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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: Динамическое кодирование
Хаффмана

Студент гр. 8381 _____ Муковский Д.В.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Ознакомиться с динамическим алгоритмом кодирования Хаффмана. Изучить особенности его реализации на языке программирования *C++*. Разработать программу визуализирующую данный алгоритм.

Задание.

На вход подается строка, которую нужно закодировать адаптивным алгоритмом Хаффмана.

Основные теоретические положения.

Адаптивный алгоритм Хаффмана является модификацией обычного алгоритма Хаффмана сжатия сообщений. Он позволяет не передавать таблицу кодов и ограничиться одним проходом по сообщению, как при кодировании, так и при декодировании. Суть адаптивного алгоритма состоит в том, что при каждом сопоставлении символу кода изменяется внутренний ход вычислений так, что в следующий раз этому же символу может быть сопоставлен другой код, т.е. происходит адаптация алгоритма к поступающим для кодирования символам.

В адаптивном алгоритме сжатия Хаффмана используется упорядоченное бинарное дерево. Бинарное дерево называется упорядоченным, если его узлы могут быть перечислены в порядке не убывания веса. Перечисление узлов происходит по ярусам снизу-вверх и слева-направо в каждом ярусе. Узлы, имеющие общего родителя, находятся рядом на одном ярусе.

Алгоритм динамического кодирования Хаффмана:

1.Элементы входного сообщения считываются побайтно.

2. Если входной символ присутствует в дереве, в выходной поток записывается код, соответствующий последовательности нулей и единиц, которыми помечены ветки дерева, при проходе от корня дерева к данному листу. Вес данного листа увеличивается на 1. Веса узлов-предков корректируются. Если дерево становится неупорядоченным - упорядочивается.

3. Если очередной символ, считанный из входного сообщения при сжатии, отсутствует в дереве, в выходной поток записывается набор нулей и единиц, которыми помечены ветки бинарного дерева при движении от корня к escape-символу, а затем 8 бит *ASCII*-кода нового символа. В дерево вместо escape-символа добавляется ветка: родитель, два потомка. Левый потомок становится escape-символом, правый - новым добавленным в дерево символом. Веса узлов-предков корректируются, а дерево при необходимости упорядочивается.

Выполнение работы.

Написание работы производилось на базе операционной системы Windows в среде разработки *QtCreator* с использованием фреймворка *Qt*. Сборка, отладка производились в *QtCreator*. Исходные коды файлов программы представлены в приложениях А-З.

Для реализации программы был разработан графический интерфейс с помощью встроенного в *QtCreator UI*-редактора. Он представляет из себя поле считывания, кнопки запуска программы в разных режимах, сцена, на которой изображается дерево и поле вывода результата. Основные слоты и методы для работы графического интерфейса приведены в табл. 1.

Таблица 1 –Методы и слоты класса *MainWindow* и их назначение

Метод и слоты	Назначение
`updateScene()`	Метод, который обновляет сцену с новым деревом
`DrawNode()`	Выводит дерево на сцену
`readInput()`	Считывает строку и проверяет ее
`on_startCodingButton_clicked()`	Запускает алгоритм единожды
`setMode()`	Переключает режим работы программы

Продолжение табл.2

on_stepByStepStart_clicked()	Запускает пошаговый режим алгоритма
on_nextStep_clicked()	Переходит к следующему шагу алгоритма
on_previousStep_clicked()	Переходит к предыдущему шагу алгоритма
on_stopButton_clicked()	Останавливает пошаговый режим
on_readFileButton_clicked()	Считывает строку из текстового файла
on_saveButton_clicked()	Сохраняет результат в файл

Бинарное дерево было реализовано через структуру *Node*, описывающую узел дерева, и класс бинарного дерева, который хранит корень дерева. Все методы данного класса приведены в табл. 2.

Таблица 2 – Основные методы работы с бинарным деревом

Метод	Назначение
QVector<Node> makeArray*()	Создает массив из дерева в порядке от нижнего яруса к верхнему, слева направо
void updateTree()	Преобразует дерево в упорядоченное дерево
int getMaxTreeDepth()	Возвращает максимальную глубину дерева
Node getRoot*()	Возвращает корень дерева
void freeMem()	Высвобождает память дерева

Реализация алгоритма.

Алгоритм был реализирован несколькими функциями. Они представлены в табл.3 с пояснениями.

Таблица 3 – описание функций, которые реализируют алгоритм

Функция	Назначение
void encode()	Функция, в которой описан алгоритм
*int*codeOfNode*()	Функция, которая записывает последовательность нулей и единиц, которыми помечены ветки дерева, при проходе от корня дерева к заданному листу(записывает в обратном порядке)
void reverseCode()	Функция, инвертирующая строку
Node getTreeFromSymbol*()	Функция, проверяющая был ли текущий символ уже закодирован
Node addSymbol*()	Функция, добавляющая новый символ в дерево
void addCodeToOut()	Функция, записывающая результат

Оценка сложности алгоритма.

Сложность данного алгоритма зависит от реализации функции перестройки дерева. В ходе тестирования было выяснено, что данный алгоритм имеет сложность $O(n*logn)$, данные тестирования в виде графика изображены на рис.1.

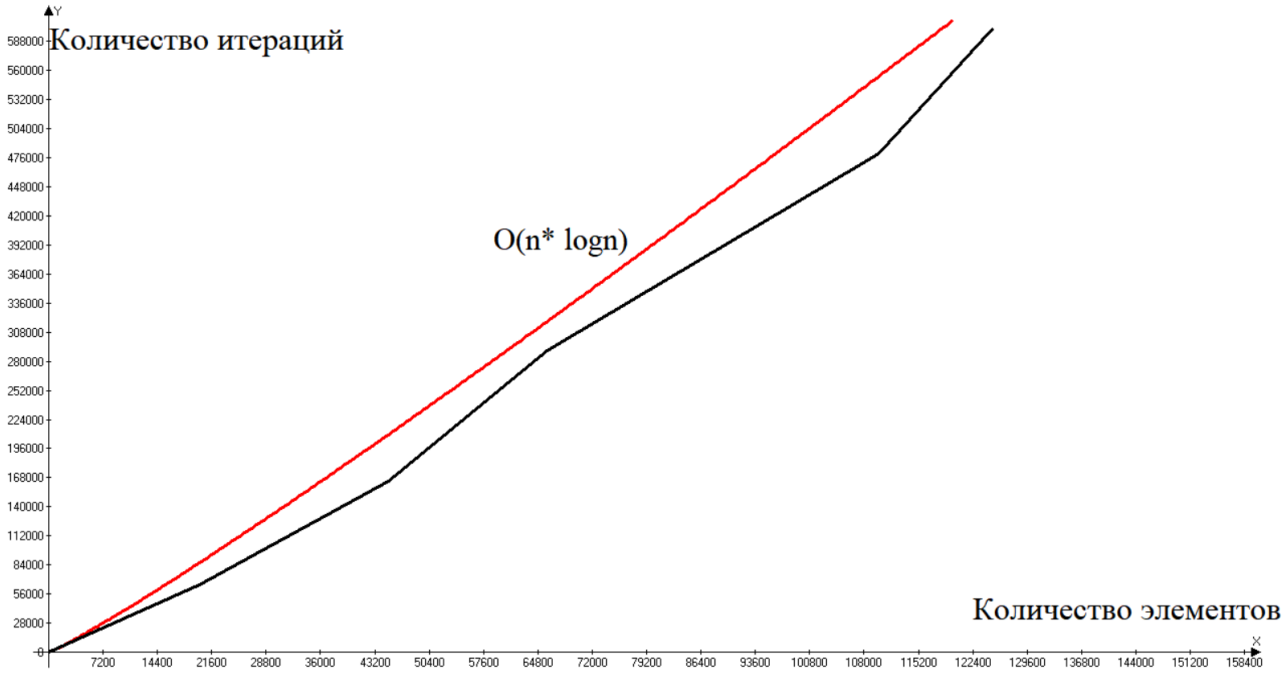

Рисунок 1 – сложность алгоритма

Тестирование программы.

Тесты программы представлены в табл. 4.

Входная строка	Закодированный результат
That was an error!	0101010000110100000011000011000111010000000100000110001110111011000011100111010100000110111011111000011001010010001110010100011000001101111011010000010000001
aaaaaaaaaaaaaaaaaaaaab	011000011111111111111111111111001100010
hihihihihiolololo	0110100000110100110110110110100011011111000110110010110011011001010
Je ma pel Daniel	0100101000110010100001000001000110110100000110000101110001110000000100011011001010111000100010001111000011011101000001101001111110 1

Вид программы после выполнения представлен на рис. 2.1, 2.2 и 2.3

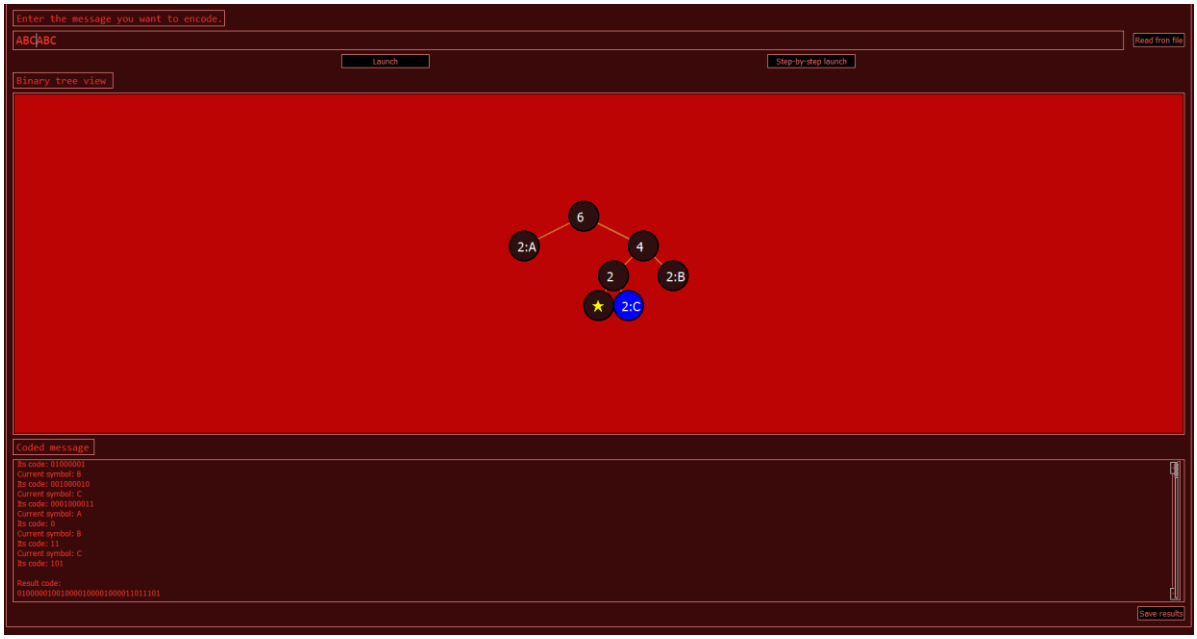

Рисунок 2.1 – выполнение программы

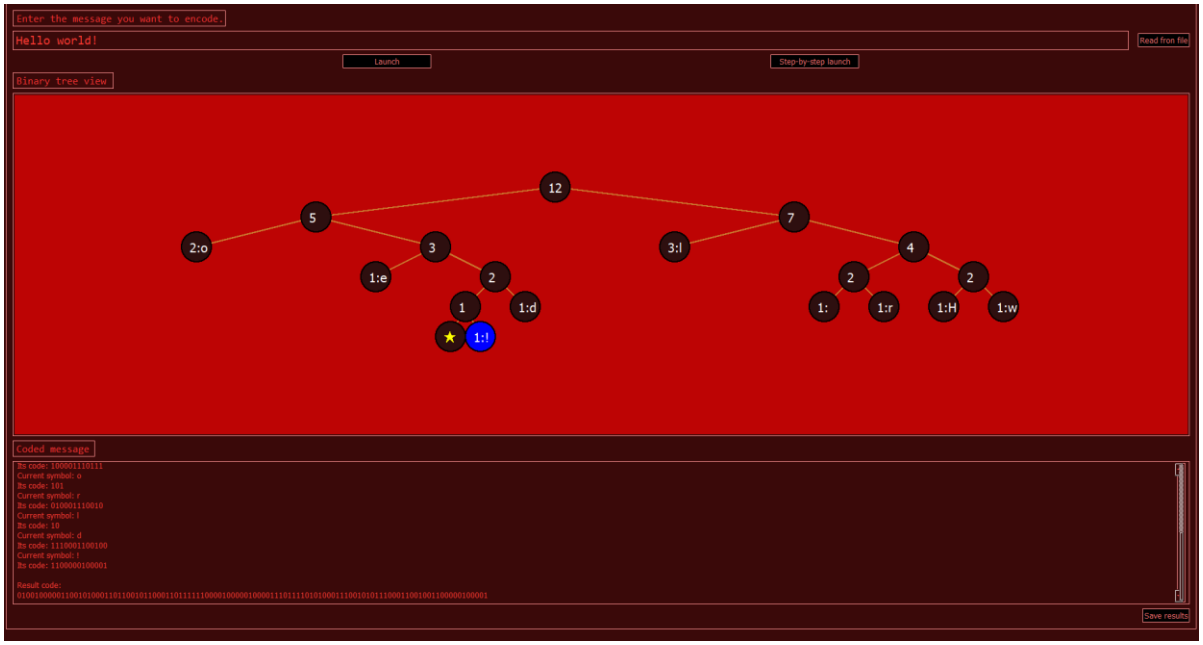

Рисунок 2.2 – выполнение программы

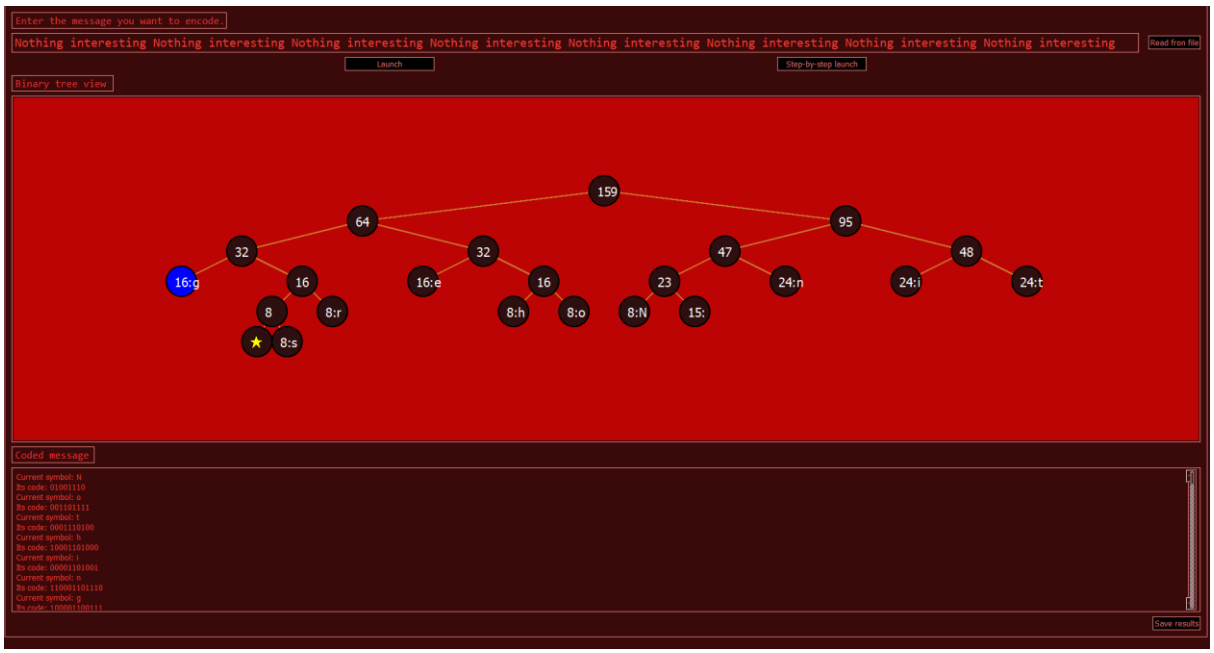

Рисунок 2.3 – выполнение программы

Выводы.

В ходе выполнения лабораторной работы была написана программа, кодирующая заданное сообщение с помощью алгоритма динамического кодирования Хаффмана. Также был реализован интерфейс, изображающий упорядоченное бинарное дерево, который позволяет увидеть работу алгоритма на каждом шаге.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    bool flag = false;
    for(int i = 1; i < argc; i++)
    {
        if(!strcmp("-console", argv[i]))
            flag = true;
    }
    if(flag){
        consoleMain();
        return 0;
    }
    else
    {
        QApplication a(argc, argv);
        MainWindow w;
        w.show();
        return a.exec();
    }
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QObject>
#include <QMessageBox>
#include <QDebug>
#include <QString>
#include <QFileDialog>
#include <QGraphicsItem>
#include <QtGui>
#include <QDialog>
#include <QColorDialog>
#include <QString>
#include <QDebug>
#include <QPainter>
#include <QComboBox>
#include <QLabel>
#include <QPushButton>
#include <QFile>
#include <QWidget>
#include <QVBoxLayout>
#include <QPushButton>
#include <QLabel>
#include <QLineEdit>
#include <QGroupBox>
#include <QRadioButton>
#include <QTextEdit>
#include <QEventLoop>
#include <QTimer>
#include <QColor>
#include <QDebug>
#include <QGraphicsView>
#include <QFormLayout>
#include "decodingandcodinghuufmanalgorithm.h"
#include <fstream>
#include "console.h"

#define RUSSIAN
"АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя"



QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

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

private slots:
    void on_startCodingButton_clicked();
```

```
    void on_stepByStepStart_clicked();

    void on_nextStep_clicked();

    void on_previousStep_clicked();

    void on_stopButton_clicked();

    void on_readFileButton_clicked();

    void on_saveButton_clicked();

private:
    Ui::MainWindow *ui;
    QGraphicsScene *mainGraphicsScene;

    std::string output = "";
    std::string resultCode = "";
    QPen pen;
    QColor color;
    QBrush brush;
    QFont font;
    BinTree* tree;

    int currentInputLen = 0;
    int inputLen = 0;
    char* input ;
    void setMode(bool isMode);
    void DrawNode(Node* n, int maxdepth, int depth = 0, int x = 0, int y = 0);
    void updateScene();
    bool readInput();

};
bool checkRussian(QString str);

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <string>
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent),
    ui(new Ui::MainWindow),
    mainGraphicsScene(new QGraphicsScene())
{
    ui->setupUi(this);
    ui->graphicsView->setScene(mainGraphicsScene);
    this->setWindowTitle("HUFFMAN CODING");
    //this->setWindowFlags(Qt::CustomizeWindowHint);
    QMainWindow::showMaximized();
    QColor color(203,119,47);
    pen.setColor (color);
    brush.setColor(color);
    font.setFamily("Roboto");
    pen.setWidth(3);
    //ui->horizontalSlider->setValue(2);
    ui->graphicsView->resetTransform();
    ui->graphicsView->scale(1.5/2,1.5/2);
    ui->nextStep->hide();
    ui->previousStep->hide();
    ui->stopButton->hide();
    ui->saveButton->hide();
}

MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::updateScene(){
    ui->graphicsView->resetTransform();
     ui->graphicsView->scale(1.5/2,1.5/2);
    Node* root = tree->getRoot();
    mainGraphicsScene->clear();
    if (!root) return;
    DrawNode(root,tree->getMaxTreeDepth(root));
}



void MainWindow::DrawNode(Node * n,int maxdepth,int depth,int x,int y){
    if (n==nullptr) return;
    int offset = pow(2,maxdepth+3)/pow(2,depth);
    if (n->left)
        mainGraphicsScene->addLine(x+32,y+32,x-offset+32,y+64+32,pen);
    if (n->right)
        mainGraphicsScene->addLine(x+32,y+32,x+offset+32,y+64+32,pen);
    QColor color_c(46,15,15);
    QBrush brush(color_c);
    QPen pen(color,3);
    if (n->isLast){
        brush.setColor(Qt::blue);
        mainGraphicsScene->addEllipse(x,y,64,64,pen,brush);
    }
```

```
    else{
        brush.setColor(color_c);
        mainGraphicsScene->addEllipse(x,y,64,64,pen,brush);
    }
    QGraphicsTextItem *numb = new QGraphicsTextItem();
    numb->setDefaultTextColor(Qt::white);
    QString textRes;
    if (QChar(n->symbol)!='\n'){
        textRes = QString::number(n->value)+':'+QChar(n->symbol);
    }
    else if (n->value==0){
        numb->setDefaultTextColor(Qt::yellow);
        textRes = "★";
    }
    else
        textRes = QString::number(n->value);
    numb->setPlainText(textRes);
    numb->setScale(2);
    numb->setPos(x+10,y+10);
    mainGraphicsScene->addItem(numb);
    DrawNode(n->left,maxdepth,depth+1,x-offset,y+64);
    DrawNode(n->right,maxdepth,depth+1,x+offset,y+64);

}

bool MainWindow::readInput(){
    QString inputString = ui->inputStr->text();
    if (inputString.size()==0){
        QMessageBox::critical(this,"ERROR!","No massage");
        return false;
    }
    if (checkRussian(inputString)){
        QMessageBox::critical(this,"ERROR!","Text contains Russian");
        return false;
    }
    int i;
    inputLen = inputString.size();
    input = new char[inputString.length()+1];
    for (i =0;i<inputString.length();i++){
        input[i] = inputString[i].toLatin1();
    }
    input[i] = '\0';
    return true;
}




void MainWindow::on_startCodingButton_clicked()
{
    ui->answer->clear();
    mainGraphicsScene->clear();
    output.clear();
    resultCode.clear();
    if (!readInput()) return;

    tree = new(BinTree);

    encode(input, output, inputLen, tree,resultCode);
    delete[] input;
```

```
    output = output +"\n"+"Result code:\n"+resultCode;
    QString answ = QString::fromStdString(output);
    ui->answer->setText(answ);
    ui->saveButton->show();
    output.clear();
    updateScene();
    Node* root = tree->getRoot();
    tree->freeMem(root);
}

void MainWindow::setMode(bool isMode){
    ui->readFileButton->setEnabled(isMode);
    ui->inputStr->setEnabled(isMode);
    currentInputLen = 0;
    output.clear();
    if (!isMode){
        ui->answer->clear();
        mainGraphicsScene->clear();
        ui->nextStep->show();
        ui->previousStep->show();
        ui->stopButton->show();
        ui->startCodingButton->hide();
        ui->stepByStepStart->hide();
    }
    else{
        ui->nextStep->hide();
        ui->previousStep->hide();
        ui->stopButton->hide();
        ui->startCodingButton->show();
        ui->stepByStepStart->show();
        delete[] input;
    }
}


void MainWindow::on_stepByStepStart_clicked()
{
    ui->saveButton->hide();
    if (!readInput()) return;
    setMode(false);
}



void MainWindow::on_nextStep_clicked()
{

    tree = new(BinTree);
    currentInputLen++;
    if (currentInputLen>inputLen){
        currentInputLen--;
        QMessageBox::about(this,"","Algorithm has reached the end");
        output = output +"\n"+"Result code:\n"+resultCode;
        QString resultAnsw = QString::fromStdString(output);
        ui->saveButton->show();
        ui->answer->setText(resultAnsw);
        setMode(true);
        return;
    }
    output = "";
```

```
    resultCode = "";
    encode(input, output, currentInputLen, tree,resultCode);

    QString answ = QString::fromStdString(output);
    ui->answer->setText(answ);

    updateScene();
    Node* root = tree->getRoot();
    tree->freeMem(root);
}




void MainWindow::on_previousStep_clicked()
{
    tree = new(BinTree);
    currentInputLen--;
    if (currentInputLen<=0){
        currentInputLen++;
        QMessageBox::about(this,"","Algorithm has reached start");
        return;
    }
    output = "";
    resultCode = "";
    encode(input, output, currentInputLen, tree,resultCode);
    QString answ = QString::fromStdString(output);
    ui->answer->setText(answ);
    updateScene();
    Node* root = tree->getRoot();
    tree->freeMem(root);
}

void MainWindow::on_stopButton_clicked()
{
    setMode(true);
}




void MainWindow::on_readFileButton_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("load"),
QDir::homePath(), tr("*.txt"));

            if (QString::compare(fileName, QString()) != 0)
            {
                std::ifstream f(qPrintable(fileName), std::ios::in);
                std::string out;
                getline(f, out);
                f.close();
                ui->inputStr->setText(QString::fromStdString(out));
            }
}

bool checkRussian(QString string){
```

```
    static QString russian =RUSSIAN;
    foreach(const QChar & ch, russian) {
            if(string.contains(ch)) {
                return true;
            }
        }
        return false;

}



void MainWindow::on_saveButton_clicked()
{
    QString filePath = QFileDialog::getSaveFileName(this, tr("save"),
QDir::homePath(), tr("*.txt"));
            if (QString::compare(filePath, QString()) != 0)
            {
                std::ofstream ff(qPrintable(filePath));
                ff << qPrintable(ui->answer->toPlainText());
                ff.close();
            }
            ui->saveButton->hide();

}
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.H

```
#ifndef BINTREE_H
#define BINTREE_H
#include <memory>
#include <iostream>
#include <string>
#include <QString>
#include <sstream>
#include <QVector>
#include <QQueue>
#include <algorithm>

#define INVALID -1 /* setted as the symbol of the non-leaf nodes */
#define HOW_MANY_POSSIBLE_SYMBOLS 256 /* how many possible symbols */

struct Node{
    Node *left = nullptr;
    Node *right =nullptr;
    Node *parent ;
    int value;
    int isZero;
    char symbol = ' ';
    int isRoot;
    int isLeaf;
    int level = 1;
    bool isLast;
};

struct Symbol{
    char symbol;
    Node *tree;
};

class BinTree{
private:
    Node* root = nullptr;
public:
    BinTree(){
        root = new Node;
        root->isZero = 1;
        root->isRoot = 1;
        root->isLeaf = 1;

        root->parent = nullptr;
        root->left = nullptr;
        root->right = nullptr;
        root->isLast = false;

        root->symbol = '\n';
        root->value = 0;
    }
    ~BinTree(){
        this->freeMem(root);
    }
    QVector<Node*> makeArray();
    void updateTree(Node* currNode);
    int getMaxTreeDepth(Node* node );
    Node* getRoot();
```

```
    void freeMem(Node* node = nullptr);

};
bool swapNodes(Node* n1, Node* n2);

}
```

ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.CPP

```
#include "bintree.h"


int BinTree::getMaxTreeDepth(Node* tree){
    if(tree == nullptr) return 0;
        else{
            int lDepth = this->getMaxTreeDepth(tree->left);
            int rDepth = this->getMaxTreeDepth(tree->right);
            if (lDepth > rDepth)  return(lDepth + 1);
            else return(rDepth + 1);
        }
}

QVector<Node*> BinTree::makeArray(){
    QQueue<Node*> queue;
    QVector<Node*> result ;
    queue.enqueue(root);
    for(;;){
        Node* current = queue.dequeue();
       (result).push_back(current);
        if (current->left==nullptr && current->right == nullptr &&
queue.size()==0){
            break;
        }
        if (current->right!=nullptr){
            queue.enqueue(current->right);
            current->right->level = current->level +1;
        }
        if (current->left!=nullptr){
            queue.enqueue(current->left);
            current->left->level = current->level +1;
        }
    }
    std::reverse( result.begin(), result.end());
    return result;
}

void BinTree::freeMem(Node* tree){
    if(!tree)
        return;
    if (tree->right)
        this->freeMem(tree->right);
    if (tree->left)
        this->freeMem(tree->left);
    delete tree;
    return;
}
Node* BinTree::getRoot(){
    return root;
}

void BinTree::updateTree(Node*currNode){
    while(!(currNode->isRoot)){
        (currNode->value)++;
         currNode = currNode->parent;
    }
    if (currNode->isRoot)
```

```
        (currNode->value)++;
    QVector<Node*> arr;
    Node* first;
    Node* second;
    for(;;){
        int i =0;
        first = nullptr;
        second = nullptr;

        arr= this->makeArray();

        int min=0;

        bool flag = false;

        for (i =0;i<arr.size();i++){
            min = arr[i]->value;
            for (int j = i+1;j<arr.size()-1;j++){
                if(arr[j]->value<min){
                    first = arr[i];
                    flag = true;
                    break;
                }
            }
            if (flag)break;
        }
        if(!first)return;
        int fir = i;
        int max = first->value;
        for(i = fir+1;i<arr.size();i++){
            if (arr[i]->value<max){
                min = arr[i]->value;
                second = arr[i];
                break;
            }

        }
        if(!second)return;

        for(i = fir+1;i<arr.size();i++){
            if (max>arr[i]->value && min>=arr[i]->value){
                min = arr[i]->value;
                second = arr[i];
            }
        }
        if(!second)return;

        if (!swapNodes(first,second)){
            std::cout<<"ERROR NODES";
            return;
        }
    }
}


bool swapNodes(Node* n1, Node* n2) {
    int diff = n1->value-n2->value;
    if (n1==n2) return false;
    if (n1->parent!=n2->parent){
        if (n1->parent->left == n1) {
            n1->parent->left = n2;
```

```
        }
        else {
            n1->parent->right = n2;
        }
        if (n2->parent->left == n2) {
            n2->parent->left = n1;
        }
        else {
            n2->parent->right = n1;
        }
        Node* temp = n1->parent;
        n1->parent = n2->parent;
        n2->parent = temp;
        Node* s2=n2->parent;
        Node *s1=n1->parent;

        while(!(s2->isRoot)){
            (s2->value)-=diff;
            s2 = s2->parent;
        }
        while(!(s1->isRoot)){
            (s1->value)+=diff;
            s1 = s1->parent;
        }
    }
    else {
        if (n1->parent->left ==n1){
            n1->parent->left = n2;
            n2->parent->right = n1;
        }
        else{
            n1->parent->right = n2;
            n2->parent->left = n1;
        }
    }
    return true;
}
```

ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ.

DECODINGANDCODINGHUUFMANALGORITHM.H

```
#ifndef DECODINGANDCODINGHUUFMANALGORITHM_H
#define DECODINGANDCODINGHUUFMANALGORITHM_H
#include "bintree.h"
Node* getTreeFromSymbol(unsigned char symbol, Symbol **symbols);
void reverseCode(int *code,int codeSize);
int* codeOfNode(Node *node, int *n);
Node* addChild(Node *parent, int isZero, int isRoot, unsigned char symbol, int
value, int order);
Node* addSymbol(unsigned char symbol, Node** zeroNode, Symbol **symbols);
void addCodeToOut(std::string&outp,std::string& resultCode,int
codeSize,int*symbCode, char byte,bool flag);
void addAsciiToOut(std::string&outp,std::string& resultCode,int byte);
void encode(char* input, std::string& output, int inputSzie,BinTree*
root,std::string& resultCode);



#endif // DECODINGANDCODINGHUUFMANALGORITHM_H
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ.

DECODINGANDCODINGHUUFMANALGORITHM.CPP

```
#include "decodingandcodinghuufmanalgorithm.h"
Node* getTreeFromSymbol(char symbol, Symbol** symbols) {
    int i = symbol;
    Symbol* symbolPtr = symbols[i];
    if (!symbolPtr)
        return nullptr;
    return symbolPtr->tree;
}


void reverseCode(int* code, int codeSize) {
    if (code == nullptr) {
        return;
    }
    int* start = code;
    int* end = code + (codeSize - 1);
    while (start < end) {
        int temp = *start;
        *start = *end;
        *end = temp;
        start++;
        end--;
    }
}

int* codeOfNode(Node* node, int* n) {
    Node* current = node;

    int* code = new int[HOW_MANY_POSSIBLE_SYMBOLS * 2];

    int i = 0;

    while (!current->isRoot) {
        Node* parent = current->parent;
        code[i] = (parent->left == current) ? 0 : 1;
        current = current->parent;
        i++;
    }
    reverseCode(code, i);

    *n = i;
    return code;
}

Node* addChild(Node* parent, int isZero, int isRoot, char symbol, int value,
bool isLast) {
    Node* node = new Node;
    node->isZero = isZero;
    node->isRoot = isRoot;
    node->isLast = isLast;
    node->isLeaf = 1;
    node->parent = parent;
    node->left = nullptr;
    node->right = nullptr;
    node->symbol = symbol;
```

```
    node->value = value;
    return node;
}

Node* addSymbol(char symbol, Node** zeroNode, Symbol** symbols) {
    Node* leftNode = addChild(*zeroNode, 1, 0, '\n', 0,false);
    Node* rightNode = addChild(*zeroNode, 0, 0, symbol, 0,false);

    (*zeroNode)->isZero = 0;
    (*zeroNode)->isLeaf = 0;
    (*zeroNode)->isLast = false;
    (*zeroNode)->left = leftNode;
    (*zeroNode)->right = rightNode;
    int i = symbol;
    symbols[i] = new Symbol;
    symbols[i]->symbol = symbol;
    symbols[i]->tree = rightNode;

    *zeroNode = leftNode;
    return rightNode;
}



template <typename T>
std::string NumberToString(T Number)
{
    std::ostringstream ss;
    ss << Number;
    return ss.str();
}


void addCodeToOut(std::string& outp,std::string& resultCode, int codeSize, int*
symbCode, char byte, bool flag) {
    outp=outp+"Current symbol: "+byte+"\n"+"Its code: ";
    for (int i = 0; i < codeSize; i++) {
        outp += NumberToString(symbCode[i]);
        resultCode += NumberToString(symbCode[i]);
    }
    if (flag){
        addAsciiToOut(outp,resultCode,byte);
    }
    outp+="\n";
}


void addAsciiToOut(std::string& outp,std::string& resultCode, int byte) {
    for (int i = 0; i < 8; i++) {
        int some = byte;
        some = some & 128;
        (some == 128) ? outp += "1" : outp += "0";
        (some == 128) ? resultCode += "1" : resultCode+= "0";
        byte <<= 1;
    }


}
```

```
void encode(char* input, std::string& output, int inputSize, BinTree*
BT,std::string& resultCode) {
    Node* zeroNode = BT->getRoot();
    Symbol** symbols = new Symbol * [HOW_MANY_POSSIBLE_SYMBOLS];

    for (int j = 0; j < HOW_MANY_POSSIBLE_SYMBOLS; j++) {
        symbols[j] = nullptr;
    }
    bool flag = false;


    char currByte;
    Node* newNode = nullptr;
    Node* symbolTree = nullptr;
    for (int i = 0; i < inputSize; i++) {
        currByte = input[i];
        symbolTree = getTreeFromSymbol(currByte, symbols);
        if (symbolTree) {
            int codeSize;
            int* symbolCode = codeOfNode(symbolTree, &codeSize);
            addCodeToOut(output,resultCode, codeSize, sym-
bolCode,currByte,false);
            BT->updateTree(symbolTree);
            flag = false;
            delete symbolCode;
        }
        else {
            int codeSize;
            int* zeroCode = codeOfNode(zeroNode, &codeSize);
            addCodeToOut(output,resultCode, codeSize, zeroCode,currByte,true);
           // addAsciiToOut(output, currByte);
            newNode = addSymbol(currByte, &zeroNode, symbols);
            BT->updateTree(newNode);
            flag = true;
            delete zeroCode;
        }
    }

    if (flag && newNode){
        newNode->isLast = true;
    }
    else if (!flag && symbolTree){
         symbolTree->isLast = true;
    }
    for (int j = 0; j < HOW_MANY_POSSIBLE_SYMBOLS; j++) {
        delete symbols[j];
    }
    delete[] symbols;
}
```

ПРИЛОЖЕНИЕ 3

ИСХОДНЫЙ КОД ПРОГРАММЫ. CONSOLE.CPP

```
#include "console.h"
#include <QFile>
#include "decodingandcodinghuufmanalgorithm.h"

int consoleMain(){
    QFile file("C:\\Users\\miair\\Desktop\\Results_lr5\\input.txt"); // создаем
объект класса QFile
    QString in;
    if (!file.open(QIODevice::ReadOnly))
        return 1 ;
    int i =0;
    in = file.readAll();
    int inputLen = in.size();
    char* input = new char[in.length()+1];
    for (i =0;i<in.length();i++){
        input[i] = in[i].toLatin1();
    }
    input[i] = '\0';

    std::string resultCode = "";
    std::string output = "";
    BinTree* tree = new(BinTree);


    encode(input, output, inputLen, tree,resultCode);

    delete[] input;

    output = output +"\n"+"Result code:\n"+resultCode;

    QString answ = QString::fromStdString(output);

    Node* root = tree->getRoot();
    tree->freeMem(root);

    QFile out("C:\\Users\\miair\\Desktop\\Results_lr5\\output.txt");
    if (out.open(QIODevice::WriteOnly)){
        out.write(answ.toUtf8());
    }
    out.close();
    return 0;
}
```