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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные деревья поиска

Студент гр. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Кузьмин Д.И.

Группа 9382

Тема работы: Бинарные деревья поиска

Исходные данные:
В качестве исходных данных выступает набор узлов для построения дерамиды, а также элемент, который будет обработан в процессе работы программы

Содержание пояснительной записки:
«Содержание», «Введение», «Описание функций и структур данных», «Описание алгоритма», «Описание интерфейса пользователя», «Тестирование», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
25-30 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 15.12.2020

Дата защиты реферата: .12.2020

Студент _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М. А.

АННОТАЦИЯ

Курсовая работа представляет собой демонстрацию алгоритма вставки и исключения элемента из дерамиды. Графическая составляющая приложения реализована при помощи фреймворка Qt. Результат работы программы представляет собой пошаговое выполнения алгоритма с визуализацией на каждом шаге. Для демонстрации работы программы было проведено тестирование. Результаты тестирование представлены в табл. 1. Исходный код см. в приложении А.

SUMMARY

The course work is a demonstration of the algorithm for inserting and deleting an element from the treap. The graphics of the application is implemented using the Qt framework. The result of the program is a step-by-step execution of the algorithm with visualization at each step. Testing was carried out to demonstrate how the program works. The test results are presented in table. 1. See Appendix A for the source code.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является изучение принципов работы бинарных деревьев поиска, в частности рандомизированных дерамид поиска. Также в задачи работы входит освоение алгоритмов, реализующих некоторые операции с БДП и разработка программы с демонстрацией их работы.

1. ЗАДАНИЕ

1.1. Задание

Вариант 12. Рандомизированные дерамиды поиска – вставка и исключение. **Демонстрация**

2. ОПИСАНИЕ ФУНКЦИЙ И СТРУКТУР ДАННЫХ

2.1. Описание БДП, используемого в работе

В качестве БДП в работе используется рандомизированная дерамида поиска. Это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. Каждый узел представляет собой пару (ключ, приоритет) и при этом для всех ключей выполнены свойства бинарного дерева(для каждого узла значение ключа левого потомка меньше, а правого потомка — больше(или равно)). И для всех приоритетов выполнены свойства кучи(для каждого узла приоритет обоих потомков меньше). Данная структура данных, в отличие от обычного бинарного дерева поиска определяется однозначно, то есть по известному набору элементов можно построить единственную дерамиду.

2.2. Описание структуры, реализующей элемент дерамиды

Для реализации элементов дерамиды был использован класс TreapElem. Среди его полей: int key – ключ, int prior - приоритет, TreapElem* right – указатель на правого потомка, TreapElem* left – указатель на левого потомка. Также у элемента есть поле QColor color – цвет (по умолчанию белый, но в процессе визулазации алгоритма может меняться).

2.3. Описание структуры, реализующей дерамиду

Для реализации дерамиды был использован класс Treap. Среди его полей: TreapElem* root – указатель на корневой узел типа TreapElem. Также были реализованы некоторые функции (см. следующие пункты).

2.4. Функция вставки элемента

void visualisedInsert(TreapElem*& root, TreapElem a) – рекурсивно реализует и демонстрирует алгоритм вставки(см. описание алгоритма) элемента в дерамиду. Параметры: ссылка на указатель на узел, и сам элемент типа TreapElem. Ничего не возвращает.

2.5. Функция считывания и построения дерамиды

Функция void read(std::istream& is) – считывание и построение дерамиды, на вход принимает ссылку на поток ввода, ничего не возвращает. В программе используется считывание из файла.

2.6. Функция исключения элемента

void visualisedDelete(TreapElem*& root, TreapElem a) – реализует и демонстрирует алгоритм исключения(см. описание алгоритма) элемента из дерамиды. На вход принимает ссылку на указатель на текущий узел и элемент типа TreapElem, который следует удалить.

2.7. Функции поворота

Функции visualisedRotateRight(TreapElem*& el) и visualisedRotateLeft(TreapElem*& el) реализуют и демонстрируют повороты дерамиды соответственно вправо и влево(см. описание алгортма). На вход принимают ссылку на указатель на элемент, относительно которго нужно сделать поворот.

2.8. Функция поиска элемента

Функция int find(TreapElem* root, TreapElem e) реализует поиска элемента e в дерамиде. В качестве параметров принимает указатель на текущий узел и элемент типа TreapElem. Возвращает количество вхождений элемента e.

3. ОПИСАНИЕ АЛГОРИТМА

3.1. Алгоритм вставки элемента

Просматривается текущий узел(начиная с корневого) и если он пустой, то на его месте создается новый, равный вставляемому элементу. Если же узел не пуст, то сравниваются значения его ключа со ключом вставляемого элемента. Далее аналогичные действия проделываются для левого потомка, если ключ узла меньше и для правого, если больше(или равен). Этим сравнением реализуются свойства бинарного дерева поиска. Далее проверяется выполнены ли свойства кучи, то есть в некотором узле значения приоритета больше, чем в потомках. Если нет, то с помощью преобразования БДП, называемого «поворот» достигается соблюдения этих свойств. «Поворот» сохраняет инвариант БДП, но переставляет элементы таким образом, что значения приоритетов может измениться. Если в текущем узле приоритет меньше чем у правого потомка, то осуществляется поворот налево. Аналогично если меньше, чем у левого потомка, то направо. В среднем сложность вставки — O(logn).

3.2. Алгоритм исключения элемента

Сначала проверяется, присутствует ли элемент в дерамиде. Если нет, то выводится об этом сообщение и исключение не реализуется. Если элемент присутствует, то выполняются следующие действия.

Просматривается текущий узел(начиная с корневого) и проверяется совпадение его с удаляемым элементом. Если они равны, то в зависимости от того какого типа этот узел(лист, имеет одного потомка, имеет двоих потомков) проделываеются некоторые операции. Если элемент - лист, то он просто удаляется. Если элемент имеет одного потомка, то ему присваивается значение этого потомка. Наконец, если элемент имеет двоих потомков, то осуществляется его поворот направо, если левый потомок имеет меньший приоритет, чем правый и налево, если наоборот. Далее для этого же элемента, но уже расположенного в другом месте проделываются те же действия.

Если же равенства удаляемого элемента с текущим нет, то проверяются их ключи и в зависимости от того, какой ключ больше следует проверка либо правого потомка элемента, если ключ удаляемого больше или равен ключу текущего, либо левого, если наоборот.

4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

4.1. Построение дерамиды

Построение дерамиды осуществляется из узлов, перечисленных в текстовом файле, при том, если файл не удается открыть, либо перечисленные узлы некорректно написаны, то строится пустая дерамида(возможность вставки для нее сохраняется).

4.2. Осуществление вставки/исключения элемента

Вставка или исключение элемента производится при нажатии на кнопку insert – для вставки и delete – для исключения. Сам же элемент вводится в текстовое окно, расположенное сверху над кнопками. Элемент вводится в виде (ключ, приоритет) и если ввод не соответсвует этому формату, то об этом сообщается и операции вставки/исключения не выполняются.

5. ТЕСТИРОВАНИЕ

5.1. Тестирование

Таблица 1 — результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарий
1	Набор узлов: (2, 10) (1, 2) Элемент: (5, 3) - вставка	построенная дерамида - (1,2)(2,10)(5,3) (порядок ЛКП)	Сначала была построена дерамида по имеющимся узлам, а затем вставлен элемент (5, 3)
2	Набор узлов: (0, 10) (5, 7) (-5, 20) (3, 4) Элемент: (4, 3) - исключение	Построенная дерамида: (null)(-5,20)(null)(0,10)(3,4) (5,7)(null) (порядок ЛКП)	Элемент не встречается в дерамиде, поэтому исключить его нельзя
3	Набор узлов: (7, 2) (2, 2) (13, 4) Элемент: (3, 4) - вставка	Построенная дерамида: (2,2)(3,4)(7,2)(13,4)(null) (порядок ЛКП)	Построение дерамиды по имеющимся узлам, затем вставка нового — (3, 4)
4	Набор узлов: (20, 20) (-20, 20) (13, 4) (5, 7) Элемент: (13, 4) - исключение	Построенная дерамида: (null)(-20,20)(5,7)(20,20) (null) (порядок ЛКП)	Построение дерамиды по имеющимся узлам, затем исключение узла (13, 4)
5	Набор узлов: (20, 20) (---) (13, 4) (5, 7) Элемент:(13, 4) - вставка	Построенная дерамида: (13, 4) (порядок ЛКП)	Так как ввод узлов некорректный, то была сначала построена пустая дерамида, а затем вставлен элемент(13, 4)
6	Набор узлов: (5, 15) (19, 3) (4, 4) Элемент:(-8*) - вставка	Построенная дерамида: (4,4)(5,15)(19,3) (порядок ЛКП)	Ввод элемента некорректен, поэтому была просто построена дерамида

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были изучены бинарные деревья поиска, в частности рандомизированные дерамиды поиска. Получены навыки реализации алгоритмов, осуществляющих некоторые операции с этими структурами данных. Была разработана программа, визулизирующая алгоритмы вставки и исключения элемента из дерамиды.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Aragon, Cecilia R.; Seidel, Raimund (1989), "Randomized Search Trees" (PDF), Proc. 30th Symp. Foundations of Computer Science (FOCS 1989), Washington, D.C.: IEEE Computer Society Press, pp. 540–545, doi:10.1109/SFCS.1989.63531, ISBN 0-8186-1982-1

2. Treaps.URL: http://faculty.cs.niu.edu/~freedman/340/340notes/340treap.htm (дата обращения: 13.12.2020)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл treap.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <sstream>
#include <cstring>
#include <QString>
#include <QColor>

//элемент деремиды
class TreapElem {
public:
	int key;
	int prior;
	TreapElem* right = nullptr;
	TreapElem* left = nullptr;
	QColor color = Qt::white;
	TreapElem(int key, int prior) : key(key), prior(prior) {}
	TreapElem(int key):key(key) {
		prior = rand() % 100;
	}
	TreapElem() {}
	friend std::istream& operator>>(std::istream& in, TreapElem& e) {
//оператор ввода из потока. каждый элемент представляется

			//в виде(x, y), где x - ключ, y - приоритет
		while (in.get() != '(' && !in.eof()) {}
		in >> e.key;
		while (in.get() != ',' && !in.eof()) {}
		in >> e.prior;
		while (in.get() != ')' && !in.eof()) {}
		return in;
	}
	friend std::ostream& operator<<(std::ostream& out, TreapElem* e) {
//оператор вывода в поток

			//пустые элементы выводятся как (null)
		if (e != nullptr)
			out << '(' << e->key << "," << e->prior << ")";
		else out << "(null)";
		return out;
	}
	operator QString()const{
		QString a;
		a.sprintf("(%d,%d)", key, prior);
		return a;
	}
};

bool isLeaf(TreapElem* e);//функция проверяет, является ли элемент листом

//дерамида
```

```
class Treap{
public:
	TreapElem* root;
	Treap() {
		root = nullptr;
	}
	//вставка элемента
	void insert(TreapElem*& root, TreapElem a);

	//поворот налево
	void rotateLeft(TreapElem*& el);

	//поворот направо
	void rotateRight(TreapElem*& el);

	//считывание и построение дерамиды
	void read(std::istream& is);

	//поиск элемента в дерамиде, функция возвращает количество
вхождений
	int find(TreapElem* root, TreapElem e);

};
```

Файл treap.cpp

```
#include "treap.h"

void Treap::insert(TreapElem*& root, TreapElem a){
	//создание нового элемента на месте некоторого пустого, выход из
функции
	if (root == nullptr)
	{
		root = new TreapElem(a);
		return;
	}
	//сравнения с ключом текущего узла
	if (a.key < root->key){
		insert(root->left, a);
	}

	else if (a.key >= root->key) {
		insert(root->right, a);
	}

	//повороты (если требуется)
	if (root->left && root->left->prior > root->prior) {
		rotateRight(root);
	}

	if (root->right && root->right->prior > root->prior) {
		rotateLeft(root);
	}
}

void Treap::read(std::istream& is){
```

```
	std::vector<TreapElem> treapVector;
	while (1) {
		TreapElem a;
		is >> a;
		treapVector.push_back(a);
		char tmp = is.get();
		if (tmp == '\n' || tmp == EOF) break;
	}
	for (auto it : treapVector) insert(root, it);

}
void Treap::rotateLeft(TreapElem*& el) {


	TreapElem* newEl = el->right;
	el->right = newEl->left;
	newEl->left = el;
	el = newEl;
}
void Treap::rotateRight(TreapElem*& el) {

	TreapElem* newEl = el->left;
	el->left = newEl->right;
	newEl->right = el;
	el = newEl;
}
int Treap::find(TreapElem* root, TreapElem e) {

	if (root){
		if (root->key == e.key && root->prior == e.prior  &&
isLeaf(root)) {
			return 1;
		}

		else if (root->key == e.key && root->prior == e.prior && !
isLeaf(root)) {

			return 1 + find(root->right, e);
		}

		else{
			if (e.key >= root->key) {
				return find(root->right, e);
			}
			else {
				return find(root->left, e);
			}
		}
	}
	else return 0;

}
bool isLeaf(TreapElem* e) {
	return (!e->left && !e->right);
}
```

файл cw.h

```
#pragma once
#include <QtWidgets/QMainWindow>
#include "ui_cw.h"
#include "QTimer"
#include "QLabel"
#include <QLineEdit>
#include "QPainter"
#include "treap.h"
#include <QTime>
#include <ctime>
#include <QTextCodec>
#include "QRegExp"
#include <regex>
#include <QPushButton>
class cw : public QMainWindow
{
    Q_OBJECT

public:
    cw(QWidget *parent = Q_NULLPTR);
     Treap treap;
     QLabel* actionLabel;
     QLabel* statusLabel;
     QLabel* inputLabel;
     QLabel* fileTextLabel;
     QPushButton* insertButton;
     QPushButton* deleteButton;
     QPushButton* stopButton;
     QPushButton* continueButton;
     QLineEdit* input;
     QRegExp elemRegExp;
     QRegExp treapRegExp;
     int mDelay = 5000;
     void paintElem(QPainter* painter, TreapElem* e, QRect* qr, float
xcoef);
     void paintTreap(QPainter* painter, Treap treap);
     void DelayMs(int ms);
     void visualisedInsert(TreapElem*& root, TreapElem a);
     void visualisedRotateRight(TreapElem*& root);
     void visualisedRotateLeft(TreapElem*& root);
     void visualisedDelete(TreapElem*& root, TreapElem a);

public slots:
     void insertButtonClicked();
     void deleteButtonClicked();

private:
    Ui::cwClass ui;

protected:
     void paintEvent(QPaintEvent *event);

};
```

Файл cw.cpp

```
#include "cw.h"


QString convertedString(const char* text);
const char* getFileText(const char* filename);
cw::cw(QWidget *parent)
    : QMainWindow(parent)
{
     ui.setupUi(this);

     this->setMinimumSize(800, 600);

     //окно для ввода элемента
     input = new QLineEdit(this);
     input->setGeometry(QRect(400, 0, 100, 60));
     input->setFont(QFont("Tahoma", 13, 13));
     input->setFocus();

     //регулярные выражения для проверки корректности ввода
     elemRegExp = QRegExp("\\s*\\
(\\s*(-?\\d+)\\s*,\\s*(-?\\d+)\\s*\\)\\s*");
     treapRegExp = QRegExp("(\\s*\\
(\\s*(-?\\d+)\\s*,\\s*(-?\\d+)\\s*\\)\\s*)*");



     //окно для вывода содержимого файла
     fileTextLabel = new QLabel(this);
     fileTextLabel->setGeometry(QRect(0, 650, 800, 60));

     //обработка файла
     std::ifstream fs("treap.txt");
     if (!fs.fail()) {
          fs.close();
          QString treapElems(getFileText("treap.txt"));
          fileTextLabel->setText(convertedString("Содержимое файла: ") +
treapElems);
          bool correctTreap = treapRegExp.exactMatch(treapElems);
          if (!correctTreap) {

               fileTextLabel->setText(fileTextLabel->text() +
convertedString("\nНекорректный ввод элементов дерамиды"));
               fileTextLabel->setText(fileTextLabel->text() +
convertedString("\nПостроена пустая дерамида"));
          }
          else {
               std::ifstream ifs("treap.txt");
               fileTextLabel->setText(fileTextLabel->text() +
convertedString("\nДерамида построена"));
               treap.read(ifs);
               ifs.close();
          }
     }
```

```
      else fileTextLabel->setText(convertedString("Не удалось открыть
файл\n Построена пустая дерамида"));

      fileTextLabel->setFont(QFont("Tahoma", 13, 13));

      //инструкция для ввода
      inputLabel = new QLabel(this);
      inputLabel->setGeometry(QRect(0, 0, 500, 60));
      inputLabel->setText(convertedString("Введите элемент в \nформате
(ключ, приоритет)"));
      inputLabel->setFont(QFont("Tahoma", 13, 13));

      //отображаение текущего действия
      actionLabel = new QLabel(this);
      actionLabel->setGeometry(QRect(650, 40, 1300, 60));
      actionLabel->setAlignment(Qt::AlignTop);
      actionLabel->setFont(QFont("Tahoma", 13, 13));

      //отображение текущего статуса
      statusLabel = new QLabel(this);
      statusLabel->setGeometry(QRect(650, 0, 1200, 60));
      statusLabel->setAlignment(Qt::AlignTop);
      statusLabel->setFont(QFont("Tahoma", 13, 13));

      //кнопка для вставки элемента
      insertButton = new QPushButton("INSERT", this);
      insertButton->setGeometry(QRect(0, 120, 100, 50));
      connect(insertButton, SIGNAL(clicked()), this,
SLOT(insertButtonClicked()));
      insertButton->setFocusPolicy(Qt::ClickFocus);

      //кнопка для удаления элемента
      deleteButton = new QPushButton("DELETE", this);
      deleteButton->setGeometry(QRect(120, 120, 100, 50));
      connect(deleteButton, SIGNAL(clicked()), this,
SLOT(deleteButtonClicked()));
      deleteButton->setFocusPolicy(Qt::ClickFocus);
}

//функция осуществляющая задержку приложения(используется при
визулизации)
void cw::DelayMs(int ms) {
      QTime dieTime = QTime::currentTime().addMSecs(ms);
      while (QTime::currentTime() < dieTime) {
            QCoreApplication::processEvents(QEventLoop::AllEvents, 100);
      }
}

//получение содержимого файла, с которого вводится дерамида
const char* getFileText(const char* filename){
      std::ifstream f(filename);
      char* tmp = new char[200];
      char d;
      int i = 0;
      while (f.get(d)) tmp[i++] = d;
      tmp[i] = '\0';
      f.close();
```

```
	return tmp;
}
//графическое изображение ребра между узлами
void connectElemes(QPainter* painter, TreapElem* e1, TreapElem* e2, QRect
qr1, QRect qr2) {
	if (e1 && e2) {
		QPointF f1(qr1.center().x(), qr1.center().y() + 25);
		QPointF f2(qr2.center().x(), qr2.center().y() - 25);;

		painter->drawLine(f1, f2);
	}
}
//графическое изображение элемента и его потомков
void cw::paintElem(QPainter* painter, TreapElem* e, QRect* qr, float
xcoef = 150){
	if (e){
		painter->setBrush(e->color);
		painter->setPen(Qt::black);
		painter->drawEllipse(*qr);
		painter->setBrush(Qt::white);
		painter->drawText(*qr, Qt::AlignCenter, *e);
		connectElemes(painter, e, e->left, *qr, QRect(qr->x() - xcoef,
qr->y() + 100, qr->width(), qr->height()));
		paintElem(painter, e->left, new QRect(qr->x() - xcoef, qr->y()
+ 100, qr->width(), qr->height()), xcoef / 3 * 2);
		connectElemes(painter, e, e->right, *qr, QRect(qr->x() +
xcoef, qr->y() + 100, qr->width(), qr->height()));
		paintElem(painter, e->right, new QRect(qr->x() + xcoef, qr-
>y() + 100, qr->width(), qr->height()), xcoef / 3 * 2);
	}
	else return;
}

//функция, графически изображающая дерамиду
void cw::paintTreap(QPainter* painter, Treap treap){

	this->paintElem(painter, treap.root, new QRect(this->width()/2 ,
200, 50, 50), 256);
}

//событие, при котором отрисовывается дерамида, вызывается по мере
выполнения операций
void cw::paintEvent(QPaintEvent *event){

	QPainter* painter = new QPainter(this);
	paintTreap(painter, treap);
}
//функция, возвращающая корректный кириллический текст
QString convertedString(const char* s){
	QTextCodec *codec1 = QTextCodec::codecForName("CP1251");
	QString text = codec1->toUnicode(s);
	return text;
}
//покраска элемента и его потомков в какой-то один цвет
void colorRecursively(QColor color, TreapElem*& e) {
	if (e) {
		e->color = color;
```

```
            colorRecursively(color, e->left);
            colorRecursively(color, e->right);
      }
      else return;
}
//визулизированный поворот налево
 void cw::visualisedRotateLeft(TreapElem*& el) {

      TreapElem* newEl = el->right;
      colorRecursively(Qt::yellow, newEl);
      colorRecursively(QColor(200, 200, 200), newEl->left);
      colorRecursively(QColor(100, 120, 33, 127), el->left);
      el->color = QColor(255, 0, 0, 127);
      update();
      DelayMs(mDelay);
      el->right = newEl->left;
      newEl->left = el;
      el = newEl;
      update();
      DelayMs(mDelay);
      colorRecursively(Qt::white, newEl);
      el->color = Qt::white;
      update();
}

//визуализированный поворот направо
 void cw::visualisedRotateRight(TreapElem*& el) {

      TreapElem* newEl = el->left;
      colorRecursively(Qt::yellow, newEl);
      colorRecursively(QColor(200, 200, 200), newEl->right);
      colorRecursively(QColor(100, 120, 33, 127), el->right);
      el->color = QColor(255, 0, 0, 127);
      update();
      DelayMs(mDelay);
      el->left = newEl->right;
      newEl->right = el;
      el = newEl;
      update();
      DelayMs(mDelay);
      colorRecursively(Qt::white, newEl);
      el->color = Qt::white;
      update();

}
 //визулизация удаления элемента из дерамиды
void cw::visualisedInsert(TreapElem*& root, TreapElem a) {


      actionLabel->setText(convertedString("Просмотр текущего узла"));
      if (root == nullptr){
            actionLabel->setText(convertedString("Встречен пустой узел.
Создание элемента ") + QString(a)
            + convertedString(" на его месте"));
            root = new TreapElem(a);
            root->color = QColor(0, 0, 255, 127);
            update();
```

```
            DelayMs(mDelay);
            root->color = Qt::white;
            return;
      }
      root->color = Qt::green;
      update();
      DelayMs(mDelay);

      //сравнения ключа искомого элемента с ключом текущего узла
      if (a.key < root->key) {
            actionLabel->setText(convertedString("Значение ключа
вставляемого элемента ") + QString(a)
            + convertedString(" меньше значения ключа текущего элемента ")
+ QString(*root)
            + convertedString("\nПереход к левому потомку"));

            DelayMs(mDelay);
            root->color = Qt::white;
            update();
            visualisedInsert(root->left, a);
      }
      //если ключ вставляемого элемента больше (или равен) ключа текущего
элемента
      else{
            actionLabel->setText(convertedString("Значение ключа
вставляемого элемента ") + QString(a)
            + convertedString(" больше(или равно) значения ключа текущего
элемента ") + QString(*root)
            + convertedString("\nПереход к правому потомку"));

            DelayMs(mDelay);
            root->color = Qt::white;
            update();
            visualisedInsert(root->right, a);
      }

      //повороты (если требуется)
      if (root->left && root->left->prior > root->prior) {
            actionLabel->setText(convertedString("Значение приоритета
левого потомка ") + QString(*root->left)
            + convertedString(" больше чем значение приоритета текущего
элемента ") + QString(*root)
            + convertedString("\nОсуществляется поворот направо\n"));

            visualisedRotateRight(root);
            update();
      }

      if (root->right && root->right->prior > root->prior) {
            actionLabel->setText(convertedString("Значение приоритета
правого потомка: ") + QString(*root->right)
            + convertedString(" больше чем значение приоритета текущего
элемента ") + QString(*root)
            + convertedString("\nОсуществляется поворот налево\n"));

            visualisedRotateLeft(root);
            update();
```

```
	}
}

//визулизированное удаление элемента из дерамиды
void cw::visualisedDelete(TreapElem*& root, TreapElem a) {

	//если элемента нет в дерамиде
	if (root == nullptr){
		return;
	}
	root->color = Qt::green;
	update();
	actionLabel->setText(convertedString("Просмотр текущего
элемента"));
	DelayMs(mDelay);
	root->color = Qt::white;
	if (a.key == root->key && a.prior == root->prior) {
		actionLabel->setText(convertedString("Обнаружено совпадение с
искомым элементом."));
		root->color = QColor(204, 102, 153);
		update();
		DelayMs(mDelay);
		//если элемент является листом, то он просто удаляется
		if (isLeaf(root)) {
			actionLabel->setText(convertedString("Элемент является
листом. Дополнительных изменений структуры дерамиды не требуется"));
			actionLabel->setText(actionLabel->text() +
convertedString("\nУдаление элемента"));
			DelayMs(mDelay);
			root = nullptr;
		}
		//если элемент имеет двух потомков, то необходимо произвести
повороты, пока этот элемент не станет
		//либо листом, либо узлом с только одним потомком
		else if (root->left && root->right){
			actionLabel->setText(convertedString("Элемент имеет двух
потомков. Необходимо произвести повороты."));
			DelayMs(mDelay);

			// в случае, если у левого потомка приоритет ниже, чем у
правого, осуществляется поворот направо
			if (root->left->prior < root->right->prior){

				actionLabel->setText(convertedString("Правый
потомок элемента имеет больший приоритет, чем левый\n"));
				actionLabel->setText(actionLabel->text() +
convertedString("Осуществляется поворот налево"));
				// Поворот налево
				visualisedRotateLeft(root);

				// Прежний элемент теперь на месте левого потомка,
выполняем его удаление
				visualisedDelete(root->left, a);
			}
			//если у правого потомка больший приоритет, то тогда
делается поворот налево
			else{
```

```
                    actionLabel->setText(convertedString("Левый потомок
элемента имеет меньший приоритет, чем правый\n"));
                    actionLabel->setText(actionLabel->text() +
convertedString("Осуществляется поворот направо"));
                    // Поворот налево
                    visualisedRotateRight(root);

                    // Прежний элемент теперь на месте левого потомка,
выполняем его удаление
                    visualisedDelete(root->right, a);
               }
          }
          //удаление узла с одним потомком
          else{
               actionLabel->setText(convertedString("Элемент имеет
одного потомка"));
               TreapElem* child = (root->left) ? root->left : root-
>right;
               colorRecursively(QColor(150, 150, 150), child);
               update();
               DelayMs(mDelay);
               root = child;
               actionLabel->setText(convertedString("Присваивание
текущему элементу значение его потомка"));
               update();
               DelayMs(mDelay);
          }
          //покраска элемента, стоящего на месте удаленного, в белый
цвет
          if (root) colorRecursively(Qt::white, root);
          return;
     }
     //если совпадения текущего элемента с искомым нет, то далее в
зависимости от
     //приоритета производится обход следующих узлов
     if (a.key < root->key) {
          actionLabel->setText(convertedString("Значение ключа
вставляемого элемента - ") + QString(a)
               + convertedString(" меньше значения ключа текущего
элемента - ") + QString(*root) +
               convertedString("\nПереход к левому потомку"));
          DelayMs(mDelay);
          visualisedDelete(root->left, a);
     }
     else {
          actionLabel->setText(convertedString("Значение ключа
вставляемого элемента - ") + QString(a)
               + convertedString(" больше значения ключа текущего
элемента - ") + QString(*root) +
               convertedString("\nПереход к правому потомку"));
          DelayMs(mDelay);
          visualisedDelete(root->right, a);
     }
}
//обработчик нажатия на кнопку удаления
void cw::deleteButtonClicked() {
```

```
     bool correct = elemRegExp.exactMatch(input->text());//проверка
ввода при помощи регулярного выражения
     if (correct) { //если ввод корректен, запускается алгоритм
удаления, кнопки для удаления и вставка становятся неактивными
                         //как и окно для ввода элемента
          input->setDisabled(1);
          int h = elemRegExp.cap(2).toInt();
          int g = elemRegExp.cap(1).toInt();
          TreapElem ex(g, h);
          if (treap.find(treap.root, ex) != 0) {
               deleteButton->setDisabled(1);
               insertButton->setDisabled(1);
               statusLabel->setText(convertedString("Удаление
элемента") + QString(ex));
               visualisedDelete(treap.root, ex);
               actionLabel->setText(convertedString(""));
               statusLabel->setText(convertedString("Удаление
завершено"));
               update();
          }
          else statusLabel->setText(convertedString("Элемент не
присутствует в дерамиде"));
     }
     else statusLabel->setText(convertedString("Некорректный ввод"));
     deleteButton->setDisabled(0);
     insertButton->setDisabled(0);
     input->setDisabled(0);
     input->setFocus();
}
//обработчик нажатия на кнопку вставки
void cw::insertButtonClicked(){

     bool correct = elemRegExp.exactMatch(input->text()); //проверка
ввода при помощи регулярного выражения
     if (correct) { //если ввод корректен, запускается алгоритм вставки,
кнопки для удаления и вставка становятся неактивными
                         //как и окно для ввода элемента
          deleteButton->setDisabled(1);
          insertButton->setDisabled(1);
          input->setDisabled(1);
          int h = elemRegExp.cap(2).toInt();
          int g = elemRegExp.cap(1).toInt();
          TreapElem ex(g, h);
          statusLabel->setText(convertedString("Вставка элемента") +
QString(ex));
          visualisedInsert(treap.root, ex);
          actionLabel->setText(convertedString(""));
          statusLabel->setText(convertedString("Вставка завершена"));
          update();
     }          //иначе выводится сообщение об этом
     else statusLabel->setText(convertedString("Некорректный ввод"));

     deleteButton->setDisabled(0);
     insertButton->setDisabled(0);
     input->setDisabled(0);
     input->setFocus();
```

```
}
```

Файл main.cpp

```
#include "cw.h"
#include <QtWidgets/QApplication>

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