МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Статическое кодирование и декодирование

Студент гр. 9303 _____ Низовцов Р. С.

Преподаватель _____ Филатов Ар. Ю.

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Низовцов Р. С.

Группа 9303

Тема работы : Статическое кодирование и декодирование методамы Хаффмана и Фано-Шеннона - демонстрация

Исходные данные:

Текст в файле / текст, введенный с клавиатуры

Содержание пояснительной записки:

«Аннотация», «Содержание», «Введение», «Основные теоретические положения», «Описание программного кода», «Описание интерфейса пользователя», «Заключение», «Исходный код программы», «Тестирование программы»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 24.12.2020

Дата защиты реферата: 25.12.2020

Студент _____ Низовцов Р. С.

Преподаватель _____ Филатов Ар. Ю.

АННОТАЦИЯ

Курсовая работа представляет собой программу, предназначенную для демонстрация кодирования и декодирования строй методами Хаффмана и Фано-Шеннона. Код программы написан на языке программирования C++, запуск программы подразумевается на операционных системах семейства Linux. При разработке кода программы активно использовались функции стандартных библеотек языка C++, основные управляющий конструкции языка C++. Для работоспособности программы проводилось тестирование. Исходный код, скриншоты, показывающие корректную работу программы, и результаты тестирования представлены в приложениях.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы — разработка программы для демонстрации задач на кодирование и декодирование латинских строк методами Хаффмана и Фано-Шеннона. Также для наиболее удобного взаимодействие с программой был создан графический интерфейс, с использованием Qt.

Для достижения поставленной цели требуется реализовать следующие задачи:

1. Изучение теоретического материала по написанию кода на языке C++ и об алгоритмах Фано-Шеннона и Хаффмана.
2. Разработка программного кода в рамках полученного задания.
3. Написание программного кода.
4. Тестирование программного кода.

   Полученное задание:

   Статическое кодирование и декодирование текстового файла методами Хаффмана и Фано-Шеннона — демонстрация.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

1.1. Основные теоретические положения об алгоритме Хаффмана.

Алгоритм Хаффмана — алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 195 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при написании им курсовой работы. В настоящее время используется во многих программах сжатия данных.

Один из первых алгоритмов эффективного кодирования информации был предложен Д. А. Хаффманом в 1952 году. Идея алгоритма состоит в следующем: зная вероятности появления символов в сообщении, можно описать процедуру построения кодов переменной длины, состоящих из целого количества битов. Символам с большей вероятностью ставятся в соответствие более короткие коды. Коды Хаффмана обладают свойством префиксности (то есть ни одно кодовое слово не является префиксом другого), что позволяет однозначно их декодировать.

Классический алгоритм Хаффмана на входе получает таблицу частот встречаемости символов в сообщении. Далее на основании этой таблицы строится дерево кодирования Хаффмана (Н-дерево).

1. Символы входного алфавита образуют список свободных узлов. Каждый лист имеет вес, который может быть равен либо вероятности, либо количеству вхождений символа в сжимаемое сообщение.
2. Выбираются два свободных узла дерева с наименьшими весами.
3. Создается их родитель с весом, равным их суммарному весу.
4. Родитель добавляется в список свободных узлов, а два его потомка удаляются из этого списка.
5. Одной дуге, выходящей из родителя, ставится в соответствие бит 1, другой — бит 0. Битовые значения ветвей, исходящих от корня, не зависят от весов потомков.

6. Шаги, начиная со второго, повторяются до тех пор, пока в списке свободных узлов не останется только один свободный узел. Он и будет считаться корнем дерева.

1.2. Основные теоретические положения об алгоритме Фано-Шеннона.

Алгоритм Шеннона — Фано — один из первых алгоритмов сжатия, который впервые сформулировали американские учёные Клод Шеннон и Роберт Фано. Данный метод сжатия имеет большое сходство с алгоритмом Хаффмана, который появился на несколько лет позже и является логическим продолжением алгоритма Шеннона. Алгоритм использует коды переменной длины: часто встречающийся символ кодируется кодом меньшей длины, редко встречающийся — кодом большей длины. Коды Шеннона — Фано — префиксные, то есть никакое кодовое слово не является префиксом любого другого. Это свойство позволяет однозначно декодировать любую последовательность кодовых слов.

Основные этапы:

1. Символы первичного алфавита m_1 выписывают по убыванию вероятностей.
2. Символы полученного алфавита делят на две части, суммарные вероятности символов которых максимально близки друг другу.
3. В префиксном коде для первой части алфавита присваивается двоичная цифра «0», второй части — «1».
4. Полученные части рекурсивно делятся, и их частям назначаются соответствующие двоичные цифры в префиксном коде.

2. ОПИСАНИЕ ПРОГРАММНОГО КОДА

2.1. Релизованные методы

Были реализованы следующие методы, не относящиеся к интерфейсу программы:

- `readFile(const char*)` – Метод, который считывает строку из предложенного файла.
- `startCoding()` - Метод, который начинает реализацию алгоритма, выбранного пользователем
- `stringAnalys()` - Метод, который анализирует строку, считает количество вхождений символов в строку
- `stringSort()` и `stringSort(int, int*, string*)` - Метод, который сортирует массив символов по количеству вхождений в строку
- `searchTreeFanoShenon(char, string&, int, int, Qstring&, int&, int, Qstring&)` - Метод, который реализует алгоритм Фано-Шеннона
- `searchTreeHaffman(int, Qstring&, Qstring&)` - Метод, который реализует алгоритм Хаффмана
- `stringCoder()` - Метод, который кодирует сообщение
- `stringDecoder()` - Метод, который декодирует сообщение

2.2. Реализованные поля

Были реализованы следующие поля, не относящиеся к интерфейсу программы:

- `isDecoding` – Поле, показывающее: ведется ли работа по декодирования
- `isCoding` – Поле, показывающее: ведется ли работа по кодированию
- `isCodeMaking` – Поле, показывающее: ведется ли работа по созданию шифра
- `algorithm` – Поле, хранящее использованый алгоритм

- `line` – Поле, которое хранит в себе исходную строку
- `symbols` – Поле, которое хранит все встречающиеся символы в исходной строке
- `freq` – Поле, которое хранит массив количества вхождений каждого символа в исходную строку
- `cipher` – Поле, которое хранит в себе шифр каждого символа, встречающегося в тексте
- `codedMessage` – Поле, которое хранит в себе закодированную строку
- `decodedMessage` – Поле, которое хранит в себе декодированную строку

2.3. Вспомогательные методы

Были реализованы следующие вспомогательные методы, не относящиеся к интерфейсу программы и алгоритмам:

- `stringCheck()` - Метод, который заменяет все специальные символы синтаксиса DOT в строке на пробелы
- `isCloseCheck()` - Метод, который контролирует возвращение всех полей в стандартное положение значений
- `closeProgramm()` - Метод для закрытия программы

3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

3.1. Общие сведения

При запуске приложения предлогается ввести строку через клавиатуру или выбрать файл для считывания. Поле этого становятся доступны кнопки «Следующий шаг» и «Пропустить шаг». С помощью них реализована демонстрация работы алгоритма и дальнейшее кодирование/декодирование сообщения. В любой момент выполнения программы есть возможность заменить входную строку или закончить выполнение программы.

3.2. Реализация графического интерфейса

Были реализованы следующие методы, относящиеся к интерфейсу программы:

- `loadFile()` - Метод, который реализует прочтение первой строки из файла и начало её дальнейшей обработки
- `loadImage()` - Метод, который реализует открытие и вывод png файла, созданного утилитой GraphViz
- `fullsize_decorator()` - Метод, который отображает изображение в исходном размере
- `decorator()` - Метод, который выравнивает изображение в рамках его виджета
- `checkClick()` - Метод, связанный с кнопкой. Он контролирует все процессы, именно через него происходит все взаимодействие пользователя с программой
- `checkFastClick()` - Метод, схожий с checkClick(). Является его копией, которая моментально переходит к следующей логической части алгоритма
- `inputText()` - Метод, который принимает строку, введенную с клафиатуры пользователем для дальнейшей обработки

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы была реализована программа с графическим интерфейсом, которая позволяет проводить демонстрацию средистудентов по теме «Статическое кодирование и декодирование методамы Хаффмана и Фано-Шеннона». Таким образом, результат полностью соответствует поставленной цели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Язык программирования СИ / Керниган Б., Ритчи Д. СПб.: Издательство "Невский Диалект", 2001. 352 с.

2. Основы программирования на языках Си и C++ [Электронный ресурс URL: http://cplusplus.com

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

MAINWINDOW.H:

```
#IFNDEF MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "scaledpixmap.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <cmath>
#include <string>
#include <QFileDialog>
#include <QMessageBox>
#include <QInputDialog>


using namespace std;

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    QLabel* text;
    ScaledPixmap* Image_widget;
    int image_width = 476;
    int image_height = 476;
    bool isDecoding = false;
    bool isCoding = false;
    bool isCodeMaking = true;

    QString algorithm;
    string line;
    string symbols;
    int* freq = nullptr;
    string* cipher = nullptr;
    int symbols_count = 0;
    QString codedMessage;
    QString decodedMessage;
```

```
    void readFile(const char*);
    void startCoding();
    void stringAnalys();
    void stringSort();
    void stringSort(int, int*, string*);
    void searchTreeFanoShenon(char, string &, int, int, QString&,
int&, int, QString&);
    bool searchTreeHaffman(int, QString&, QString&);
    bool stringCoder();
    bool stringDecoder();

    void loadFile();
    void loadImage();
    void stringCheck();
    void isCloseCheck();
    void closeProgramm();
    void fullsize_decorator();
    void decorator();
    void checkClick();
    void checkFastClick();
    void inputText();
};
#endif // MAINWINDOW_H
```

SCALEDPIXMAP.H:

```
#IFNDEF SCALEDPIXMAP_H
#define SCALEDPIXMAP_H

#include <QLabel>
#include <QPainter>
#include <QMouseEvent>
#include <QScrollArea>

class ScaledPixmap : public QWidget {

public:
  ScaledPixmap(QWidget *parent = 0);
  void setScaledPixmap(const QPixmap &pixmap);
  void deleteScaledPixmap();
  void removeScaledPixmap();
  QPoint resizeWidget();
  void resetMouse();
  QPoint backMouse(int count);

protected:
  void paintEvent(QPaintEvent *event);

private:
```

```
  int pixmap_width = 0;
  int pixmap_height = 0;
  int pixmap_x_start = 0;
  int pixmap_y_start = 0;
  QPoint press_point;
  QPoint release_point;
  QPixmap m_pixmap;
  void mousePressEvent(QMouseEvent *event);
  void mouseReleaseEvent(QMouseEvent *event);
};

#endif // SCALEDPIXMAP_H
```

MAIN.CPP:

```
#INCLUDE "MAINWINDOW.H"


#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

MAINWINDOW.CPP:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    connect(ui->Beauty_button, &QAction::triggered, this,
&MainWindow::decorator);
    connect(ui->Fullsize_button, &QAction::changed, this,
&MainWindow::fullsize_decorator);
    connect(ui->Open_button, &QAction::triggered, this,
&MainWindow::loadFile);
    connect(ui->Open, &QPushButton::clicked, this,
&MainWindow::loadFile);
    connect(ui->Close_button, &QAction::triggered, this,
&MainWindow::closeProgramm);
    connect(ui->Next_turn, &QPushButton::clicked, this,
&MainWindow::checkClick);
```

```
    connect(ui->Turn_over, &QPushButton::clicked, this,
&MainWindow::checkFastClick);
    connect(ui->Write, &QPushButton::clicked, this,
&MainWindow::inputText);
    connect(ui->Write_button, &QAction::triggered, this,
&MainWindow::inputText);
    ui->Next_turn->setDisabled(true);
    ui->Turn_over->setDisabled(true);
    ui->explanation->setAlignment(Qt::AlignCenter);
    ui->explanation->setStyleSheet("font: 17px Times New Roman;");
    Image_widget = new ScaledPixmap;
    ui->scrollArea->setWidget(Image_widget);
    ui->Beauty_button->setDisabled(true);
    ui->Fullsize_button->setDisabled(true);
}

MainWindow::~MainWindow()
{
    isCloseCheck();
    system("rm ./result.txt");
    system("rm ./result.png");
    delete ui->scrollArea->takeWidget();
    delete ui;
}

void MainWindow::readFile(const char* path){
    ifstream inFile(path);   //Чтение строки из файлы, открытие
файла
    if (!inFile.is_open()){
        QMessageBox::warning(this, "Ошибка", "Не удалось прочитать
файл:\nошибка доступа или неверный путь");
        return;
    }

    getline(inFile, line);
    transform(line.begin(), line.end(),
line.begin(), ::tolower); //Перевод строки в нижний регистр
    stringCheck();
    ui->Next_turn->setEnabled(true);
    ui->Turn_over->setEnabled(true);
    inFile.close();
    codedMessage = QString::fromStdString(line);
    startCoding();
}

void MainWindow::startCoding(){
    size_t pos;
    stringAnalys();      //Функция для расчета количества
вхождений символов
    cipher = new string[symbols.length()];
    stringSort();        //Функция для сортировки этих вхождений
по убыванию
    pos = symbols.find(' ');
```

```
    if(pos != string::npos)
        symbols[pos] = '_';    //Замена пробелов на символ "_" для
удобства
    QString outInfo; outInfo+="graph graphname{\n";
    for(int i = 0; i < symbols.length(); i++){
        outInfo+= '\t' + QString::number(i) + " [label=\"" +
QString::fromStdString(symbols.substr(i, 1)) + '(' +
QString::number(freq[i]) + ")\"]\n";
    }
    outInfo+='}';
    ofstream outFile("./result.txt");
    outFile << outInfo.toStdString();
    outFile.close();
    system("dot -Tpng ./result.txt -o result.png");   //Показ
массива символов
    loadImage();
    ui->explanation->setText("Буквы, которые\nвстречаются в
тексте,\nрасположены по\nубыванию вхождений\n(в скобках указано\
nколичество вхождений\nбукв в тексте)\n\nПостроим дерево\nдля
кодирования\nсообщения");
    this->setDisabled(true);
    QStringList items;
    items << tr("Фано-Шеннона") << tr("Хаффмана");
    bool ok;
    algorithm = QInputDialog::getItem(this, tr("NotMainWindow"),
tr("Выбор алгоритма:"), items, 0, false, &ok);  //Выбор
пользователем алгоритма
    this->setEnabled(true);
    if(!ok)
        algorithm.clear();
    isCodeMaking = true;
}

void MainWindow::stringAnalys(){   //Расчет вхождений символов в
строку
    int size = 0;
    int start_size = line.length();
    size_t pos;
    freq = new int[size];
    for(int i = 0; i < start_size; i++){
        pos = symbols.find(line[i]);
        if(pos == string::npos){
            symbols+=line[i];
            int *freq1 = new int[++size];
            for(int j = 0; j < size - 1; j++)
                freq1[j] = freq[j];
            freq1[size-1] = 1;
            delete[] freq;
            freq = freq1;
        }
        else{
            freq[pos]++;
        }
```

```
    }
}

void MainWindow::stringSort(){    //Сортировка символов и их
вхождений
    int n = symbols.length();
    int temp = 0;
    char ctemp;
    for (int k = 0; k < n; k++){
        for (int j = k % 2; j + 1 < n; j += 2){
            if (freq[j] < freq[j + 1]){
                temp = freq[j];
                freq[j] = freq[j + 1];
                freq[j + 1] = temp;
                ctemp = symbols[j];
                symbols[j] = symbols[j + 1];
                symbols[j + 1] = ctemp;
            }
        }
    }
}

void MainWindow::stringSort(int size, int* freqHaff, string*
symbolsHaff){
    int temp = 0;
    string ctemp;
    for (int k = 0; k < size; k++){
        for (int j = k % 2; j + 1 < size; j += 2){
            if (freqHaff[j] < freqHaff[j + 1]){
                temp = freqHaff[j];
                freqHaff[j] = freqHaff[j + 1];
                freqHaff[j + 1] = temp;
                ctemp = symbolsHaff[j];
                symbolsHaff[j] = symbolsHaff[j + 1];
                symbolsHaff[j + 1] = ctemp;
            }
        }
    }
}

void MainWindow::searchTreeFanoShenon(char lastCodeElem, string&
code, int start, int end, QString& outInfo, int& count, int aim,
QString& comment)
{
    double totalSum = 0;   //Сумма вхождений изначальной части
    int i, currSum = 0;
    string currCode = "";  //Строка для кодировки текущей части
строки
    if(lastCodeElem != '\0')
        currCode = code + lastCodeElem;
    else
        currCode = code;
```

```
    if (start==end)  //Случай конечного определения кодировки
текущего символа
    {
        cipher[start] = currCode;
        comment+='\n'; comment+=QString::fromStdString(symbols)
[start] + " -- " + QString::fromStdString(currCode);
        return;
    }
    for (i=start;i<=end;i++)  //Расчет суммы вхождений
        totalSum+=freq[i];
    totalSum/=2;
    i=start+1;
    currSum +=freq[start];
    while (fabs(totalSum-(currSum+freq[i])) < fabs(totalSum-
currSum) && (i<end))  //Деление части строки на две с учетом суммы
вхождений каждой части
    {
        currSum+=freq[i];
        i++;
    }

    if(count > aim){
        if(aim==symbols_count && count > aim)
            symbols_count = count;
        return;
    }

    outInfo.remove(outInfo.length()-1,1);      //Создание
структуры для обработки утилиты GrsphViz
    outInfo+='\t' + QString::fromStdString(symbols.substr(start,
i-start)) + " [label=\"" +
QString::fromStdString(symbols.substr(start, i-start)) + '(' +
QString::number(currSum) + ")\"]\n";
    outInfo+='\t' + QString::fromStdString(symbols.substr(i,
end+1-i)) + " [label=\"" +
QString::fromStdString(symbols.substr(i, end+1-i)) + '(' +
QString::number(totalSum*2-currSum) + ")\"]\n";
    outInfo+='\t' + QString::fromStdString(symbols.substr(start,
end+1-start)) + " -- " +
QString::fromStdString(symbols.substr(start, i-start)) + "
[label=0]\n";
    outInfo+='\t' + QString::fromStdString(symbols.substr(start,
end+1-start)) + " -- " + QString::fromStdString(symbols.substr(i,
end+1-i)) + " [label=1]\n";
    outInfo+='}';

    count++;
    searchTreeFanoShenon('0', currCode , start, i-1, outInfo,
count, aim, comment);
    count++;
    searchTreeFanoShenon('1', currCode , i, end, outInfo, count,
aim, comment);
}
```

```
bool MainWindow::searchTreeHaffman(int aim, QString& outInfo,
QString& comment){
    if(cipher!=nullptr){     //Изначальное создание
вспомогательных полей
        delete[] cipher;
    }
    cipher = new string[symbols.length()];
    string* symbolsHaff = new string[symbols.length()];
    int* freqHaff = new int[symbols.length()];
    for(int i = 0; i < symbols.length(); i++){
        symbolsHaff[i] = symbols[i];
        freqHaff[i] = freq[i];
    }
    int size = symbols.length();
    outInfo+="graph graphname{\n";
    for(int i = 0; i < symbols.length(); i++){
        outInfo+= '\t'; outInfo+= QString::fromStdString(symbols)
[i] + " [label=\""; outInfo+= QString::fromStdString(symbols)[i];
outInfo+= '(' + QString::number(freq[i])+ ")\"]\n";
    }
    for(int i = 0; i < aim-1 && size != 1; i++){   //Возобновление
информации, существующая до текущего шага(aim)
        outInfo+= '\t' + QString::fromStdString(symbolsHaff[size-
2]) + QString::fromStdString(symbolsHaff[size-1]) + " [label=\"" +
QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]); outInfo+= '(' +
QString::number(freqHaff[size-2]+freqHaff[size-1])+ ")\"]\n";
        outInfo+='\t' + QString::fromStdString(symbolsHaff[size-
2]) + " -- " + QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]) + " [label=0]\n";
        outInfo+='\t' + QString::fromStdString(symbolsHaff[size-
1]) + " -- " + QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]) + " [label=1]\n";
        for(int j = 0; j < symbolsHaff[size-2].length(); j++)
            cipher[symbols.find(symbolsHaff[size-2].substr(j, 1))]
= '0' + cipher[symbols.find(symbolsHaff[size-2][j])];
        for(int j = 0; j < symbolsHaff[size-1].length(); j++)
            cipher[symbols.find(symbolsHaff[size-1].substr(j, 1))]
= '1' + cipher[symbols.find(symbolsHaff[size-1][j])];
        freqHaff[size-2]+=freqHaff[size-1];
        symbolsHaff[size-2]+=symbolsHaff[size-1];
        stringSort(size, freqHaff, symbolsHaff);
        size--;
    }
    if(size == 1){
        outInfo+='}';
        return true;
    }
    comment+=QString::fromStdString(symbolsHaff[size-2]) + "\n+\n"
+ QString::fromStdString(symbolsHaff[size-1]) + "\n=\n" +
QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]);
```

```
    outInfo+= '\t' + QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]) + " [label=\"" +
QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]); outInfo+= '(' +
QString::number(freqHaff[size-2]+freqHaff[size-1])+ ")\"]\n";

    outInfo+='\t' + QString::fromStdString(symbolsHaff[size-2]) +
" -- " + QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]) + " [label=0]\n";
    outInfo+='\t' + QString::fromStdString(symbolsHaff[size-1]) +
" -- " + QString::fromStdString(symbolsHaff[size-2]) +
QString::fromStdString(symbolsHaff[size-1]) + " [label=1]\n";
    for(int j = 0; j < symbolsHaff[size-2].length(); j++)
        cipher[symbols.find(symbolsHaff[size-2].substr(j, 1))] =
'0' + cipher[symbols.find(symbolsHaff[size-2][j])];
    for(int j = 0; j < symbolsHaff[size-1].length(); j++)
        cipher[symbols.find(symbolsHaff[size-1].substr(j, 1))] =
'1' + cipher[symbols.find(symbolsHaff[size-1][j])];
    freqHaff[size-2]+=freqHaff[size-1];
    symbolsHaff[size-2]+=symbolsHaff[size-1];
    stringSort(size, freqHaff, symbolsHaff);
    size--;
    outInfo+='}';
    if(size == 1)
        return true;
    return false;
}

bool MainWindow::stringCoder(){
    int count = 1;
    size_t pos;
    int size = codedMessage.length();
    for(int i = 0; i < size; i++){ //Кодирование первого символа
не 0 и не 1, /n вставляется каждые ~50 символов
        if(codedMessage[i] == '\n'){
            count++;
            continue;
        }

        if(codedMessage[i] == ' '){
            ui->explanation->setText("Кодируем, заменяя:\n_ >>>
"+QString::fromStdString(cipher[symbols.find('_')]));
            if(i > 50*count)
                codedMessage = codedMessage.mid(0, i) +
QString::fromStdString(cipher[symbols.find('_')]) + '\n' +
codedMessage.mid(i+1, codedMessage.length()-i-1);
            else
                codedMessage = codedMessage.mid(0, i) +
QString::fromStdString(cipher[symbols.find('_')]) +
codedMessage.mid(i+1, codedMessage.length()-i-1);
            return false;
        }
        pos = symbols.find(codedMessage.toStdString()[i]);
```

```
        if(pos != string::npos){
            ui->explanation->setText("Кодируем, заменяя:\
n"+codedMessage[i]+" >>> "+QString::fromStdString(cipher[pos]));
            if(i > 50*count)
                codedMessage = codedMessage.mid(0, i) +
QString::fromStdString(cipher[pos]) + '\n' + codedMessage.mid(i+1,
codedMessage.length()-i-1);
            else
                codedMessage = codedMessage.mid(0, i) +
QString::fromStdString(cipher[pos]) + codedMessage.mid(i+1,
codedMessage.length()-i-1);
            return false;
        }
    }
    return true;
}

bool MainWindow::stringDecoder(){
    int symbCounter = 0;    //Декодирует первую чассть шифра,
игнорирует \n
    for(int i = 0; i < decodedMessage.length(); i++){
        if(decodedMessage[i] != '0' && decodedMessage[i] != '1'){
            if(decodedMessage[i] == '\n'){
                decodedMessage.remove(i, 1);
                continue;
            }
            symbCounter++;
        }
        else
            break;
    }
    if(decodedMessage.length() == symbCounter)
        return true;

    int idCounter = symbCounter;
    QString code; code+=decodedMessage[idCounter];
    char decodedElem;
    while(1){
        bool checker = false;
        for(int i = 0; i < symbols.length(); i++){
            if(QString::fromStdString(cipher[i]) == code){
                checker = true;
                decodedElem = symbols[i];
                break;
            }
        }
        if(checker){
            break;
        }
        else{
            idCounter++;
            code+=decodedMessage[idCounter];
        }
```

```
    }
    ui->explanation->setText("Декодируем, заменяя:\n" +
decodedMessage.mid(symbCounter, idCounter+1-symbCounter) + " >>> "
+ decodedElem);
    decodedMessage = decodedMessage.mid(0, symbCounter) +
decodedElem + decodedMessage.mid(idCounter+1,
decodedMessage.length() - idCounter - 1);
    return false;
}

void MainWindow::loadFile()
{
    QString path = QFileDialog::getOpenFileName(this, "Открыть
файл", "", "*.txt"); //Выбор файла для прочтения
    if(path.isEmpty())
        return;
    const char* char_path = path.toUtf8().constData();
    isCloseCheck();
    readFile(char_path);
}

void MainWindow::loadImage()
{
    QString path = "./result.png";   //Метод для вывода
изображения, созданного утилитой graphViz
    QPixmap pixmap(path);
    image_width = pixmap.width();
    image_height = pixmap.height();
    Image_widget->setScaledPixmap(pixmap);
    setMaximumHeight(image_height + 96);
    setMaximumWidth(image_width + 258);
    fullsize_decorator();
}

void MainWindow::isCloseCheck()
{
    delete ui->scrollArea->takeWidget();   //Возвращение полей в
изначальное состояние
    Image_widget = new ScaledPixmap;
    ui->scrollArea->setWidget(Image_widget);
    ui->Beauty_button->setEnabled(true);
    ui->Fullsize_button->setEnabled(true);
    algorithm.clear();
    line.clear();
    codedMessage.clear();
    decodedMessage.clear();
    if(cipher!=nullptr)
        delete[] cipher;
    cipher = nullptr;
    isDecoding = false;
    isCoding = false;
    isCodeMaking = true;
    symbols.clear();
```

```
    if(freq!=nullptr)
        delete[] freq;
    freq = nullptr;
    symbols_count = 0;
}

void MainWindow::closeProgramm(){
    QApplication::quit();
}

void MainWindow::fullsize_decorator(){
    if(ui->Fullsize_button->isChecked()){
        Image_widget->setMinimumHeight(image_height);
        Image_widget->setMinimumWidth(image_width);
    }
    else{
        Image_widget->setMinimumHeight(476);
        Image_widget->setMinimumWidth(476);
    }
}

void MainWindow::decorator(){
    QPoint start_pixmap = Image_widget->resizeWidget();
    if(start_pixmap.x() == 0 && start_pixmap.y() == 0){
        return;
    }
    int x_deviation = 2 * (start_pixmap.x() - ((width() - 734) /
2)) * image_height / image_width;
    int y_deviation = 2 * (start_pixmap.y() - ((height() - 572) /
2)) * image_width / image_height;
    resize(width() - (start_pixmap.x()*2) , height() -
(start_pixmap.y()*2));

    if(width() == 734 && x_deviation > 0 && image_height >
image_width){
        resize(width(), height() + x_deviation);
    }
    if(height() == 572 && y_deviation > 0 && image_width >
image_height){
        resize(width() + y_deviation, height());
    }
}

void MainWindow::checkClick(){
    if(isDecoding){  //Случай, когда сообщение декодируется
        if(isCoding){   //Пограничный случай - переход между
кодировкой и декодировкой
            isCoding = false;
            decodedMessage = codedMessage;
            text->setText(decodedMessage);
            ui->explanation->setText("Декодируем созданное\
nсообщение шифром,\nкоторый мы получили");
            return;
```

```
        }
        if(stringDecoder()){
            ui->Next_turn->setDisabled(true);
            ui->Turn_over->setDisabled(true);
            text->setText(codedMessage + "\n\n" + decodedMessage);
            ui->explanation->setText("Была\nпродемонстрирована\
nработа выбранного\nалгоритма\n\nПредставлены\nзакодированное\nи
декодированное\nсообщения");
            return;
        }
        text->setText(decodedMessage);
    }
    if(isCoding){  //Случай, когда сообщение кодируется
        if(isCodeMaking){  //Пограничный случай - переход между
созданием шифра и кодировкой
            isCodeMaking = false;
            codedMessage = QString::fromStdString(line);
            ui->Beauty_button->setDisabled(true);
            ui->Fullsize_button->setDisabled(true);
            text = new QLabel;
            text->setAlignment(Qt::AlignCenter);
            text->setStyleSheet("font: 25px Times New Roman;");
            ui->scrollArea->setWidget(text);
            text->setText(codedMessage);
            ui->explanation->setText("Закодируем исходное\
ncообщение шифром,\nкоторый мы получили");
            return;
        }
        if(stringCoder()){
            isDecoding = true;
            checkClick();
        }
        text->setText(codedMessage);
    }
    if(isCodeMaking){  //Случай, когда создается шифр
        if(algorithm.isEmpty()){
            this->setDisabled(true);
            QStringList items;
            items << tr("Фано-Шеннона") << tr("Хаффмана");
            bool ok = false;
            algorithm = QInputDialog::getItem(this,
tr("NotMainWindow"), tr("Выбор алгоритма:"), items, 0, false,
&ok);
            this->setEnabled(true);
            if(!ok){
                algorithm.clear();
                return;
            }
        }

        if(algorithm == "Фано-Шеннона"){
            string code;
            int temp = 0;
```

```
            int count = symbols_count;
            QString comment; comment+="Закодированные буквы\nна
данный момент:";
            QString outInfo; outInfo+="graph graphname{\n";
            outInfo+= '\t' +
QString::fromStdString(symbols.substr(0, symbols.length())) + "
[label=\"" + QString::fromStdString(symbols.substr(0,
symbols.length())) + '(' + QString::number(line.length()) + ")\"]\
n}";
            searchTreeFanoShenon('\0', code, 0, symbols.length() -
1, outInfo, temp, symbols_count, comment);
            ofstream outFile("./result.txt");
            outFile << outInfo.toStdString();
            outFile.close();
            system("dot -Tpng ./result.txt -o result.png");
            loadImage();
            ui->explanation->setText(comment);
            if(symbols_count == count){
                isCoding = true;
            }
        }

        if(algorithm == "Хаффмана"){
            QString comment; comment+="Закодированная часть\nна
данном этапе:\n";
            QString outInfo;
            symbols_count++;
            if(searchTreeHaffman(symbols_count, outInfo, comment))
{
                isCoding = true;
            }
            ofstream outFile("./result.txt");
            outFile << outInfo.toStdString();
            outFile.close();
            system("dot -Tpng ./result.txt -o result.png");
            loadImage();
            ui->explanation->setText(comment);
        }
    }
}

void MainWindow::checkFastClick(){  //Идентично checkClick(),
только мгновенно
 переходит на следующий этап алгоритма
    if(isDecoding){
        if(isCoding){
            isCoding = false;
            decodedMessage = codedMessage;
            text->setText(decodedMessage);
            ui->explanation->setText("Декодируем созданное\
ncообщение шифром,\nкоторый мы получили");
            return;
        }
```

```
        while(!stringDecoder());
        ui->Next_turn->setDisabled(true);
        ui->Turn_over->setDisabled(true);
        text->setText(codedMessage + "\n\n" + decodedMessage);
        ui->explanation->setText("Была\nпродемонстрирована\nработа
выбранного\nалгоритма\n\nПредставлены\nзакодированное\nи
декодированное\nсообщения");
        return;
    }
    if(isCoding){
        if(isCodeMaking){
            isCodeMaking = false;
            codedMessage = QString::fromStdString(line);
            ui->Beauty_button->setDisabled(true);
            ui->Fullsize_button->setDisabled(true);
            text = new QLabel;
            text->setAlignment(Qt::AlignCenter);
            text->setStyleSheet("font: 25px Times New Roman;");
            ui->scrollArea->setWidget(text);
            text->setText(codedMessage);
            ui->explanation->setText("Закодируем исходное\
nсообщение шифром,\nкоторый мы получили");
            return;
        }
        while(!stringCoder());
        isDecoding = true;
        text->setText(codedMessage);
    }
    if(isCodeMaking){
        if(algorithm.isEmpty()){
            this->setDisabled(true);
            QStringList items;
            items << tr("Фано-Шеннона") << tr("Хаффмана");
            bool ok = false;
            algorithm = QInputDialog::getItem(this,
tr("NotMainWindow"), tr("Выбор алгоритма:"), items, 0, false,
&ok);
            this->setEnabled(true);
            if(!ok){
                algorithm.clear();
                return;
            }
        }
        if(algorithm == "Фано-Шеннона"){
            string code;
            int temp = 0;
            QString comment; comment+="Закодированные буквы\nна
данный момент:";
            QString outInfo; outInfo+="graph graphname{\n";
            outInfo+= '\t' +
QString::fromStdString(symbols.substr(0, symbols.length())) + "
[label=\"" + QString::fromStdString(symbols.substr(0,
symbols.length())) + '(' + QString::number(line.length()) + ")\"]\
```

```
n}";
            searchTreeFanoShenon('\0', code, 0, symbols.length() -
1, outInfo, temp, 1000, comment);
            ofstream outFile("./result.txt");
            outFile << outInfo.toStdString();
            outFile.close();
            system("dot -Tpng ./result.txt -o result.png");
            loadImage();
            ui->explanation->setText(comment);
            isCoding = true;
        }
        if(algorithm == "Хаффмана"){
            QString comment; comment+="Закодированная часть\nна
данном этапе:\n";
            QString outInfo;
            symbols_count++;
            searchTreeHaffman(symbols.length() - 1, outInfo,
comment);
            isCoding = true;
            ofstream outFile("./result.txt");
            outFile << outInfo.toStdString();
            outFile.close();
            system("dot -Tpng ./result.txt -o result.png");
            loadImage();
            ui->explanation->setText(comment);
        }
    }
}

void MainWindow::inputText(){  //Метод для считывания строки с
клавиатуры
    bool bOk;
    QString str = QInputDialog::getText( 0, "NotMainWindow",
"Введенный текст:", QLineEdit::Normal, "Best course work", &bOk );
    if (bOk && !str.isNull()) {
        isCloseCheck();
        line = str.toStdString();
        transform(line.begin(), line.end(),
line.begin(), ::tolower);
        stringCheck();
        ui->Next_turn->setEnabled(true);
        ui->Turn_over->setEnabled(true);
        startCoding();
    }
}

void MainWindow::stringCheck(){  //Проверка на наличие в строке
спец символов DOT, их замена
    string check = ";:&|->{}()[],\\/";
    string answer;
    for(int i = 0; i < line.length(); i++)
        for(int j = 0; j < check.length(); j++)
            if(line[i] == check[j]){
```

```
                answer.push_back(line[i]);
                line[i] = ' ';
            }
    if(!answer.empty())
        QMessageBox::warning(this, "Внимание", "Эти символы:\n" +
QString::fromStdString(answer) + "\nявляются частью синтаксиса
DOT.\nОни заменены на пробел(_)");
}
```

SCALEDPIXMAP.CPP:

```
#include "scaledpixmap.h"
#include "mainwindow.h"


ScaledPixmap::ScaledPixmap(QWidget *parent): QWidget(parent) {
}

void ScaledPixmap::setScaledPixmap(const QPixmap &pixmap) {
  m_pixmap = pixmap;
  update();
}

void ScaledPixmap::removeScaledPixmap(){
    pixmap_width = 0;
    pixmap_height = 0;
    pixmap_x_start = 0;
    pixmap_y_start = 0;
}

void ScaledPixmap::paintEvent(QPaintEvent *event) {
  QPainter painter(this);

  if (false == m_pixmap.isNull()) {
    QSize widgetSize = size();
    QPixmap scaledPixmap = m_pixmap.scaled(widgetSize,
Qt::KeepAspectRatio);
    pixmap_width = scaledPixmap.width();
    pixmap_height = scaledPixmap.height();
    QPoint center((widgetSize.width() - scaledPixmap.width())/2,
                  (widgetSize.height() -
scaledPixmap.height())/2);
    pixmap_x_start = center.x();
    pixmap_y_start = center.y();
    painter.drawPixmap(center, scaledPixmap);
  }

  QWidget::paintEvent(event);
}

void ScaledPixmap::mousePressEvent(QMouseEvent *event) {
    press_point.setX(event->x());
```

```
    press_point.setY(event->y());
}

void ScaledPixmap::mouseReleaseEvent(QMouseEvent *event) {
    release_point.setX(event->x());
    release_point.setY(event->y());
}

QPoint ScaledPixmap::resizeWidget(){
    QPoint start_pixmap;
    start_pixmap.setX(pixmap_x_start);
    start_pixmap.setY(pixmap_y_start);
    return start_pixmap;
}

void ScaledPixmap::resetMouse(){
    press_point.setX(0);
    press_point.setY(0);
    release_point.setX(0);
    release_point.setY(0);
}

QPoint ScaledPixmap::backMouse(int count){
    if(count == 1)
        return press_point;
    else
        return release_point;
}
```

ПРИЛОЖЕНИЕ Б

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ ПРОГРАММЫ

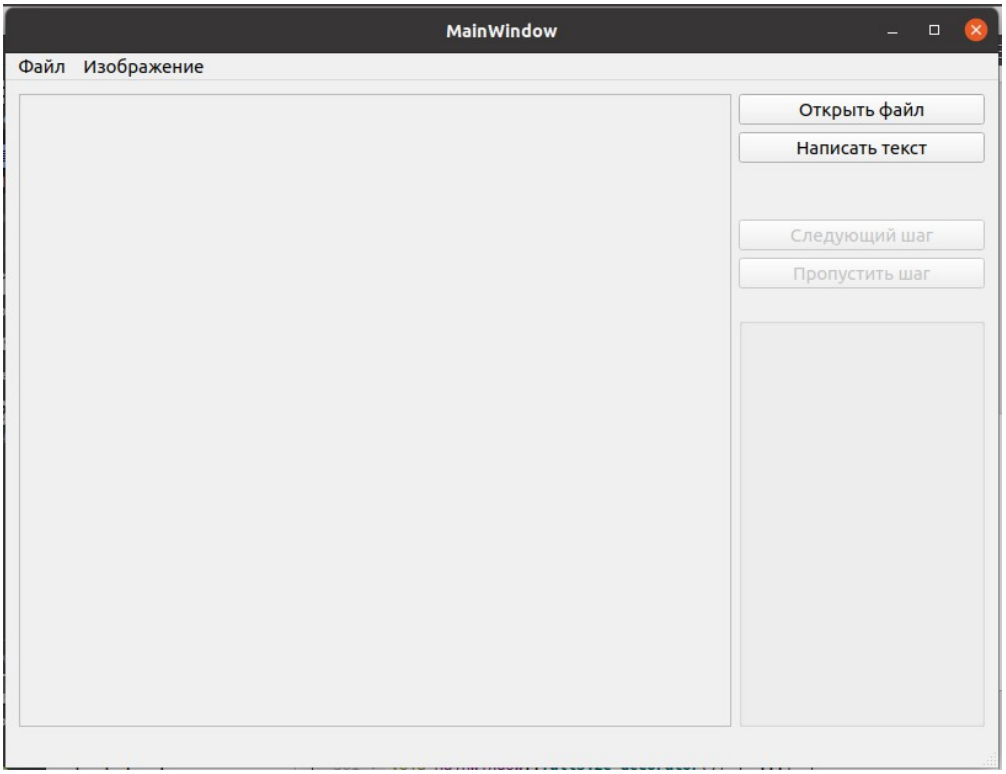

Рисунок 1 — Начало работы программы

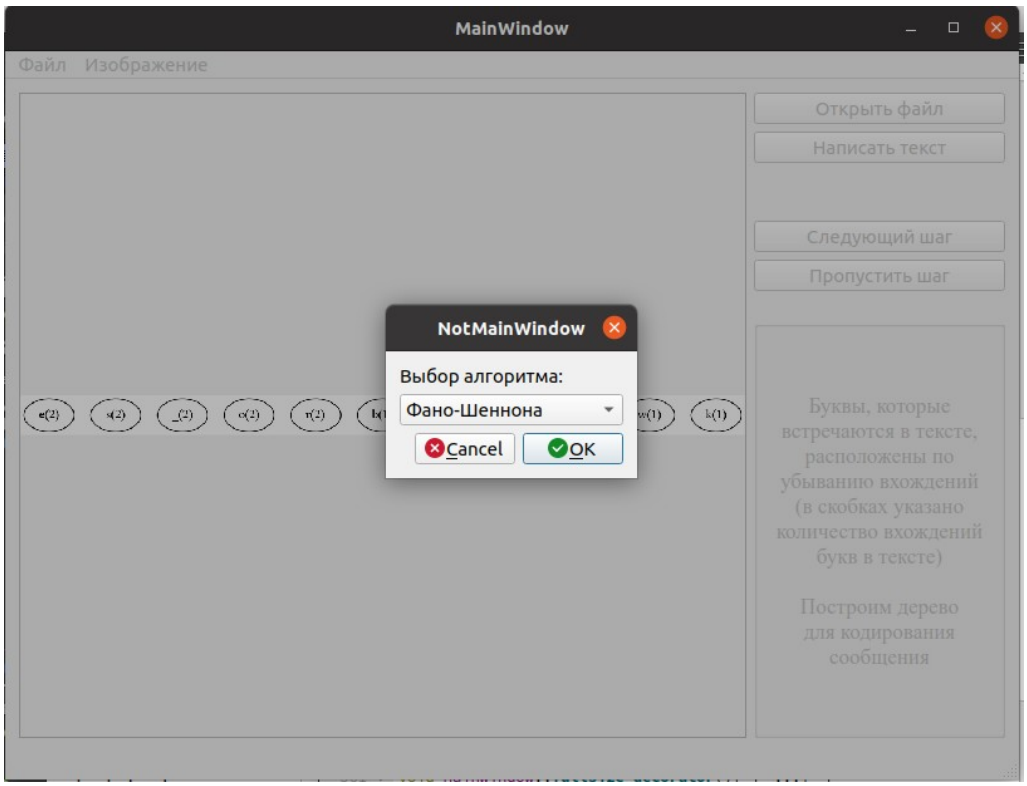

Рисунок 2 — Выбор используемого алгоритмам

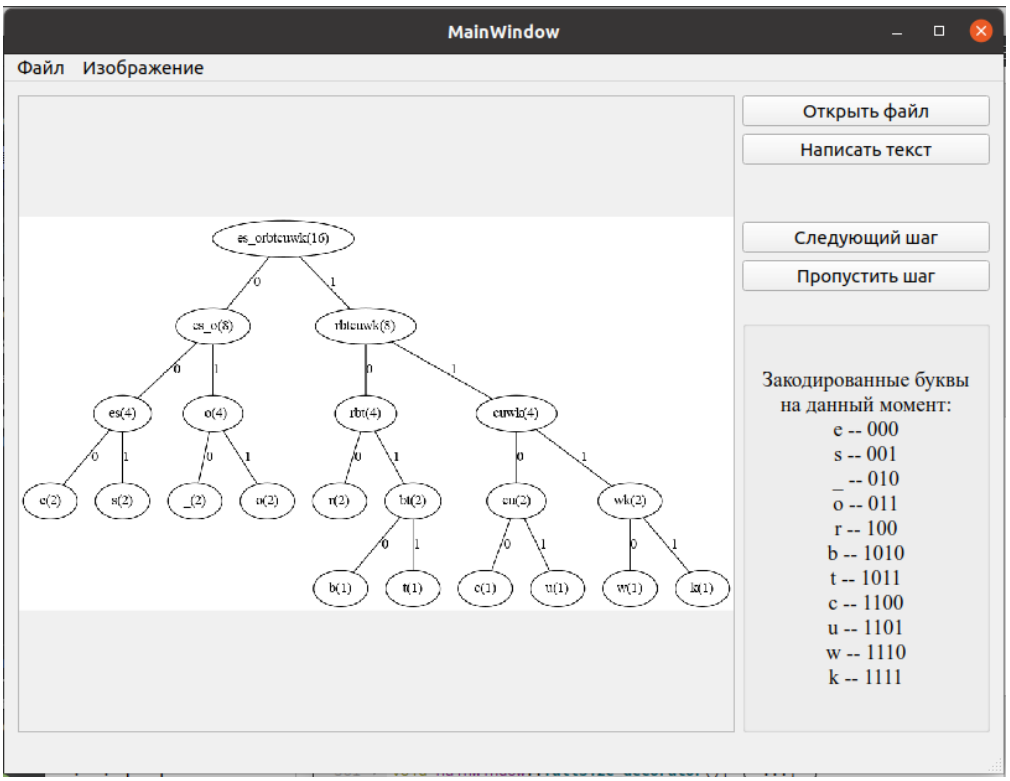

Рисунок 3 — Выполнение создание цифра, дерева

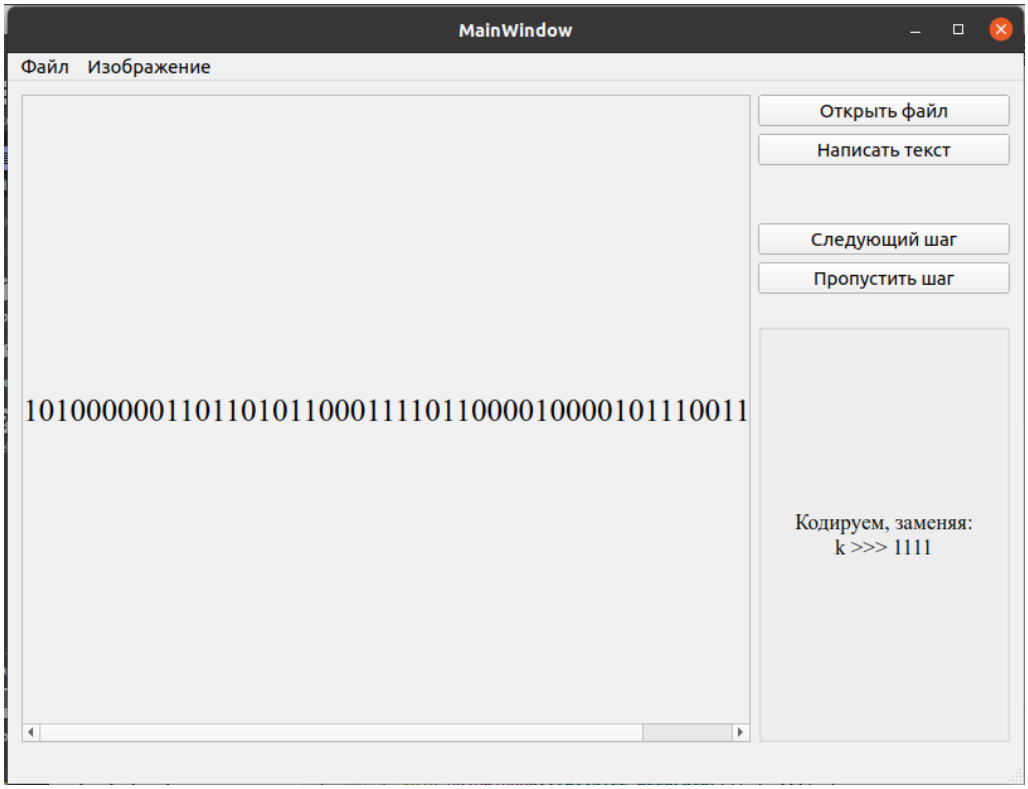

Рисунок 4 — Кодировка сообщения

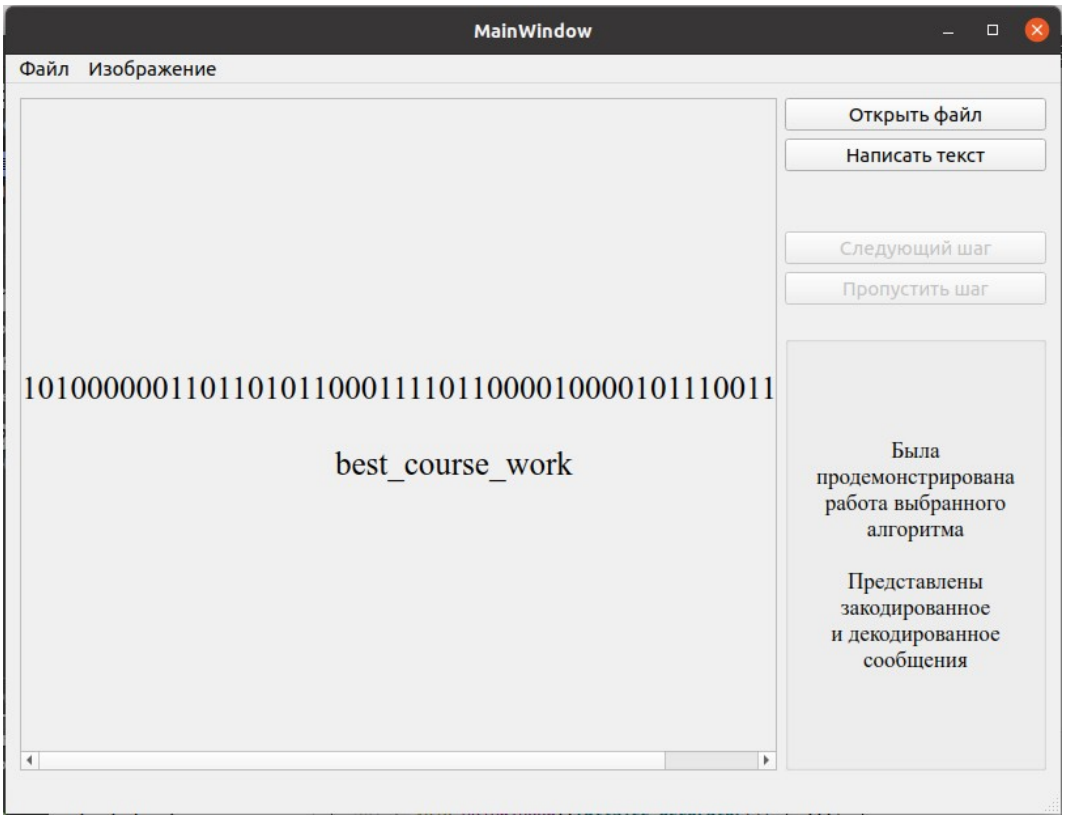

Рисунок 5 — Декодировка сообщения