**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: 15.11.2020 | | |
| Дата сдачи реферата: 24.12.2020 | | |
| Дата защиты реферата: 24.12.2020 | | |
| 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.

Был 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 листами) 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 QRegExp и QRegExpValidator. 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.

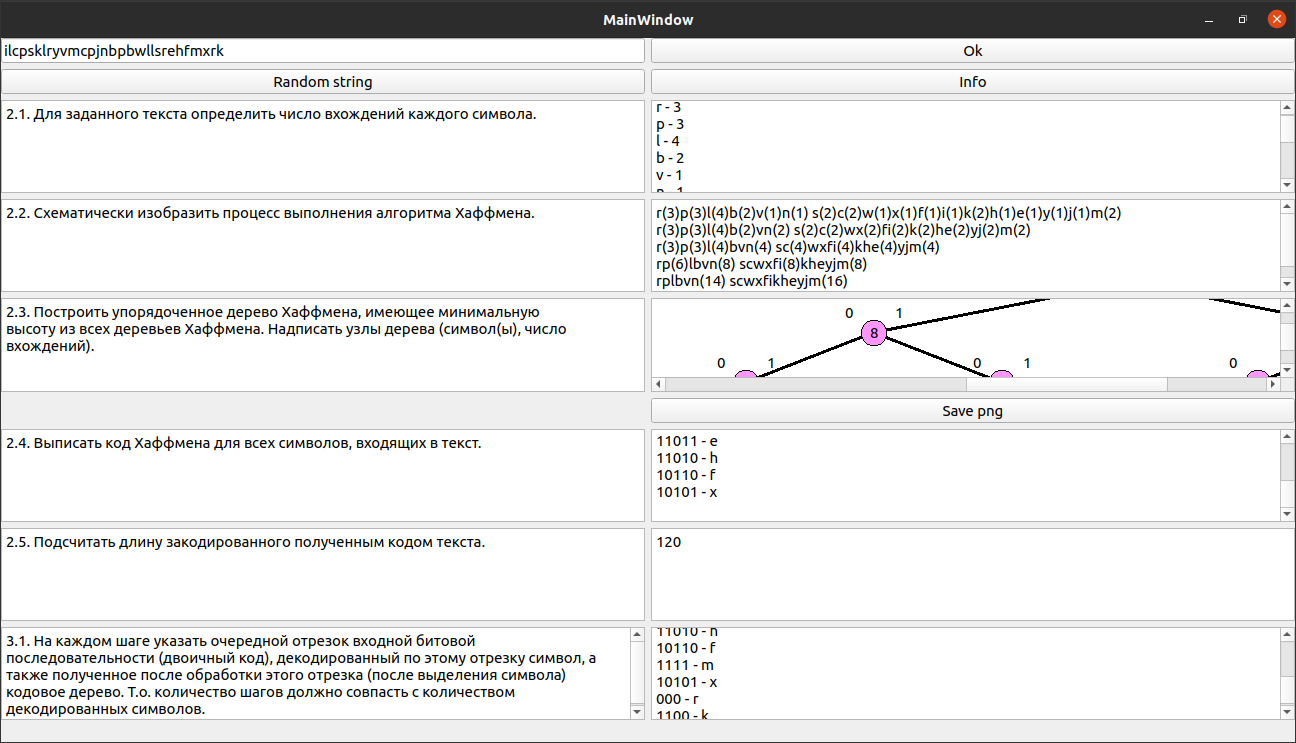
**2. Визуализация**

**2.1. Реализация интерфейса.**

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

Слева находится задания, а справа ответы.

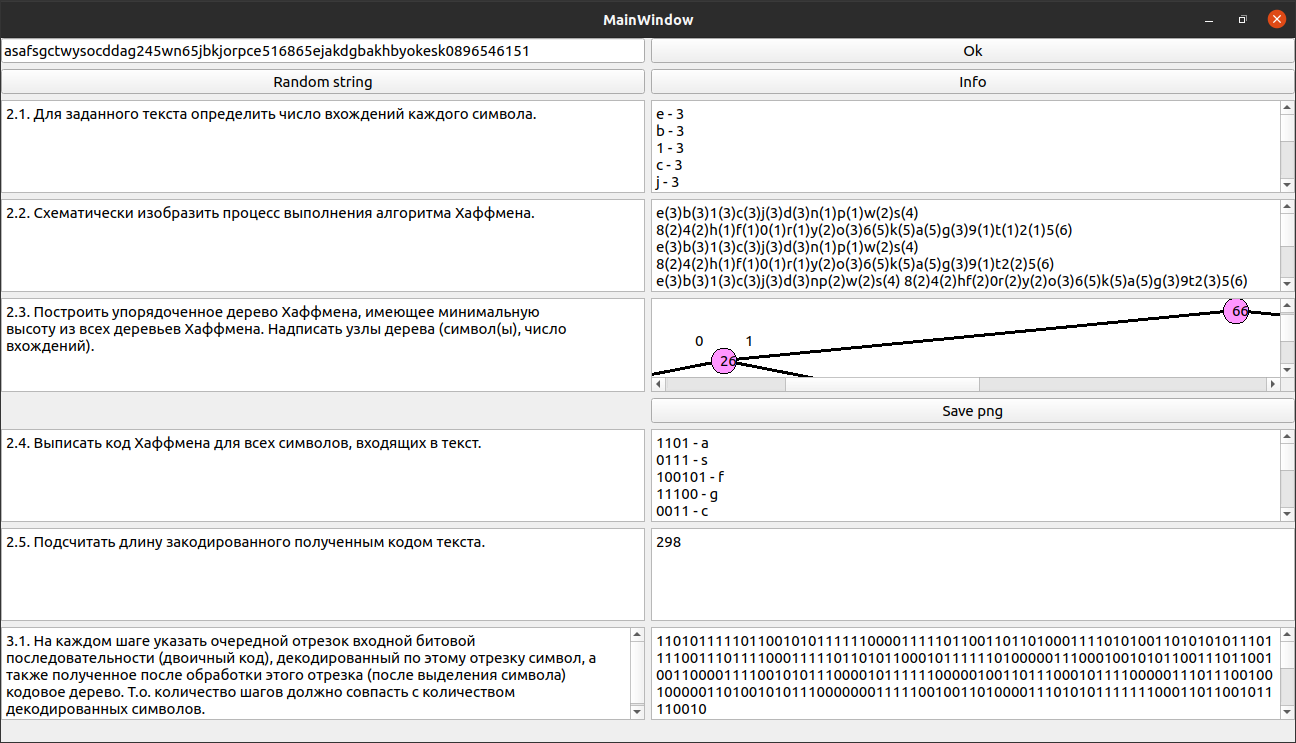
Получается 6 блоков. Текстовые ответы даются на все задания кроме 2.3, ответом на него является дерево поэтому предусмотрена функция сохранения картинки.

Рисунок 1 — Пример вывода ответов.

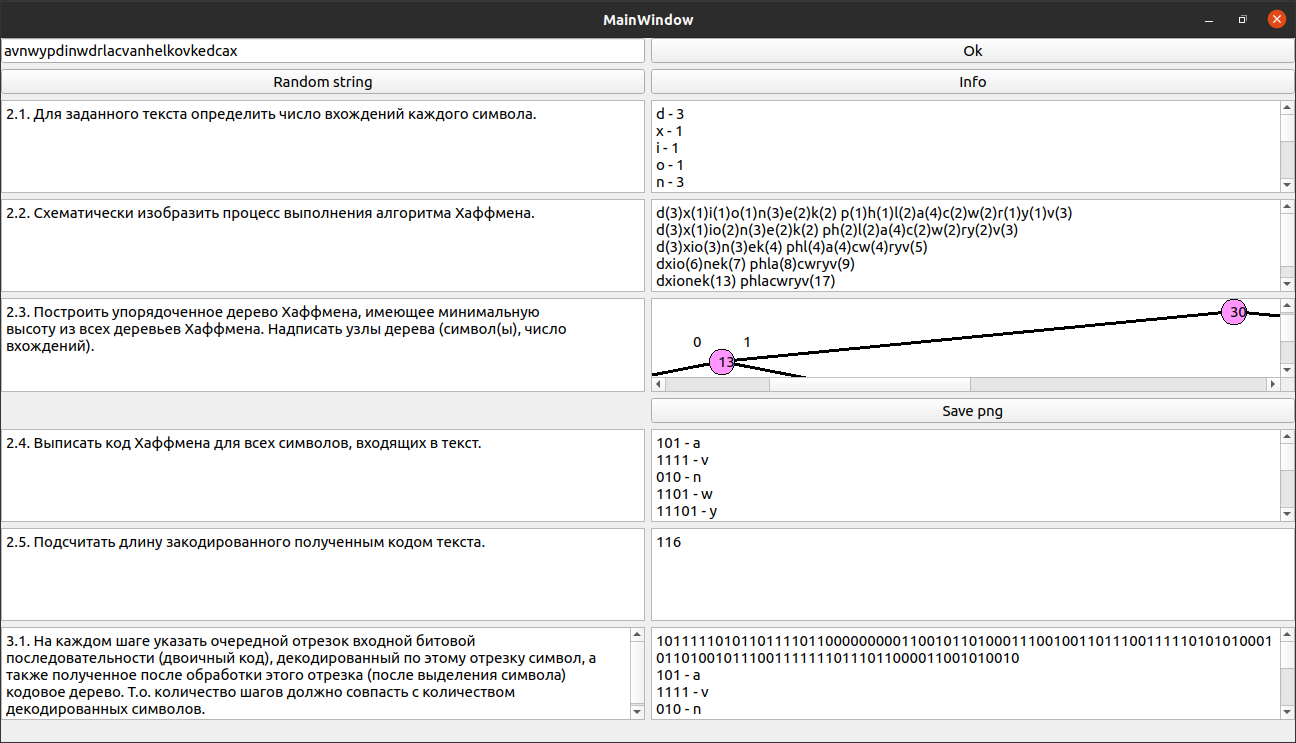
Есть кнопка info, которая выводит информацию о том как взаимодействовать с программой.

**3. Тестирование**

**3.1. Тестирование приложения.**

Рисунок 2 — тестирование программы при вводе пользователем.

Программа корректно дает ответы на задания.

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 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 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\_startButtom\_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