности символов которых максимально близки друг другу, после чего создает левое и правное поддерево, передавая в них соответственно левую и правую часть алфавита, и добавляет к коду 1 для правого и 0 для левого поддерева. Если в переданном алфвавите один символ, то в словарь кодов записыватся текущий переданный код. Код функции представлен ниже:

```
void makeShannonCodes(Node* node, std::map<char, int>& freq, std::map<char,
std::string>& map, std::string code=""){
    if (node->chars.size() == 1) map[node->chars[0]] = code;
    if(node->chars.size() > 1) {
        int startPos = 0;
        int first_part = 0;
        int total_freq = node->freq;
        for (startPos = 0; startPos < node->chars.size(); startPos++) {
            if (first_part >= (total_freq - first_part) || (startPos + 1 ==
node->chars.size())) break;
            first_part += freq[node->chars[startPos]];
        }
        Node* left = new Node({node->chars.begin(), node->chars.begin() +
startPos}, first_part);
        Node* right = new Node({node->chars.begin() + startPos, node-
>chars.end()}, total_freq - first_part);
        node->left = left;
        node->right = right;
        makeShannonCodes(node->left, freq, map, code+"0");
        makeShannonCodes(node->right, freq, map, code+"1");
    }
}
```

1.1.4. Метод huffman_encode()

Функция считывает информацию из файла, осуществляет подсчет частот символов (количества вхождений в текст), осуществляет сортировку символов в порядке убывания частот, после чего построение словаря дерева: выбираются два свободных узла дерева с наименьшими частотами, создается их родитель с весом, равным их суммарному весу. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка. Алгоритм повторяется до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева. После чего функция осуществляет

построение словаря с помощью фунции makeHuffmanCodes(). Функция возврщает полученный словарь. Код функции представлен ниже:

```
void huffman_encode(std::string const& file_name){
    clear();
    std::ifstream file;
    file.open(file_name, std::ios::in | std::ios::binary);
    std::string buf;
    while(file.good()){
        std::getline(file, buf);
        _text += buf;
    }
    std::map<char, int> freq;
    for(auto& ch : _text){
        freq[ch]++;
    }
    std::priority_queue<Node*, std::vector<Node*>, HuffCompare> sortedChars;
    for(auto& node : freq){
        sortedChars.push(new Node(std::string(1, node.first), node.second));
    }

    std::map<char, std::string> res;
    if(sortedChars.size() == 1){
        auto el = sortedChars.top();
        res[el->chars[0]] = "0";
        delete sortedChars.top();
        _map = res;
        return;
    }

    while(sortedChars.size() > 1){
        auto left = sortedChars.top();
        sortedChars.pop();
        auto right = sortedChars.top();
        sortedChars.pop();
        sortedChars.push(new Node(left, right));
    }
    _root = sortedChars.top();
    sortedChars.pop();
    makeHuffmanCodes(_root, res);
    _map = res;
}
```

1.1.5. Метод makeHuffmanCodes()

Функция принимает указатель на уже построенное дерево, словарь кодов, который необходимо построить, и текущий код. Если переданный узел дерева – лист, то функция добавляет текущий код в словарь как код символа в листе и завершает работу. В противном случае функция вызывает саму себя для левого и правого поддерева, прибавляя к коду 0 для левого, и 1 для правого поддерева. Код функции представлен на следующей странице:

```
void makeHuffmanCodes(Node* node, std::map<char, std::string>& map,
std::string code=""){
    if (!node) return;
    if (node->chars[0] != '\0') map[node->chars[0]] = code;
    makeHuffmanCodes(node->left, map, code+"0");
    makeHuffmanCodes(node->right, map, code+"1");
}
```

1.1.6. Метод write_tree()

Функция принимает указатель на дерево и поток для вывода. Функция реализует запись дерева в сокращенном виде для возможности декодировать файл. В случае, если текущий элемент – не лист, в поток записывается 0 и функция вызывает саму себя для правого и левого поддерева, в противном случае функция записывает 1 и значение в листе (таким образом записывается необходимая для декодирования “структура дерева” и алфавит). Данный способ позволяет в дальнейшем однозначно восстановить дерево по данному представлению. Код функции представлен ниже:

```
void write_tree(Node* node, std::ofstream& file){
    if(!node->left && !node->right){
        file.write("1", sizeof(char));
        file.write(reinterpret_cast<char*>(&node->chars[0]), sizeof(char));
    } else {
        file.write("0", sizeof(char));
        write_tree(node->left, file);
        write_tree(node->right, file);
    }
}
```

1.1.7. Метод read_tree ()

Функция принимает поток для чтения и реализует чтение дерева из потока, в который оно записано в вышеописанном виде. Функция возвращает указатель на корень считаннгого дерева. Код функции представлен ниже:

```
Node* read_tree(std::ifstream& file){
    unsigned char ch;
    file.read(reinterpret_cast<char*>(&ch), 1);
    if(ch == '1'){
        file.read(reinterpret_cast<char*>(&ch), 1);
        return new Node(std::string(1, ch), 0);
    } else {
        Node* left = read_tree(file);
        Node* right = read_tree(file);
        return new Node(left, right);
    }
}
```

1.1.8. Метод encode_file()

Функция принимает строку с именем файла, которое будет у закодированного файла. Функция создает этот файла, записывает в него дерево с помощью функции write_tree(), затем “упаковывает” кодированную строку в char’ы, разбивая на куски по 8 символов, и устанавливая соответствующие биты символа. Например, если у символа “x” код 1, а у символа “y” код 0, и исходная строка представляет собой “xyyyyyx”, то строка из кодов – 1000001, если представить ее в виде 8 бит то это 01000001 – число 65 в десятичной системе, а 65 это код символа “A”, сл-но “xyyyyyx” = ”A”. Таким образом вместо 7 байт нам потребовался всего 1. Код функции представлен ниже:

```
std::string encode_file(std::string const& file_name){
    std::ofstream file;
    file.open(file_name, std::ios::out | std::ios::binary);
    write_tree(_root, file);
    file.write(" ", 1);
    std::string res;
    int count  = 0;
    unsigned char ch = 0;
    for(auto& c : _text){
        std::string code = _map[c];
        res += _map[c];
        for(int i = 0; i < code.size(); i++){
            if(code[i] == '1')
                ch = ch | (1 << (7 - count));
            count++;
            if(count == 8){
                file.write(reinterpret_cast<char*>(&ch), 1);
                ch = 0;
                count = 0;
            }
        }
    }
    file.close();
    return res;
}
```

1.1.9. Метод decode_file()

Функция принимает имя декодируемого файла и открывает его на чтение. Далее восстанавливается дерево с помощью функции read_tree(). Затем функция считывает закодированную в символы битовую последовательности и с помощью дерева восстанавливает исходную информацию. Например, на входе код символа 010, для полуения символа необходимо перейти в левого

поддерево, затем в правое, и опять в левое, в котором будет лист с нужным символом. Код функции представлен на ниже

```
std::string encode_file(std::string const& file_name){
    std::ofstream file;
    file.open(file_name, std::ios::out | std::ios::binary);
    write_tree(_root, file);
    file.write(" ", 1);
    std::string res;
    int count  = 0;
    unsigned char ch = 0;
    for(auto& c : _text){
        std::string code = _map[c];
        res += _map[c];
        for(int i = 0; i < code.size(); i++){
            if(code[i] == '1')
                ch = ch | (1 << (7 - count));
            count++;
            if(count == 8){
                file.write(reinterpret_cast<char*>(&ch), 1);
                ch = 0;
                count = 0;
            }
        }
    }
    file.close();
    return res;
}
```

1.1.10. Метод clear()

Функция очищает словарь кодов, дерево и исходную текстовую последовательность для дальнейшего переиспользования. Код функции представлен ниже.

```
void clear(){
    _text.clear();
    _map.clear();
    if(_root){
        delete _root;
    }
}
```

2. ОПИСАНИЕ ПРИЛОЖЕНИЯ QT

2.1. Приложение Qt

Вышеописанный класс был подключен в приложение, созданное с использоваением фреймворка Qt. Был реализован графический интерфейс, добавлены кнопки для выбора файла с исходными данными, для выбора выходного файла, переключатели для выбора метода кодирования, меню для

вызова справки. Были добавлены виджеты для вывода содержимого исходного файла и словаря кодов, для вывода декодированной информации.

3. КРАТКОЕ ОПИСАНИЕ РАБОТЫ ПРОГРАММЫ

3.1. Работа с приложением

Запустив приложение, пользователю необходимо выбрать требуемый метод кодирования, после чего выбрать исходный текстовый файл с помощью кнопки Input file, после чего выбрать имя выходного файла с помощью кнопки Output file. Для кодирования исходного файла и записи в выходной файл необходимо нажать кнопку Encode. Одновременно на экран будет выведен полученный словарь кодов. Для декодирования пользователю необходимо выбрать файл для декодирования, и нажать на кнопку Decode, и на экран будет выведена декодированная информация. Для получения справки необходимо перейти в меню Options → Help, либо нажать кнопку F1. Для получения сведений об авторе программы необходимо перейти в меню Options → About, либо нажать кнопку F2.

3.2. Состояния программы

3.2.1. Исходное состояние

Вид исходного состояния приложения приведен на рис. 1 в приложении 1.

3.2.2. Открытие файла с исходными данными

Вид приложения после открытия файла с исходными данными приведен на рис. 2 в приложении 1.

3.2.3. Кодирование

Вид приложения после кодирования файла с исходными данными приведен на рис. 3 в приложении 1.

3.2.4. Открытие файла с закодированными данными

Вид приложения после открытия файла с закодированными данными приведен на рис. 4 в приложении 1.

3.2.5. Декодирование

Вид приложения после декодирования приведен на рис. 5 в приложении 1.

3.2.6. Вызов справки

Вызова справки приведен на рис. 6 в приложении 1.

4. ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

4.1. Кодирование и декодирование файла sample1.txt

Кодирование и декодирование файла sample1.txt приведены на рис. 7-9 в Приложении 1.

4.2. Кодирование и декодирование файла sample2.txt

Кодирование и декодирование файла sample2.txt приведены на рис. 10-12 в Приложении 1.

4.3. Кодирование и декодирование файла sample3.txt

Кодирование и декодирование файла sample3.txt приведены на рис. 13-15 в Приложении 1.

ЗАКЛЮЧЕНИЕ

Была написана программа для кодирования и декодирования текстовых файлом с помощью методов Хаффмана и Шеннона-Фано, реализованная в виде оконного графического приложения с использованием C++ фреймворка Qt. Программа позволяет пользователю загружать, кодировать и декодировать текстовые файлы выбранным методом. Сборка проекта приложения происходит без ошибок и предупреждений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Электронная версия книги “Язык программирования C++” за авторством Бьерна Страуструпа

URL:http://8361.ru/6sem/books/Straustrup-Yazyk_programmirovaniya_c.pdf

2. Официальная документация Qt

URL:http://doc.qt.io/qt-5/index.html

3. Алгоритм Хаффмана

URL: http://algolist.manual.ru/compress/standard/huffman.php

4. Алгоритм Шеннона-Фано

URL: https://studfiles.net/preview/4349019/page:11/

ПРИЛОЖЕНИЕ 1

ПРИМЕРЫ СОСТОЯНИЯ ПРИЛОЖЕНИЯ И РАБОТЫ ПРОГРАММЫ

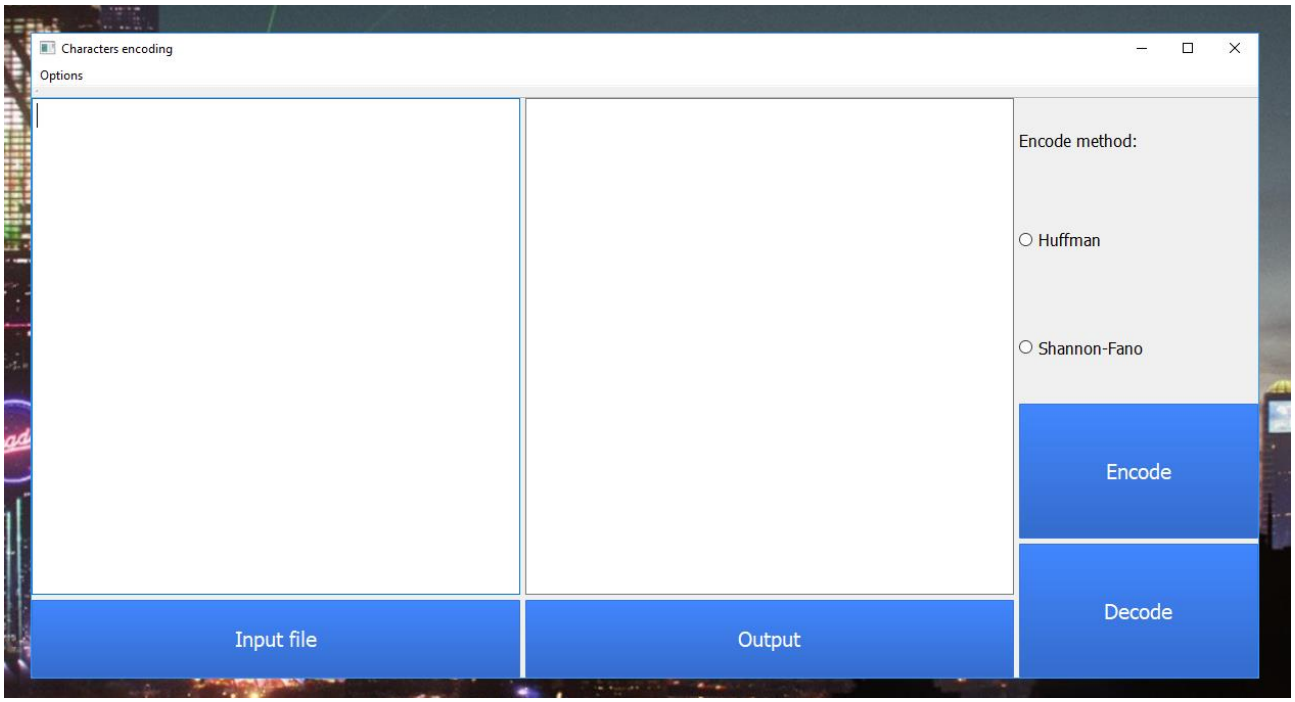

Рисунок 1 – Исходное состояние приложения

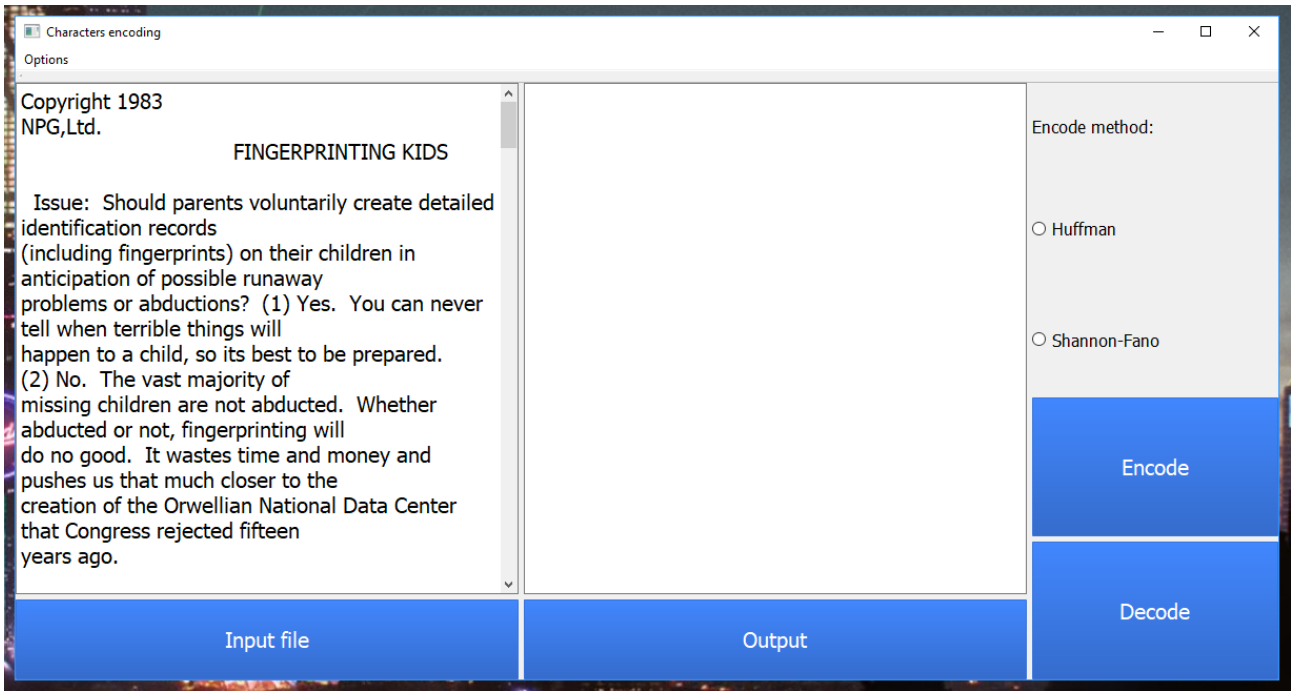

Рисунок 2 – Состояние после открытия файла с данными для кодирования

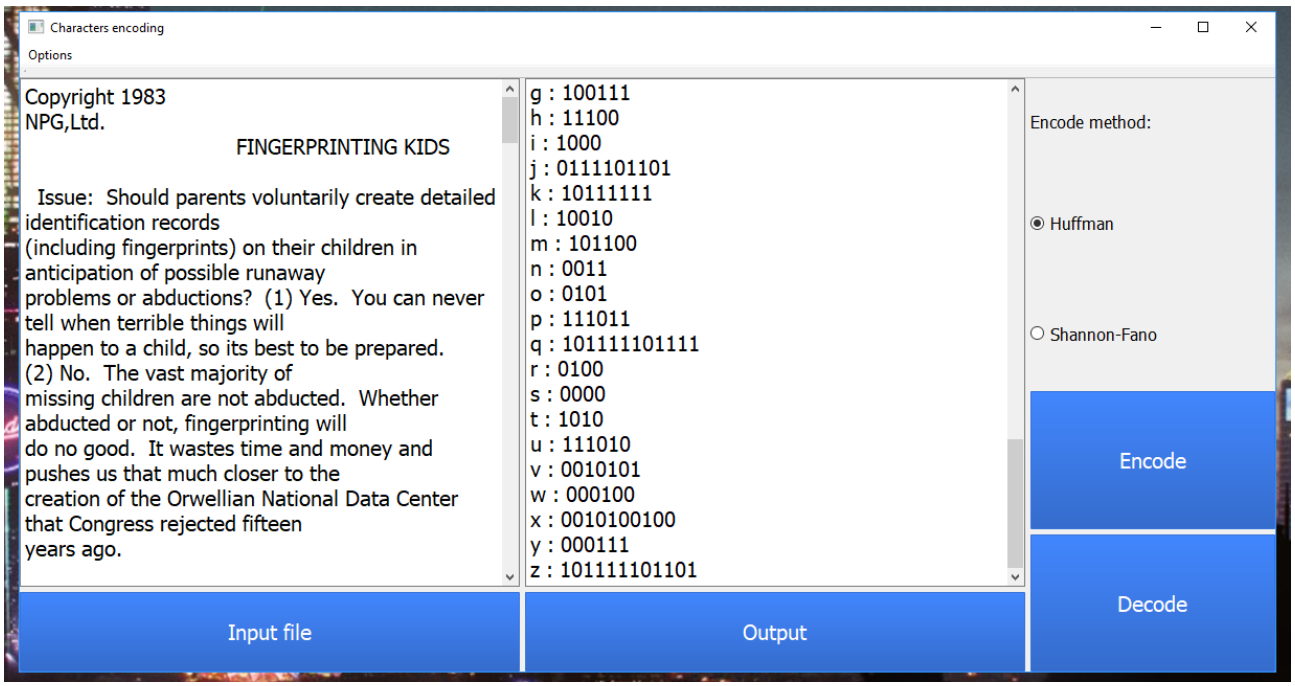

Рисунок 3 – Состояние после кодирования данных

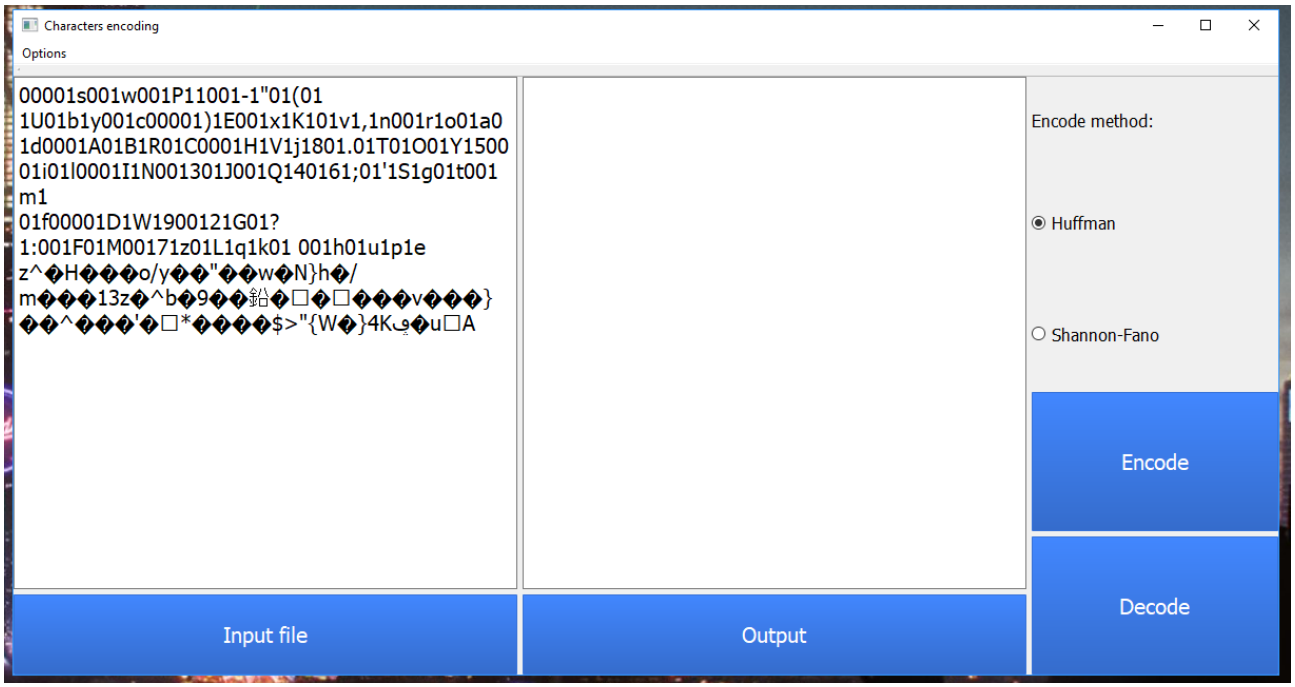

Рисунок 4 – Состояние после открытия файла с закодированными данными

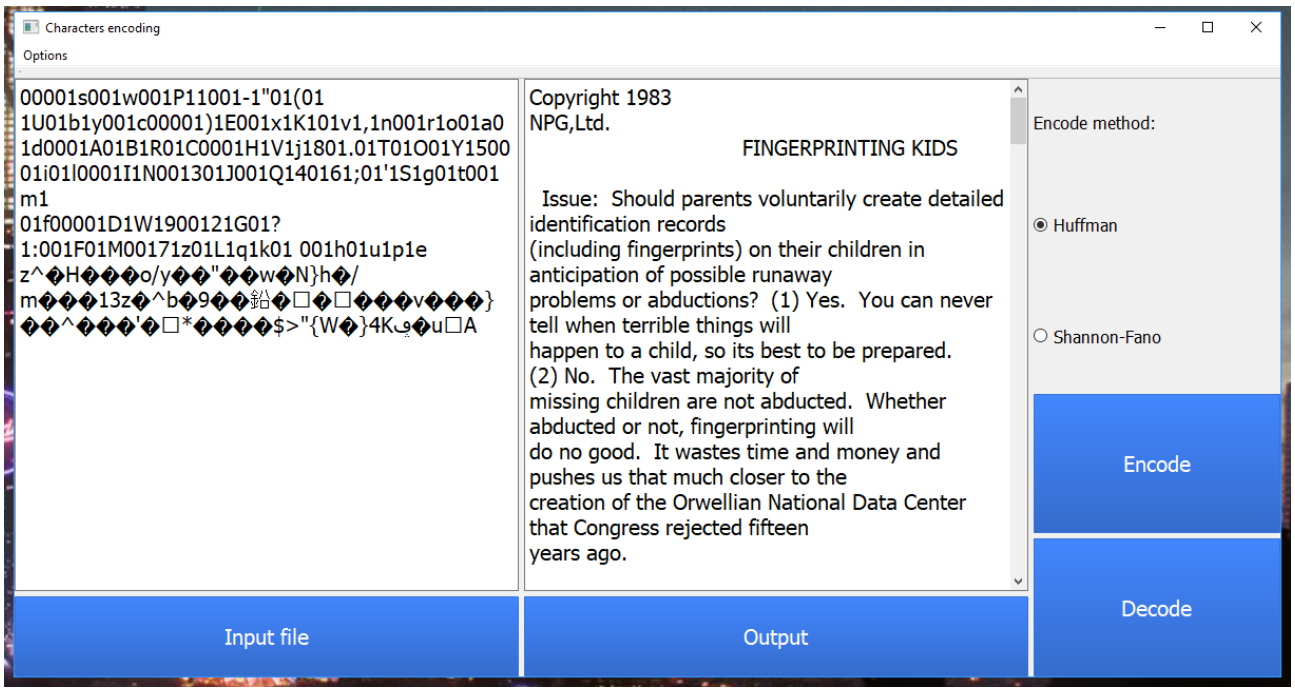

Рисунок 5 – Состояние после декодирования

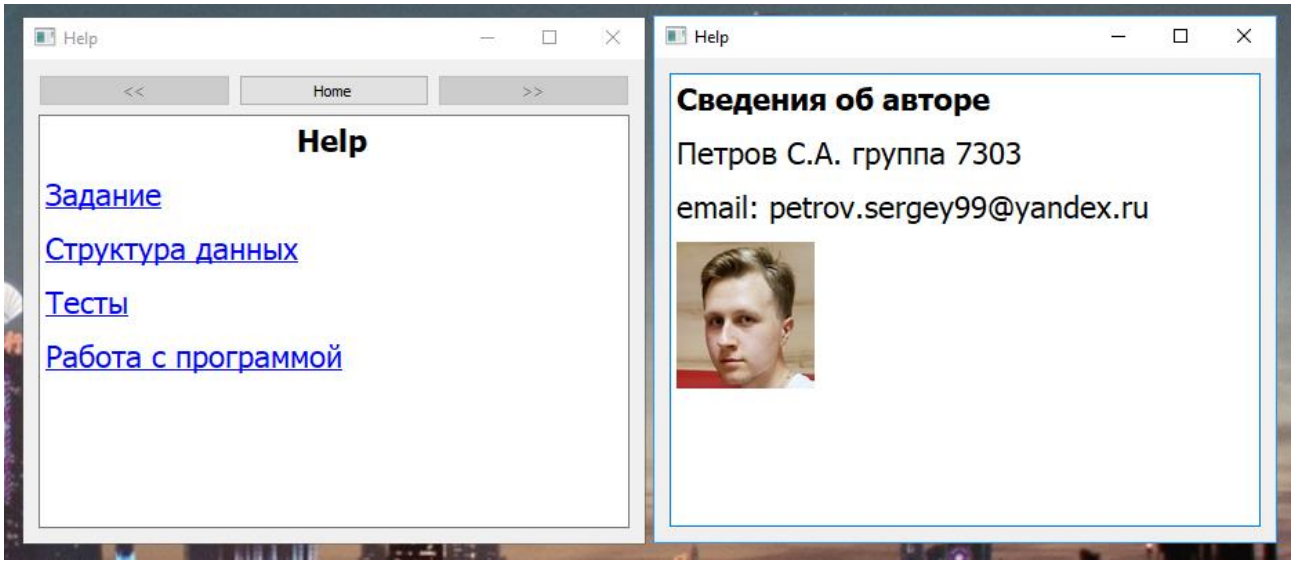

Рисунок 6 – Вызов справки

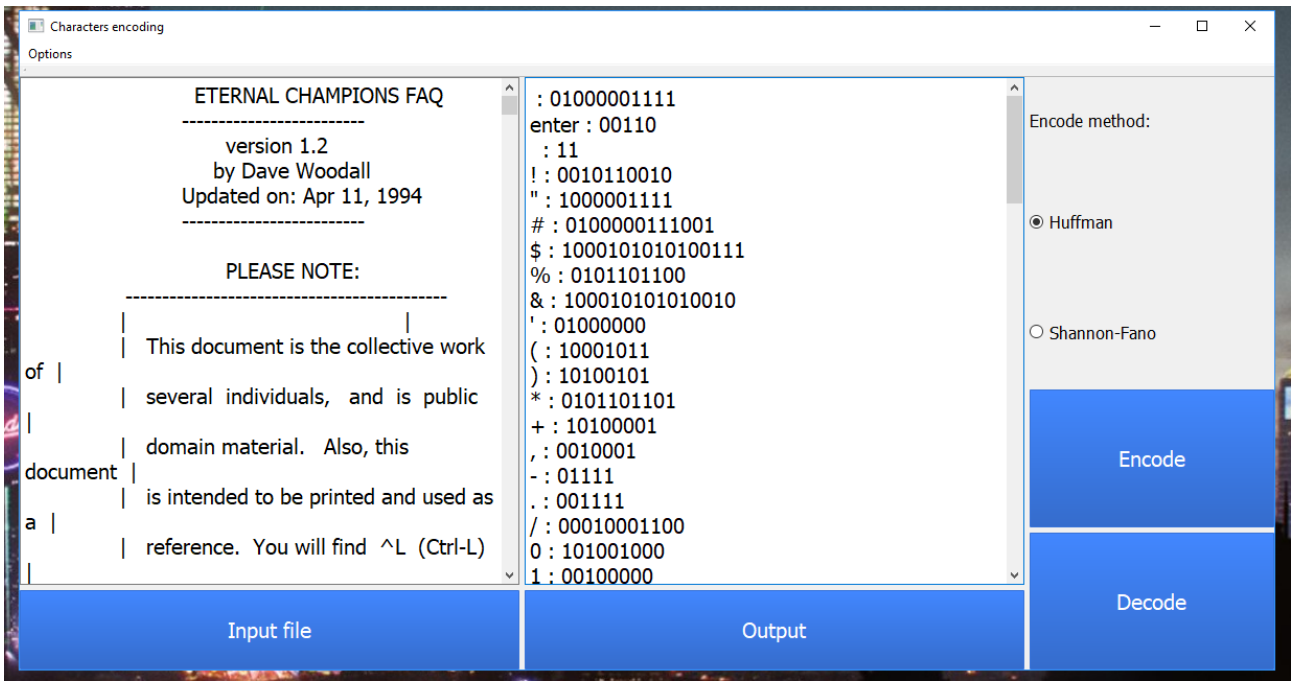

Рисунок 7 – Кодирование файла sample1.txt

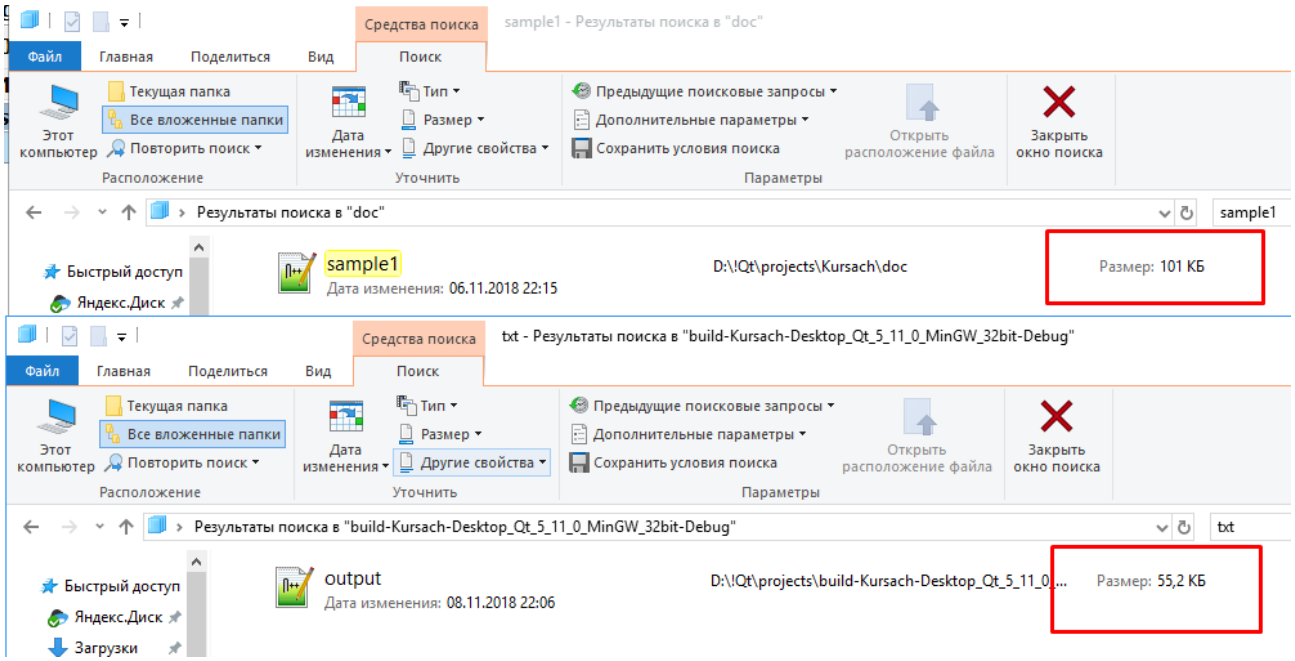

Рисунок 8 – Сравнение размера исходного и закодированного файлов

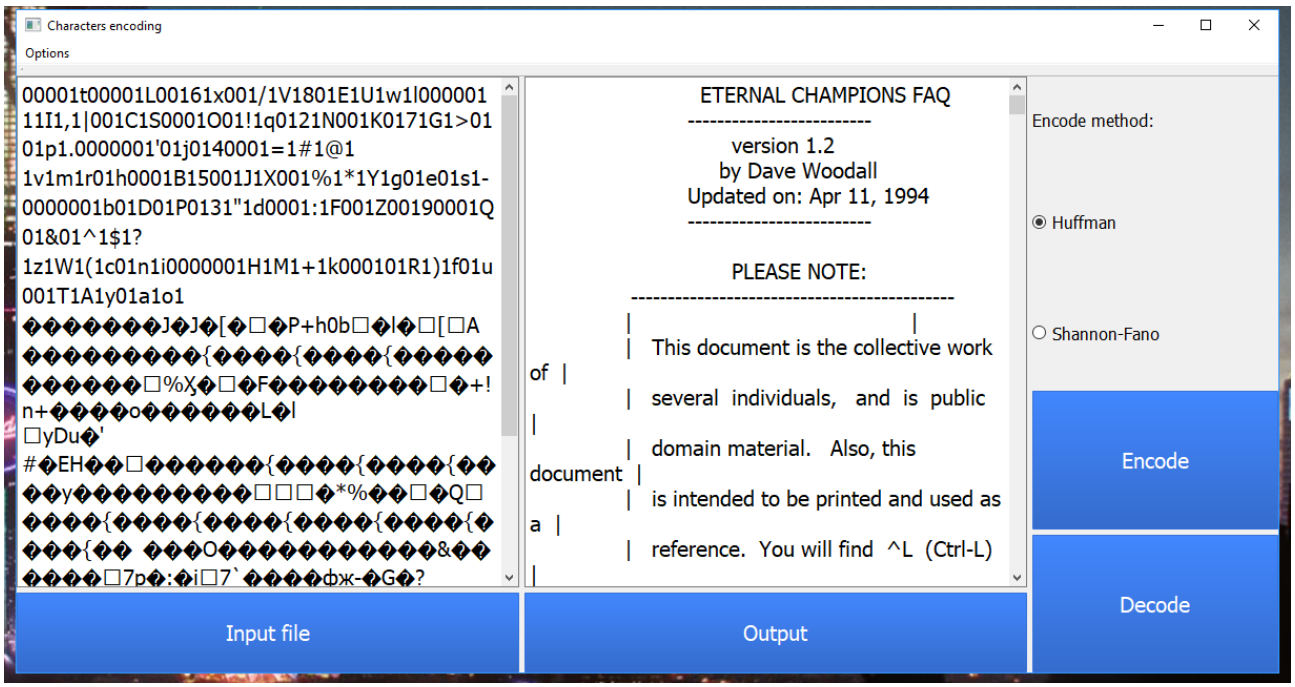

Рисунок 9 – Декодирование файла output.txt

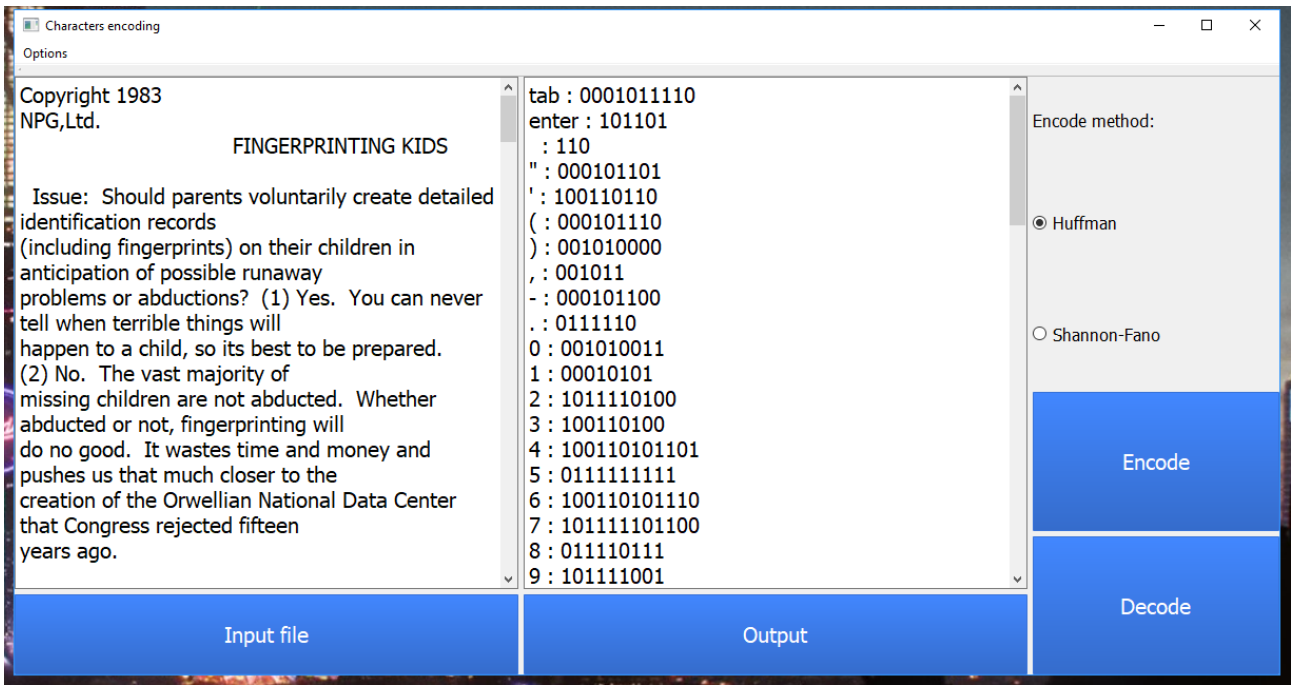

Рисунок 10 – Кодирование файла sample2.txt

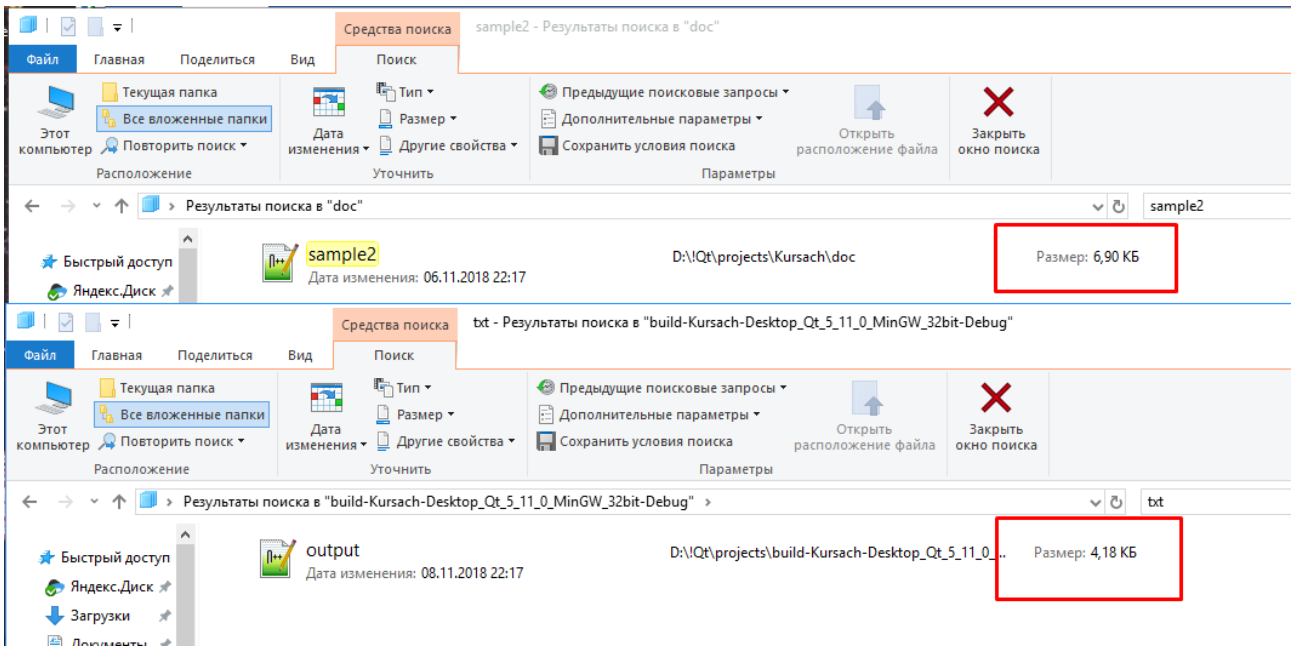

Рисунок 11 – Сравнение размера исходного и закодированного файлов

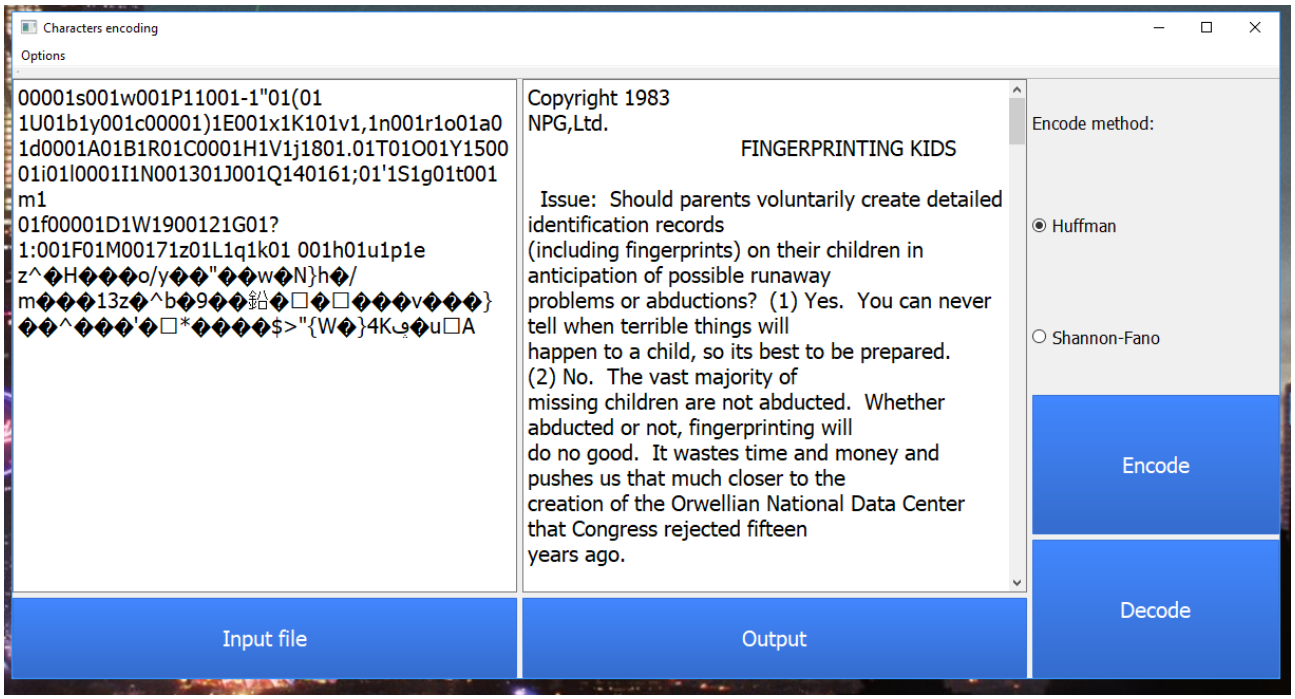

Рисунок 12 – Декодирование файла output.txt

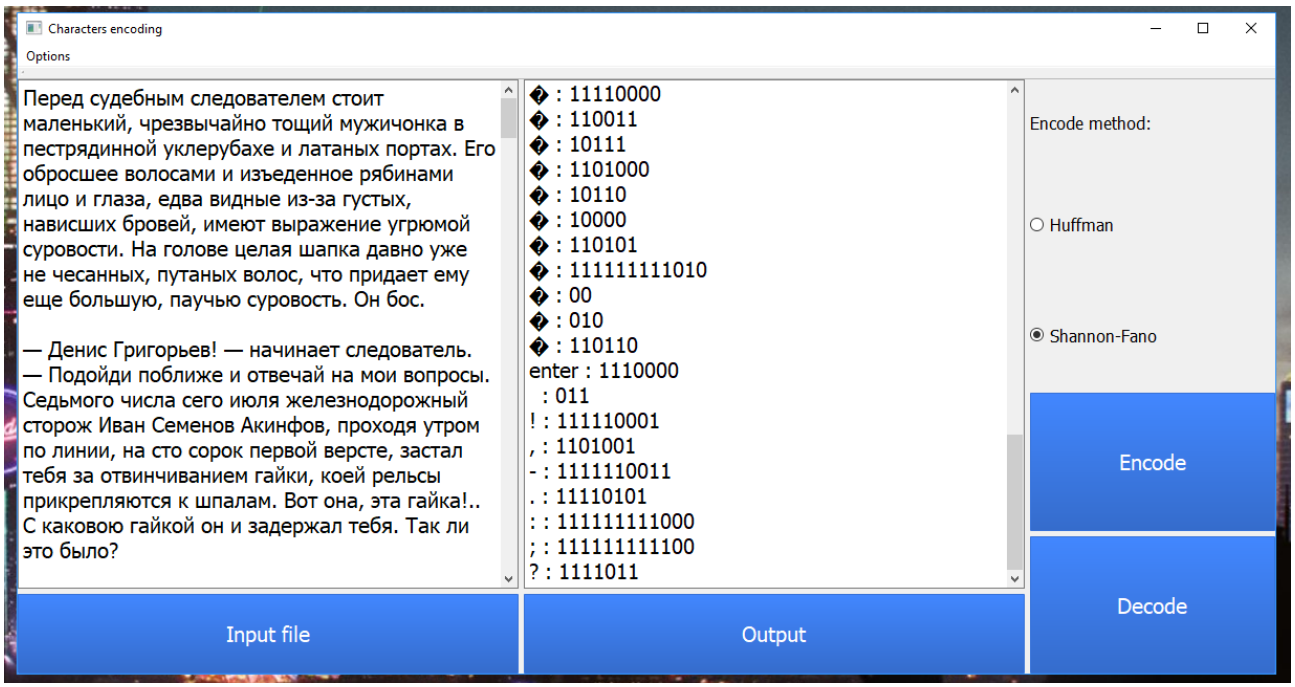

Рисунок 13 – Кодирование файла sample3.txt

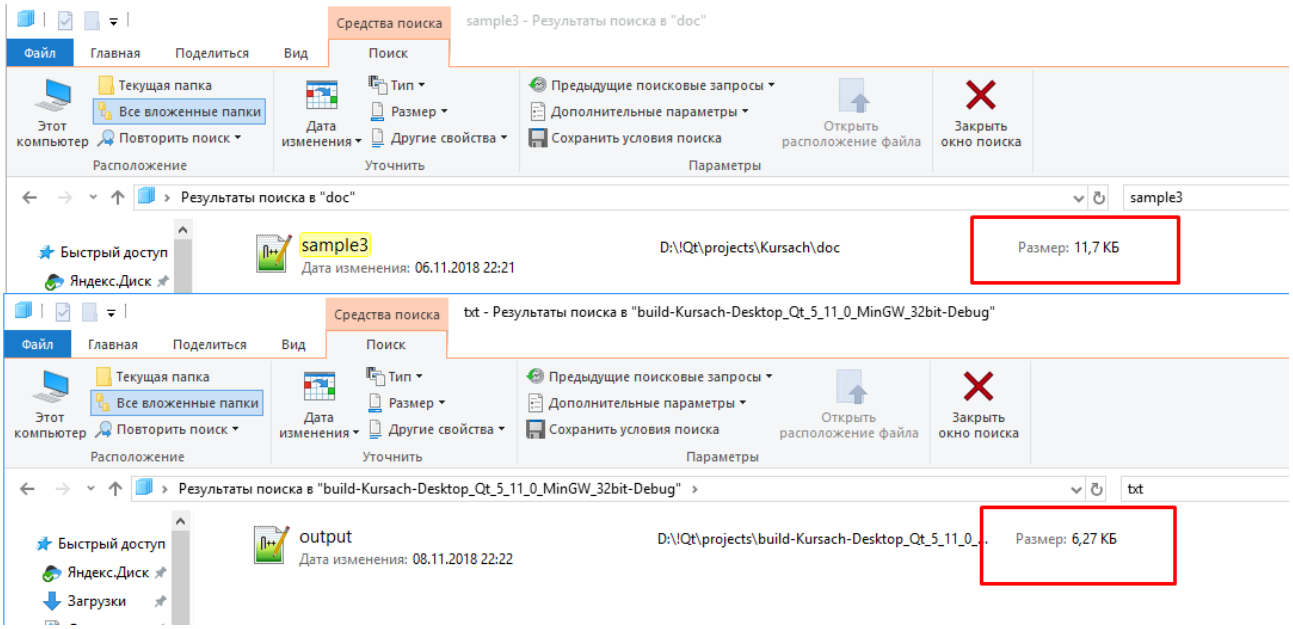

Рисунок 14 – Сравнение размера исходного и закодированного файлов

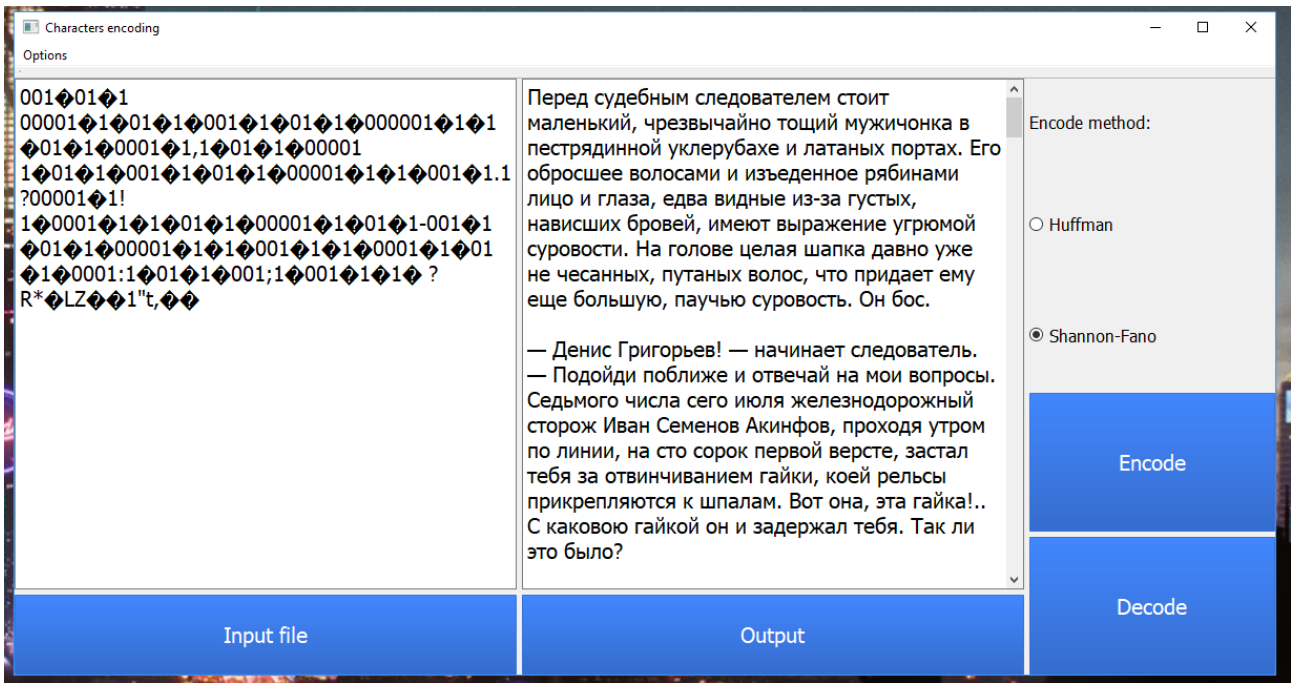

Рисунок 15 – Декодирование файла output.txt

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <algorithm>
#include <string>
#include <map>
#include <vector>
#include <queue>
#include <fstream>
#include <sstream>
#include <QDebug>

class Coder{
private:
    struct Node {
        Node(std::string _chars, int frequence) {
            chars = _chars;
            freq = frequence;
        }

        Node(Node* _left, Node* _right){
            chars[0] = '\0';
            freq = _left->freq + _right->freq;
            left = _left;
            right = _right;
        }

        std::string chars;
        int freq = 0;
        Node* left = nullptr;
        Node* right = nullptr;
        std::string code = "";

        ~Node(){
            delete left;
            delete right;
        }
    };

    struct HuffCompare{
        bool operator()(const Node* first, const Node* second) const{
            return first->freq >= second->freq;
        }
    };

    struct ShanCompare{
        bool operator()(const Node* first, const Node* second) const{
            return first->freq <= second->freq;
        }
    };

public:
    Coder(){}

    ~Coder(){
        delete _root;
    }
```

Продолжение на сл. странице

```
    void shannon_encode(std::string const& file_name){
        clear();
        std::ifstream file;
        file.open(file_name, std::ios::in | std::ios::binary);
        std::string buf;
        while(file.good()){
            std::getline(file, buf);
            _text += buf;
        }
        std::map<char, int> freq;
        for(auto& ch : _text){
            freq[ch]++;
        }
        std::priority_queue<Node*, std::vector<Node*>, ShanCompare>
sortedChars;
        for(auto& node : freq){
            sortedChars.push(new Node(std::string(1, node.first),
node.second));
        }

        std::map<char, std::string> res;
        if(sortedChars.size() == 1){
            auto el = sortedChars.top();
            res[el->chars[0]] = "0";
            delete sortedChars.top();
            _map = res;
            return;
        }

        std::string start;
        while(sortedChars.size() > 0){
            auto el = sortedChars.top();
            start += el->chars;
            sortedChars.pop();
        }
        _root = new Node(start, _text.size());
        makeShannonCodes(_root, freq, res);
        _map = res;
    }

    void huffman_encode(std::string const& file_name){
        clear();
        std::ifstream file;
        file.open(file_name, std::ios::in | std::ios::binary);
        std::string buf;
        while(file.good()){
            std::getline(file, buf);
            _text += buf;
        }
        std::map<char, int> freq;
        for(auto& ch : _text){
            freq[ch]++;
        }
        std::priority_queue<Node*, std::vector<Node*>, HuffCompare>
sortedChars;
        for(auto& node : freq){
            sortedChars.push(new Node(std::string(1, node.first),
node.second));
        }
```

Продолжение на сл. странице

```
        std::map<char, std::string> res;
        if(sortedChars.size() == 1){
            auto el = sortedChars.top();
            res[el->chars[0]] = "0";
            delete sortedChars.top();
            _map = res;
            return;
        }

        while(sortedChars.size() > 1){
            auto left = sortedChars.top();
            sortedChars.pop();
            auto right = sortedChars.top();
            sortedChars.pop();
            sortedChars.push(new Node(left, right));
        }
        _root = sortedChars.top();
        sortedChars.pop();
        makeHuffmanCodes(_root, res);
        _map = res;
    }

    std::string encode_file(std::string const& file_name){
        std::ofstream file;
        file.open(file_name, std::ios::out | std::ios::binary);
        write_tree(_root, file);
        file.write(" ", 1);
        std::string res;
        int count  = 0;
        unsigned char ch = 0;
        for(auto& c : _text){
            std::string code = _map[c];
            res += _map[c];
            for(int i = 0; i < code.size(); i++){
                if(code[i] == '1')
                    ch = ch | (1 << (7 - count));
                count++;
                if(count == 8){
                    file.write(reinterpret_cast<char*>(&ch), 1);
                    ch = 0;
                    count = 0;
                }
            }
        }
        file.close();
        return res;
    }

    std::string decode_file(std::string const& file_name){
        clear();
        std::ifstream file;
        file.open(file_name, std::ios::in | std::ios::binary);
        _root = read_tree(file);
        file.seekg(1, std::ios::cur);
        unsigned char ch;
        int count = 0;
        bool b;
        Node* r = _root;
        std::string res;
```

Продолжение на сл. странице

```
        file.read(reinterpret_cast<char*>(&ch), 1);
        while(file.good()){
            b = ch & (1 << (7 - count));
            if(b) r = r->right;
            else r = r->left;
            if(!r->left && !r->right){
                res += r->chars;
                r = _root;
            }
            count++;
            if(count == 8){
                count = 0;
                file.read(reinterpret_cast<char*>(&ch), 1);
            }
        }
        file.close();
        return res;
    }

    std::map<char, std::string> get_map() const{
        return _map;
    }

private:


    void write_tree(Node* node, std::ofstream& file){
        if(!node->left && !node->right){
            file.write("1", sizeof(char));
            file.write(reinterpret_cast<char*>(&node->chars[0]),
sizeof(char));
        } else {
            file.write("0", sizeof(char));
            write_tree(node->left, file);
            write_tree(node->right, file);
        }
    }

    Node* read_tree(std::ifstream& file){
        unsigned char ch;
        file.read(reinterpret_cast<char*>(&ch), 1);
        if(ch == '1'){
            file.read(reinterpret_cast<char*>(&ch), 1);
            return new Node(std::string(1, ch), 0);
        } else {
            Node* left = read_tree(file);
            Node* right = read_tree(file);
            return new Node(left, right);
        }
    }

    void clear(){
        _text.clear();
        _map.clear();
        if(_root){
            delete _root;
        }
    }
```

Продолжение на сл. странице

```
    void makeShannonCodes(Node* node, std::map<char, int>& freq,
std::map<char, std::string>& map, std::string code=""){
        if (node->chars.size() == 1) map[node->chars[0]] = code;
        if(node->chars.size() > 1) {
            int startPos = 0;
            int first_part = 0;
            int total_freq = node->freq;
            for (startPos = 0; startPos < node->chars.size(); startPos++) {
                if (first_part >= (total_freq - first_part) || (startPos + 1
== node->chars.size())) break;
                first_part += freq[node->chars[startPos]];
            }
            Node* left = new Node({node->chars.begin(), node->chars.begin() +
startPos}, first_part);
            Node* right = new Node({node->chars.begin() + startPos, node-
>chars.end()}, total_freq - first_part);
            node->left = left;
            node->right = right;
            makeShannonCodes(node->left, freq, map, code+"0");
            makeShannonCodes(node->right, freq, map, code+"1");
        }
    }

    void makeHuffmanCodes(Node* node, std::map<char, std::string>& map,
std::string code=""){
        if (!node) return;
        if (node->chars[0] != '\0') map[node->chars[0]] = code;
        makeHuffmanCodes(node->left, map, code+"0");
        makeHuffmanCodes(node->right, map, code+"1");
    }

    Node* _root = nullptr;
    std::map<char, std::string> _map;
    std::string _text = "";
};
```