

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

КУРСОВАЯ РАБОТА
по дисциплине « АиСД »

Тема: ДИНАМИЧЕСКОЕ КОДИРОВАНИЕ И ДЕКОДИРОВАНИЕ ПО ХАФФМАНУ

Студент гр. 9384

Звега А.Р.

Преподаватель

Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студент Звега А.Р.

Группа 9384

Тема работы : Динамическое кодирование и декодирование по Хаффману – текущий контроль.

Исходные данные:

"Текущий контроль" - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.

Содержание пояснительной записки:

«Содержание», «Введение», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 15.11.2000

Дата сдачи реферата: 24.12.2000

Дата защиты реферата: 24.12.2000

Студент		Звега А.Р.
Преподаватель		Ефремов М.А.

АННОТАЦИЯ

Основное содержание курсового проекта — создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Требуется реализовать алгоритмы кодирования и декодирования по Хаффману. Для этого нужно реализовать структуру бинарного дерева, и алгоритмы которые будут генерировать задания.

SUMMARY

The main content of the course project is the creation of a program for generating tasks with answers to them for conducting monitoring among students. It is required to implement Huffman encoding and decoding algorithms. To do this, you need to implement the structure of a binary tree, and algorithms that will generate tasks.

СОДЕРЖАНИЕ

	Введение	5
1.	Анализ задачи.	6
1.1.	Основные теоретические сведения.	6
1.2.	План работы с программой.	7
1.3	Взаимодействие с программой.	7
2.	Визуализация.	8
2.1.	Реализация интерфейса.	8
3.	Тестирование.	9
3.1.	Тестирование приложения.	9
	Заключение	11
	Список использованных источников	12
	Приложение А. Исходный код.	13

ВВЕДЕНИЕ

Цель работы - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Требуется реализовать алгоритмы кодирования и декодирования по Хаффману, а также выводы в файл и на экран посредством GUI.

1. АНАЛИЗ ЗАДАЧИ

1.1. Основные теоретические сведения

Целью курсовой работы является создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Требуется реализовать алгоритмы кодирования и декодирования по Хаффману, а также выводы в файл и на экран посредством GUI.

Была реализована структура CodeTree. А так же функции для работы с ней:

`make_leaf` - создает лист.

`make_node` - создает узел.

`destroy` - уничтожает дерево.

`fill_symbols_map` - заполняет листья значениями.

Заполнение дерева реализовано через очередь.

Пользователь вводит строку или генерирует ее случайно. После чего к ней применяется ФГК алгоритм. Он позволяет динамически регулировать дерево Хаффмана, не имея начальных частот. В ФГК дереве Хаффмана есть особый внешний узел, называемый 0-узел, используемый для идентификации входящих символов. То есть, всякий раз, когда встречается новый символ — его путь в дереве начинается с нулевого узла. Самое важное — то, что нужно усекать и балансировать ФГК дерево Хаффмана при необходимости, и обновлять частоту связанных узлов. Как только частота символа увеличивается, частота всех его родителей должна быть тоже увеличена. Это достигается путём последовательной перестановки узлов, поддеревьев или и тех и других.

Важной особенностью ФГК дерева является принцип братства (или соперничества): каждый узел имеет два потомка (узлы без потомков называются листьями) и веса идут в порядке убывания.

Декодирование выполняется путем перехода к левому или правому дереву, в зависимости от символа ('0' лево, '1' право), и смещением индекса проверяемого символа строки. Если, идя по строке и дереву, программа приходит в лист (нет левого и правого дерева), то в ответ записывается

соответствующий символ. Дерево возвращается в корень, индекс переходит на следующий символ строки. Цикл работает пока не дойдет до конца строки. Получается раскодированное сообщение, которое было изначально.

Затем на основе полученного дерева генерируются задания.

1.2. План работы программы.

1. Пользователь вводит строку или генерирует ее случайно.
2. Создается дерево Хаффмена.
3. Генерируются ответы к заданиям.
4. Ответы выводятся справа.
5. Пользователь сохраняет ответы путем копирования.
6. При необходимости создании других вариантов нужно повторить предыдущие шаги с другими входными данными.

1.3. Взаимодействие с программой.

Программа поддерживает только английский язык и цифры. Для проверки используется QRegExr и QRegExrValidator. Программа может быть использована, как для создания теста, так и для решения(в случаи утери ответов). Соответственно, пользователю необязательно сохранять ответы, так как при наличии большого количества вариантов нужно помимо текстовых ответов, сохранять и рисунки деревьев. И как следствие, программу можно использовать для генерации строк и проверки ответов.

Использование программы для проверки, кажется более удобным, нежели создании таблиц с ответами.

2. ВИЗУАЛИЗАЦИЯ

2.1. Реализация интерфейса.

Визуализация заданий и ответов выполняется с помощью классов QGraphicsScene и QGraphicsView QTextBrowser фреймворка Qt.

Слева находятся задания, а справа ответы.

Получается 6 блоков. Текстовые ответы даются на все задания кроме 2.3, ответом на него является дерево поэтому предусмотрена функция сохранения картинки.

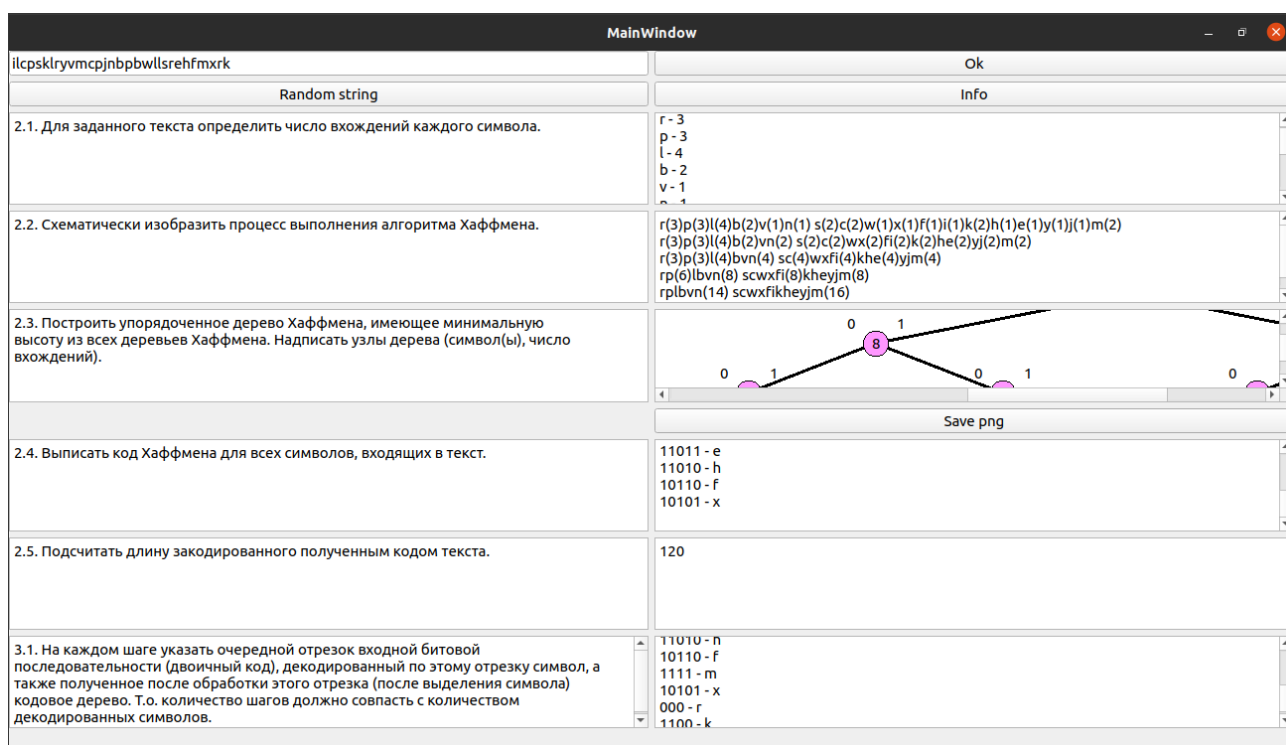


Рисунок 1 — Пример вывода ответов.

Есть кнопка info, которая выводит информацию о том как взаимодействовать с программой.

3. ТЕСТИРОВАНИЕ

3.1. Тестирование приложения.

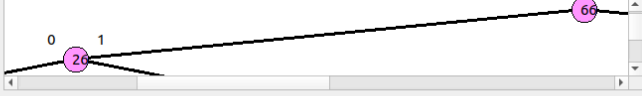
MainWindow	
asafsgctwysocddag245wn65jbkiorpce516865ejakdgbakhbyokesk0896546151	Ok
Random string	Info
2.1. Для заданного текста определить число вхождений каждого символа.	e - 3 b - 3 1 - 3 c - 3 j - 3
2.2. Схематически изобразить процесс выполнения алгоритма Хаффмена.	e(3)b(3)1(3)c(3)j(3)d(3)n(1)p(1)w(2)s(4) 8(2)4(2)h(1)f(1)0(1)r(1)y(2)o(3)6(5)k(5)a(5)g(3)9(1)t(1)2(1)5(6) e(3)b(3)1(3)c(3)j(3)d(3)n(1)p(1)w(2)s(4) 8(2)4(2)h(1)f(1)0(1)r(1)y(2)o(3)6(5)k(5)a(5)g(3)9(1)t(2)5(6) e(3)b(3)1(3)c(3)j(3)d(3)np(2)w(2)s(4) 8(2)4(2)hf(2)0r(2)y(2)o(3)6(5)k(5)a(5)g(3)9t2(3)5(6)
2.3. Построить упорядоченное дерево Хаффмена, имеющее минимальную высоту из всех деревьев Хаффмена. Надписать узлы дерева (символ(ы), число вхождений).	
	Save png
2.4. Выписать код Хаффмена для всех символов, входящих в текст.	1101 - a 0111 - s 100101 - f 11100 - g 0011 - c
2.5. Подсчитать длину закодированного полученным кодом текста.	298
3.1. На каждом шаге указать очередной отрезок входной битовой последовательности (двоичный код), декодированный по этому отрезку символ, а также полученное после обработки этого отрезка (после выделения символа) кодовое дерево. Т.о. количество шагов должно совпасть с количеством декодированных символов.	1101011111010010101111110000111101100110110100011101010011010101011101 111001110111100011110110101000101111101000001100010010101100111011001 0011000011110010101110000101111110000010011011100010111100000111011100100 100000110100101011100000001111100100110100001110101011111100011011001011 110010

Рисунок 2 — тестирование программы при вводе пользователем.

Программа корректно дает ответы на задания.

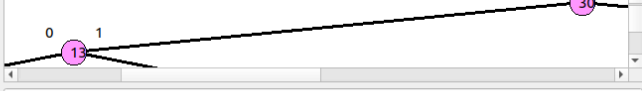
MainWindow	
avnwypdinwdrilacvanhelkovkedcax	Ok
Random string	Info
2.1. Для заданного текста определить число вхождений каждого символа.	d - 3 x - 1 i - 1 o - 1 n - 3
2.2. Схематически изобразить процесс выполнения алгоритма Хаффмена.	d(3)x(1)i(1)o(1)n(3)e(2)k(2) p(1)h(1)l(2)a(4)c(2)w(2)r(1)y(1)v(3) d(3)x(1)io(2)n(3)e(2)k(2) ph(2)l(2)a(4)c(2)w(2)ry(2)v(3) d(3)xio(3)n(3)ek(4) phl(4)a(4)cw(4)ryv(5) dxio(6)nek(7) phla(8)cwryv(9) dxione(13) phlacwryv(17)
2.3. Построить упорядоченное дерево Хаффмена, имеющее минимальную высоту из всех деревьев Хаффмена. Надписать узлы дерева (символ(ы), число вхождений).	
	Save png
2.4. Выписать код Хаффмена для всех символов, входящих в текст.	101 - a 1111 - v 010 - n 1101 - w 11101 - y
2.5. Подсчитать длину закодированного полученным кодом текста.	116
3.1. На каждом шаге указать очередной отрезок входной битовой последовательности (двоичный код), декодированный по этому отрезку символ, а также полученное после обработки этого отрезка (после выделения символа) кодовое дерево. Т.о. количество шагов должно совпасть с количеством декодированных символов.	101111101011011110110000000011001011010001110010011011100111110101010001 0110100101100111111011101101000011001010010 101 - a 1111 - v 010 - n

Рисунок 3 — тестирование программы при генерации случайной строки.

Не важно как вводится строка, так как применяется одни и те же алгоритмы.

ЗАКЛЮЧЕНИЕ

Была создана программа, которая генерирует задания для контроля. В ходе работы были получены навыки создания интерфейса, кодирования и декодирования по Хаффману. Так же были улучшены навыки работы с Qt.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритмы: построение и анализ : пер. с англ. / Т. Кормен и др. – 2-е изд. – М. : Издательский дом «Вильямс», 2007, 2009

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД

Название файла: main.cpp

```
#include "mainwindow.h"
#include <QTextCodec>
#include <QApplication>

int main(int argc, char *argv[])
{
    QTextCodec::setCodecForLocale(QTextCodec::codecForName("UTF-8")); //изменения
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Название файла: mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <math.h>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    scene = new QGraphicsScene(0,0,0,0);
    ui->setupUi(this);
    ui->graphicsView->setScene(scene);
    QRegExp rx("[A-Z;a-z;0-9]*");
    QRegExpValidator *validator = new QRegExpValidator(rx,this);
    ui->lineEdit->setValidator(validator);
}

MainWindow::~MainWindow()
{
    destroy(Tree);
    delete ui;
}

void MainWindow::on_startButtom_clicked()
{
    taskTable.clear();
    taskCode.clear();
    taskText.clear();
    taskNumL.clear();
}
```

```

taskText = ui->lineEdit->text().replace(" ", "");
if(taskText.length() <= 0)
    return;
Tree = haffman(taskText.toStdString().c_str());
taskCode = encode(Tree, taskText.toStdString().c_str());
taskText = ui->lineEdit->text().replace(" ", "");
printTree(Tree, 0);
printTask();
scene->setBackgroundBrush(Qt::white);
scene->clearSelection();
scene->setSceneRect(scene->itemsBoundingRect());
haffman22(Tree, (height(Tree, 0))-1);
decode(Tree, taskCode.toStdString().c_str());
}

void MainWindow::printTask(){
    ui->n21->setText(taskNumL);
    ui->n24->setText(taskTable);
    ui->n25->setText(QString().setNum(taskCode.toStdString().length()));
}

CodeTree* MainWindow::haffman(const Symbol* symbols, int len)
{
    PriorityQueue<CodeTree*>* queue = create_pq<CodeTree*>(len);
    for (int i = 0; i < len; ++i){
        push(queue, symbols[i].weight, make_leaf(symbols[i]));
    }
    while (sizeQ(queue) > 1) {
        CodeTree* ltree = pop(queue);
        CodeTree* rtree = pop(queue);
        int weight = ltree->s.weight + rtree->s.weight;
        CodeTree* node = make_node(weight, ltree, rtree);
        ltree->parent = node;
        rtree->parent = node;
        push(queue, weight, node);
        if(ltree->s.c)
            showHaffman += QString(ltree->s.c) + " " +QString().setNum(ltree->s.weight) + ";";
        else{
            showHaffman += "\n";
        }
    }
    CodeTree* result = pop(queue);
    destroy_pq(queue);
    return result;
}

CodeTree* MainWindow::haffman(const char* message) {
    Symbol symbols[ UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols[i].c = i + CHAR_MIN;

```

```

        symbols[i].weight = 0;
    }
    int size = strlen(message);
    for (int i = 0; i < size; ++i)
        symbols[message[i] - CHAR_MIN].weight++;
    std::sort(symbols, symbols + UCHAR_MAX, symbol_greater);
    int len = 0;
    while (symbols[len].weight > 0 && len < UCHAR_MAX) len++;
    return haffman(symbols, len);
}

char* MainWindow::decode(const CodeTree* tree, const char* code) {
    QString ans;
    ans += taskCode + "\n";
    char* message = new char[MAX_CODE_LEN];
    int index = 0;
    int len = strlen(code);
    const CodeTree* v = tree;
    for (int i = 0; i < len; ++i) {
        if (code[i] == '0'){
            ans += "0";
            v = v->left;
        }
        else{
            ans += "1";
            v = v->right;
        }
        if (is_leaf(v)) {
            ans += " - " + QString(v->s.c) + "\n";
            message[index++] = v->s.c;
            v = tree;
        }
    }
    ui->n31->setText(ans);
    message[index] = '\0';
    return message;
}

char* MainWindow::encode(const CodeTree* tree, const char* message){
    char* code = new char[MAX_CODE_LEN];
    const CodeTree** symbols_map = new const CodeTree * [UCHAR_MAX];
    for (int i = 0; i < UCHAR_MAX; ++i) {
        symbols_map[i] = nullptr;
    }
    fill_symbols_map(tree, symbols_map);
    int len = strlen(message);
    int index = 0;
    char path[UCHAR_MAX];
    std::string check;
    for (int i = 0; i < len; ++i) {
        const CodeTree* node = symbols_map[message[i] - CHAR_MIN];
        int j = 0;

```

```

while (!is_root(node)) {
    if (node->parent->left == node)
        path[j++] = '0';
    else
        path[j++] = '1';
    node = node->parent;
}
path[j] = '\0';
while (j > 0) {
    code[index++] = path[--j];
    if (-1 == check.find(symbols_map[message[i] - CHAR_MIN]->s.c)){
        taskTable += path[j];
    }
}
if (-1 == check.find(symbols_map[message[i] - CHAR_MIN]->s.c)) {
    check.push_back(symbols_map[message[i] - CHAR_MIN]->s.c);
    taskTable += " - ";
    taskTable += symbols_map[message[i] - CHAR_MIN]->s.c;
    taskTable += "\n";
}

}
code[index] = 0;
delete[] symbols_map;
return code;
}

void MainWindow::printTree(CodeTree* Tree, int index){
    int x = 300*15;
    if(Tree){
        if(index == 0){
            scene = new QGraphicsScene(0,0,x,100*10);
            ui->graphicsView->setScene(scene);
            printTreeL(Tree->left, index+1, x/2);
            printTreeR(Tree->right, index+1, x/2);
            scene->addEllipse(x/2,50*(index+1),25,25,QColor(0,0,0),QColor(255,150,255));
            scene->addText(QString().setNum(Tree->s.weight))->setPos(x/2+5,50*(index+1));
            scene->addText(QString("0"))->setPos(x/2+5-25,50*(index+1)-20);
            scene->addText(QString("1"))->setPos(x/2+5+25,50*(index+1)-20);
        }
    }
}

void MainWindow::printTreeL(CodeTree* Tree, int index, int offset){
    if(Tree){
        int x = offset - 4*pow(2,8-index);
        scene->addLine(x + 12.5,50*(index+1)+12.5,offset+12.5,50*index +12.5,QPen(Qt::black,3));
        printTreeL(Tree->left, index+1, x);
        printTreeR(Tree->right, index+1, x);
        scene->addEllipse(x,50*(index+1),25,25,QColor(0,0,0),QColor(255,150,255));
        if(Tree->left || Tree->right){
            scene->addText(QString().setNum(Tree->s.weight))->setPos(x+5,50*(index+1));
        }
    }
}

```



```

        scene->addText(QString("0"))->setPos(x+5-25,50*(index+1)-20);
        scene->addText(QString("1"))->setPos(x+5+25,50*(index+1)-20);
    }
    else{
        scene->addText(QString(Tree->s.c))->setPos(x+5,50*(index+1));
        scene->addText(QString().setNum(Tree->s.weight))->setPos(x+5,50*(index+1)-25);
        taskNumL += QString(Tree->s.c) + " - " + QString().setNum(Tree->s.weight) + "\n";
    }
}
}

void MainWindow::printTreeR(CodeTree* Tree, int index, int offset){
    if(Tree){
        int x = offset + 4*pow(2,8-index);
        scene->addLine(x + 12.5,50*(index+1)+12.5,offset+12.5,50*index + 12.5,QPen(Qt::black,3));
        printTreeL(Tree->left, index+1, x);
        printTreeR(Tree->right, index+1, x);
        scene->addEllipse(x,50*(index+1),25,25,QColor(0,0,0),QColor(255,150,255));
        if(Tree->left || Tree->right){
            scene->addText(QString("0"))->setPos(x+5-25,50*(index+1)-20);
            scene->addText(QString("1"))->setPos(x+5+25,50*(index+1)-20);
            scene->addText(QString().setNum(Tree->s.weight))->setPos(x+5,50*(index+1));
        }
        else{
            scene->addText(QString(Tree->s.c))->setPos(x+5,50*(index+1));
            scene->addText(QString().setNum(Tree->s.weight))->setPos(x+5,50*(index+1)-25);
            taskNumL += QString(Tree->s.c) + " - " + QString().setNum(Tree->s.weight) + "\n";
        }
    }
}

void MainWindow::resizeEvent(QResizeEvent *event){
    if(event){
        int h = ui->centralwidget->height();
        int w = ui->centralwidget->width();
        ui->layout->setGeometry(QRect(0,0,w,h));
    }
}

void MainWindow::on_info_clicked()
{
    QMessageBox::information(this, "Info", "Поддерживается только английский.\n"
        "Напишите строку и нажмите ок для построения дерева или\n"
        "нажмите 'random string'.\n"
        "В узле если это не лист показывается вес узла.\n"
        "В листе показывается символ узла, а над ним его вес(число\n"
        "вхождений символа).\n"
        "Если узел не лист показывается слева и справа '0' или '1'.\n"
        "Можно сохранить картинку дерева.");
}

```

```

void MainWindow::on_save_clicked()
{
    QImage image(scene->sceneRect().size().toSize(), QImage::Format_ARGB32);
    image.fill(Qt::transparent);

    QPainter painter(&image);
    scene->render(&painter);
    image.save("tree.png");
}

void MainWindow::on_random_clicked()
{
    taskTable.clear();
    taskCode.clear();
    taskText.clear();
    taskNumL.clear();

    for(int i = 0; i < 30; i++)
        taskText.append(char('a' + rand() % ('z' - 'a')));
    ui->lineEdit->setText(taskText);
    if(taskText.length() <= 0)
        return;
    Tree = haffman(taskText.toStdString().c_str());
    taskCode = encode(Tree, taskText.toStdString().c_str());
    taskText = ui->lineEdit->text().replace(" ", "");
    printTree(Tree, 0);
    printTask();
    scene->setBackgroundBrush(Qt::white);
    scene->clearSelection();
    scene->setSceneRect(scene->itemsBoundingRect());
    haffman22(Tree, (height(Tree, 0))-1);
    decode(Tree, taskCode.toStdString().c_str());
}

void MainWindow::haffman22(CodeTree *Tree, int index){
    QString ans;
    cout << index << endl;
    for(int i = index; i >= 0; i--){
        QString l = haffmanL(Tree->left, i);
        QString r = haffmanL(Tree->right, i);
        ans += l + " " + r + "\n";
    }
    ans += haffmanR(Tree) + "(" + QString().setNum(Tree->s.weight) + ")";
    ui->n22->setText(ans);
}

QString MainWindow::haffmanL(CodeTree *Tree, int index){
    if(!(Tree->left || Tree->right))
        return QString(Tree->s.c) + "(" + QString().setNum(Tree->s.weight) + ")" ;
    if(index == 0)
        return haffmanR(Tree) + "(" + QString().setNum(Tree->s.weight) + ")"; // +

```

```

        //haffmanR(Tree->left) + "(" + QString().setNum(Tree->left->s.weight)+ ")";
        return haffmanL(Tree->left, index-1)+haffmanL(Tree->right, index-1);

    }
    QString MainWindow::haffmanR(CodeTree *Tree){
        if(!(Tree->left || Tree->right))
            return QString(Tree->s.c);
        return haffmanR(Tree->left)+haffmanR(Tree->right);
    }

    int MainWindow::height(CodeTree* Tree, int index){
        if(Tree->left || Tree->right){
            int l = height(Tree->left, index+1);
            int r = height(Tree->right, index+1);
            if(r > l)
                return r;
            return l;
        }
        return index;
    }
}

```

название файла: mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

```

```

#include <QMainWindow>
#include <QGraphicsScene>
#include <QGraphicsItem>
#include <QMessageBox>
#include <QColor>
#include <QString>
#include <QMessageBox>
#include "priority_queue.h"
#include "code_tree.h"
#include <functional>
#include <algorithm>
#include <climits>
#include <cstring>
#include <iostream>
#include <string>
#include <QPainter>
#include <QRegExpValidator>
#include <QRegExp>

```

```
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();
    CodeTree *Tree;
    QString taskText;
    QString taskTable;
    QString taskCode;
    QString taskNumL;
    QString showHaffman;
    QString encodeTree(CodeTree *, int);
    void printTree(CodeTree *, int);
    void printTreeL(CodeTree *, int, int);
    void printTreeR(CodeTree *, int, int);

    void haffman22(CodeTree *, int);
    QString haffmanL(CodeTree *, int);
    QString haffmanR(CodeTree *);

    void printTask();
    void resizeEvent(QResizeEvent *event);

    CodeTree* haffman(const Symbol* symbols, int len);
    CodeTree* haffman(const char* message);
    char* decode(const CodeTree* tree, const char* code);
    char* encode(const CodeTree* tree, const char* message);
    QString n22;

    int height(CodeTree *, int index);

    QGraphicsScene *scene;
private slots:
    void on_startButton_clicked();

    void on_info_clicked();

    void on_save_clicked();

    void on_random_clicked();

private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H

название файла: code_tree.cpp
#include "code_tree.h"
#include "iostream"
#include <climits>
#include <cstring>

```

```

#include <string>

bool symbol_less(const Symbol& l, const Symbol& r){
    return l.weight < r.weight;
}

bool symbol_greater(const Symbol& l, const Symbol& r){
    return l.weight > r.weight;
}

CodeTree* make_leaf(const Symbol& s){
    return new CodeTree{ s, nullptr, nullptr, nullptr };
}

CodeTree* make_node(int weight, CodeTree* left, CodeTree* right){
    Symbol s{ 0, weight };
    return new CodeTree{ s, nullptr, left, right };
}

bool is_leaf(const CodeTree* node){
    return node->left == nullptr && node->right == nullptr;
}

bool is_root(const CodeTree* node){
    return node->parent == nullptr;
}

void fill_symbols_map(const CodeTree* node, const CodeTree** symbols_map){
    if (is_leaf(node))
        symbols_map[node->s.c - CHAR_MIN] = node;
    else {
        fill_symbols_map(node->left, symbols_map);
        fill_symbols_map(node->right, symbols_map);
    }
}

void destroy(CodeTree* tree){
    if (tree == nullptr) return;
    destroy(tree->left);
    destroy(tree->right);
    delete tree;
    tree = nullptr;
}

```

название файла: code_tree.h

```
#ifndef CODE_TREE_H
#define CODE_TREE_H
#define MAX_CODE_LEN 1000

struct Symbol {
    char c;
    int weight;
};
bool symbol_less(const Symbol & l, const Symbol & r);
bool symbol_greater(const Symbol& l, const Symbol& r);

struct CodeTree {
    Symbol s;
    CodeTree* parent;
    CodeTree* left;
    CodeTree* right;
};

CodeTree* make_leaf(const Symbol& s);
CodeTree* make_node(int weight, CodeTree* left, CodeTree* right);
bool is_leaf(const CodeTree* node);
bool is_root(const CodeTree* node);
void destroy(CodeTree* tree);
void fill_symbols_map(const CodeTree* node, const CodeTree** symbols_map);
#endif // CODE_TREE_H
```

название файла: priority_queue.h

```
#ifndef PRIORITY_QUEUE_H
#define PRIORITY_QUEUE_H
#include <utility>
#include <iostream>

using namespace std;

template <typename T>
struct PriorityQueueItem {
    int key;
    T data;
};

template <typename T>
struct PriorityQueue {
    int size_;
    int capacity_;
    PriorityQueueItem<T>* heap_;
};

template <typename T>
```

```

PriorityQueue<T>* create_pq(int capacity)
{
    PriorityQueue<T>* pq = new PriorityQueue<T>;
    pq->heap_ = new PriorityQueueItem<T>[capacity];
    pq->capacity_ = capacity;
    pq->size_ = 0;
    return pq;
}
template <typename T>
int sizeQ(PriorityQueue<T>* pq)
{
    return pq->size_;
}
template <typename T>
void sift_up(PriorityQueue<T>* pq, int index)
{
    int parent = (index - 1) / 2;
    while (parent >= 0 && pq->heap_[index].key < pq->heap_[parent].key) {
        std::swap(pq->heap_[index], pq->heap_[parent]);
        index = parent;
        parent = (index - 1) / 2;
    }
}

template <typename T>
bool push(PriorityQueue<T>* pq, int key, const T& data)
{
    if (pq->size_ >= pq->capacity_) return false;
    pq->heap_[pq->size_].key = key;
    pq->heap_[pq->size_].data = data;
    pq->size_++;
    sift_up(pq, pq->size_ - 1);
    return true;
}
template <typename T>
void sift_down(PriorityQueue<T>* pq, int index)
{
    int l = 2 * index + 1;
    int r = 2 * index + 2;
    int min = index;
    if (l < pq->size_ && pq->heap_[l].key < pq->heap_[min].key)
        min = l;
    if (r < pq->size_ && pq->heap_[r].key < pq->heap_[min].key)
        min = r;
    if (min != index) {
        std::swap(pq->heap_[index], pq->heap_[min]);
        sift_down(pq, min);
    }
}

template <typename T>
T pop(PriorityQueue<T>* pq)

```

```

{
    std::swap(pq->heap_[0], pq->heap_[pq->size_ - 1]);
    pq->size_--;
    sift_down(pq, 0);
    return pq->heap_[pq->size_].data;
}

template <typename T>
void destroy_pq(PriorityQueue<T>* pq)
{
    delete[] pq->heap_;
    delete pq;
}
#endif // PRIORITY_QUEUE_H

```