**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: Рандомизированные дерамиды поиска. Вставка и исключение. Демонстрация.**

Студент гр. 9384 Николаев А.А.

Преподаватель _____ Ефремов М.А.

_____

Санкт-Петербург
2020

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ (КУРСОВОЙ ПРОЕКТ)**

Студент Николаев А.А.

Группа 9384

Тема работы : **Рандомизированные дерамиды поиска. Вставка и исключение. Демонстрация.**

Исходные данные:

Для разработки программы использовался графический фреймворк “Qt”. Пользователю предоставляется интерфейс, при помощи которого он без проблем может добавить элементы в дерамиду или удалить их, а также поэтапно рассмотреть процесс добавления или удаления элементов.

Содержание пояснительной записки:

“Содержание ” “Введение” “Описание алгоритма” “Описание структур данных и функций” “Описание пользовательского интерфейса” “Тестирование” “Заключение”

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 03.12.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 28.12.2020

Студент Николаев А.А.

Преподаватель Ефремов М.А.

## АННОТАЦИЯ

Основная цель данной курсовой работы — создание программы позволяющей продемонстрировать процесс добавления и удаления элементов из рандомизированной дерамиды поиска. При написании кода создается структура данных, представляющая собой бинарное дерево поиска, и позволяющая добавить или исключить указанный узел из этого дерева. Эти методы с некоторой вероятностью перестраивают дерево относительно данного узла. Пользователю будет предоставлена возможность поэтапно рассмотреть и изучить эти процессы. Для демонстрации работы данных методов и объяснения того, что происходит на каждом этапе, программа использует графическое окно, на котором схематично отображаются процессы происходящие на данном этапе, а также текстовое окно, на котором объясняется то, что происходит на экране. Ползунок под текстовым окном позволяет переходить от одного этапа к другому.

## SUMMARY

The main goal of this course work is to create a program that demonstrates the process of adding and removing elements from a randomized search deramide. When you write the code, a data structure is created that is a binary search tree and allows you to add or exclude a specified node from this tree. These methods with some probability rebuild the tree relative to the given node. The user will be given the opportunity to step by step review and study these processes. To demonstrate how these methods work and explain what happens at each stage, the program uses a graphical window that schematically displays the processes occurring at this stage, as well as a text window that explains what is happening on the screen. The slider below the text box allows you to move from one stage to the next.

**СОДЕРЖАНИЕ**

# ВВЕДЕНИЕ

В данной курсовой работе объектом исследования является рандомизированная дерамида поиска и методы insert и erase позволяющие добавить и удалить элемент из дерева соответственно. Для демонстрации работы этих методов, пользователю предоставляется графическое окно и текстовое окно в которых расписан каждый этап работы программы, происходящий при вызове этих двух методов.

# 1. ОПИСАНИЕ АЛГОРИТМА

Создается класс Treap. Эта структура данных хранящая в себе пары элементов key (Значение узла дерева) и priority (Приоритет узла дерева). Особенность рандомизированной дерамиды поиска состоит в том, что при использовании такой структуры данных, во всех левых элементах в таком дереве значения key будут меньше корня, а правые больше. Так же и с priority, все значения priority в родительских элементах будут меньше, чем в дочерних.

Сами по себе, priority представляет собой случайно созданное число. Преимущество построения таких деревьев поиска с приоритетами, по сравнению с обычными деревьями поиска состоит в том, что использование приоритетов позволяет решить проблему, при которой в некоторых ситуациях, при вводе некоторых данных, дерево вырождается в обыкновенный линейный список.

В дерамиде описаны следующие методы:

1. Метод split - разбивает дерево на два поддерева, такие, что одно содержит все элементы меньшие чем указанный, а второе большие.
2. Метод merge - метод, обратный по отношению к методу split. Он объединяет два поддерева и возвращает новое дерево в качестве результата.
3. Insert - спускается по дереву, либо пока не дойдет до конца и не вставит туда новый элемент, либо пока не дойдет до элемента, у которого приоритет ниже, чем у новго. Вызовется метод split, который на место этого элемента поставит новый элемент и свяжет его с другими элементами.
4. Erase - осуществляет спуск по дереву либо пока не найдет первый элемент с указанным ключом, либо пока не дойдет до конца дерева. При условии, что указанный элемент всё таки был найден вызовется метод merge, который свяжет наследников удаленного элемента, с его родительским элементом.

## 2. ВИЗУАЛИЗАЦИЯ

Визуализация дерева выполняется с помощью классов QGraphicsScene и QGraphicsView, входящих в состав фреймворка Qt. У объекта класса Treap вызывается метод draw, который при помощи вызова метода maxDepth, узнает максимальную глубину дерева и на основе этих данных рисует на экране дерево так, чтобы исключить возможные ошибки, возникающие при отображении дерева, такие как пересечение ребер дерева. Красным цветом пользователю подсвечивается элемент, с которым сейчас происходят какие-либо операции. Зеленым цветом отображаются все остальные элементы, которые являются потомками данного элемента. Под каждым корнем подписаны его ключ(key) и приоритет(priority).

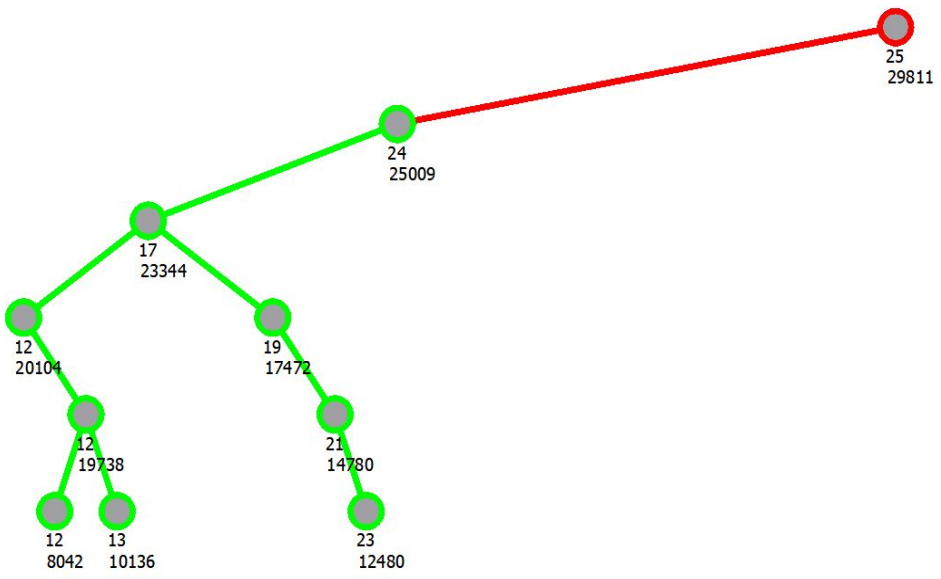


Рис. 1. Визуализация построенного дерева.

## 3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ.

Пользовательский интерфейс осуществляющий взаимодействие пользователя с программой, располагается на верхней панели и представляет из себя два больших раздела “Создание” и “Редактирование”.

В разделе “Создание”, пользователь может добавить элементы в дерево тремя способами. Загрузить из текстового документа. Программа откроет файл и считает первую строку, затем преобразует ее в массив чисел и для каждого из этих чисел вызовет метод Insert. Выбрав второй пункт меню, пользователь сможет вручную добавить любое количество элементов, просто введя их в строку через пробел. И третьим способом, добавить элементы в строку, можно при помощи генератора. Который в диалоговом окне запросит у пользователя количество элементов, которое надо сгенерировать. При желании пользователь может и указать два числа, тем самым задав максимальное и минимальное число которое можно сгенерировать.

Также пользователю предоставляется возможность записать дерево в файл, причем существует три метода записи дерева в файл, каждый из которых представляет собой один из трех основных способов обхода дерева, прямой, центрированный и обратный обход

Во втором разделе “Редактирование”, есть две кнопки Добавить и удалить. Они запрашивают у пользователя ключ и соответственно, либо записывают его, либо удаляют.

Также, для изучения работы дерамиды, предусмотрены графическое окно, где находится графическое представление дерева, текстовое окно, в котором печатаются пояснения к тому, что происходит на графическом окне и наконец ползунок под ним, с помощью которого можно перемещаться по шагам и рассматривать каждый этап. Внутренняя реализация ползунка состоит в следующем: создается вектор, хранящий пары значений: дерево, которое необходимо изобразить вывести в графическое окно и текст, который необходимо вывести в текстовое окно. Во время выполнения программой, каких-либо заданий, вызываемые методы принимают этот вектор и в его конец

записывают копию дерева, которое получилось в момент выполнения этой команды и строку с пояснением.

Когда пользователь двигает ползунок, срабатывает функция, которая принимает значение, на которое сместился ползунок, обращается к элементу вектора, соответствующему данному числу и выводит на экран дерево и текст указанных  по этому адресу.

# 4. ТЕСТИРОВАНИЕ

Ниже представлены изображения показывающие результат работы программы при выборе различных команд.

Тест 1

Генерация 10 элементов. Программе на вход подается команда, создать дерево из 10 случайных чисел. На выходе должно получиться сбалансированное дерево, состоящее из 10 элементов. Сбалансированным, будет считаться дерево, у которого значения ключей всех левых дочерних элементов меньше родительских, правых больше, а так же у такого дерева значения всех приоритетов дочерних элементов, должно быть меньше родительских.

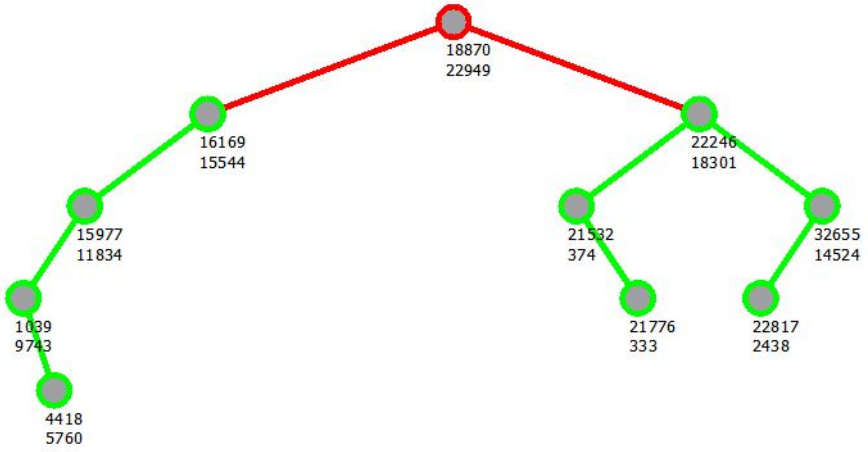


Рисунок 2. Генерация 10 элементов.

На выходе получается результат, удовлетворяющий всем перечисленным условиям.

Тест 2.

Удаление элемента из дерева. На вход программе поступает команда удалить элемент из дерева, программа принимает ключ, который надо удалить, в данном примере надо удалить элемент с ключом 18870. После выполнения команды на экране  и в дереве  не должно остаться указанного элемента, а также дерево  остаться сбалансированным.

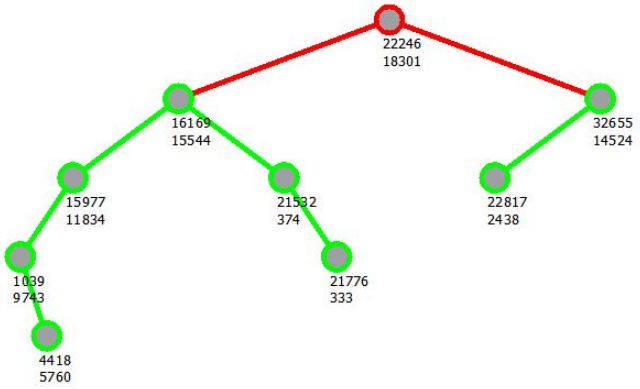


Рисунок 3. Удаление элемента 18870.

Элемент удален из дерева и оно осталось сбалансированным.

Тест 3.

Добавление элемента в дерево. На вход программа принимает значение, в данном примере число 1, которое нужно добавить в дерево. После выполнения задания, в дереве должен появиться указанный элемент и дерево должно остаться сбалансированным.

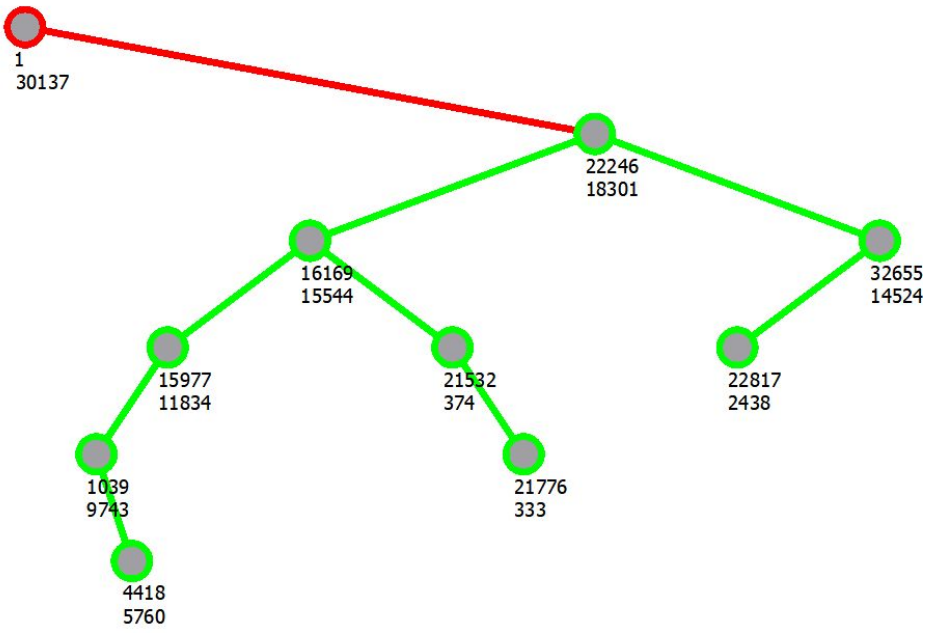


Рисунок 4. Добавление элемента 1 и перестройка дерева.

Элемент добавлен в дерево и оно осталось сбалансированным.

# ЗАКЛЮЧЕНИЕ

При выполнении работы были реализованы методы добавления и исключения элементов, а также разработан интерфейс для наглядной демонстрации пользователю работы алгоритма.

# ПРИЛОЖЕНИЕ А
# ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "mainwindow.h"

#include <QApplication>

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

Название файла: mainwindow.cpp

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

MainWindow::MainWindow(QWidget *parent)
   : QMainWindow(parent)
   , ui(new Ui::MainWindow)
{
   ui->setupUi(this);

   zoom = new graphicsviewzoom(ui->graphicsView);
   zoom->set_modifiers(Qt::NoModifier);

   scene = new QGraphicsScene();
   ui->graphicsView->setScene(scene);

   ui->delete_item->setEnabled(false);
}

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


void MainWindow::on_from_file_triggered()
{
   QString filepath = QFileDialog::getOpenFileName(this, "Load", QDir::homePath(), tr("Load File
(*.txt)"));

   QFile file(filepath);
   if(!file.open(QFile::ReadOnly | QFile::ReadOnly))
   {
```

```
        QMessageBox::warning(this, "Внимание!", "Файл не открыт!");
    }
    else
    {
        QString str = file.readLine();

        QStringList lst = str.split(" ");

        for (qsizetype index = 0; index < lst.size(); index++)
        {
            QString num = lst[index];
            treap.insert(num.toInt(), &list);
            ui->horizontalSlider->setRange(0, list.size() - 1);
        }
    }

    file.close();
}

void MainWindow::on_from_string_triggered()
{
    QString str = QInputDialog::getText(this, "Ввод последовательности.", "Введите
последовательность чисел: ");

    QStringList lst = str.split(" ");
    for (qsizetype index = 0; index < lst.size(); index++)
    {
        QString num = lst[index];
        treap.insert(num.toInt(), &list);
        ui->horizontalSlider->setRange(0, list.size() - 1);
    }
}

void MainWindow::on_add_item_triggered()
{
    int data = QInputDialog::getInt(this, "Добавление элемента.", "Введите ключ: ");
    treap.insert(data, &list);
    ui->horizontalSlider->setRange(0, list.size() - 1);
}

void MainWindow::on_delete_item_triggered()
{
    int data = QInputDialog::getInt(this, "Удаление элемента.", "Введите ключ: ");
    treap.erase(data, &list);
    ui->horizontalSlider->setRange(0, list.size() - 1);
}

void MainWindow::on_horizontalSlider_sliderMoved(int position)
{
    ui->textBrowser->clear();
    ui->textBrowser->append(list[position].second);
```

```
    scene->clear();
    list[position].first->draw(scene);
}

void MainWindow::on_generate_triggered()
{
    int size = QInputDialog::getInt(this, "Генерация элементов.", "Введите количество
элементов: ");

    if (size <= 0)
    {
        QMessageBox::warning(this, "Внимание!", "Неверные данные!");
    }
    else
    {
        QMessageBox::StandardButton reply = QMessageBox::question(this, "Генерация
элементов.", "Вы хотите задать минимальное / максимальное значение?",
QMessageBox::Yes|QMessageBox::No);
        if (reply == QMessageBox::Yes)
        {
            int min = QInputDialog::getInt(this, "Enter min.", "Enter min: ");
            int max = QInputDialog::getInt(this, "Enter max.", "Enter max: ");

            if (min > max)
                std::swap(min, max);

            for(int iter = 0; iter < size; iter++)
            {
                treap.insert(rand() % (max - min + 1) + min, &list);
            }
        }
        else
        {
            for(int iter = 0; iter < size; iter++)
            {
                treap.insert(rand(), &list);
            }
        }

        ui->horizontalSlider->setRange(0, list.size() - 1);
        scene->clear();
        list.back().first->draw(scene);
        ui->textBrowser->clear();
        ui->textBrowser->append(list.back().second);

        ui->delete_item->setEnabled(true);
    }
}

void MainWindow::on_NLR_triggered()
{
```

```
   QString filepath = QFileDialog::getSaveFileName(this, "Сохранить в тектовый документ.",
QDir::homePath(), tr("Treap(*.txt)"));

        QFile file(filepath);
        if(!file.open(QFile::WriteOnly | QFile::WriteOnly))
        {
           QMessageBox::warning(this, "Внимание!", "Файл не открыт.");
        }
        else
        {
           treap.save(file, TreapTraversal::NLR);
        }

        file.close();
}

void MainWindow::on_LNR_triggered()
{
   QString filepath = QFileDialog::getSaveFileName(this, "Сохранить в тектовый документ.",
QDir::homePath(), tr("Treap(*.txt)"));

        QFile file(filepath);
        if(!file.open(QFile::WriteOnly | QFile::WriteOnly))
        {
           QMessageBox::warning(this, "Внимание!", "Файл не открыт.");
        }
        else
        {
           treap.save(file, TreapTraversal::LNR);
        }

        file.close();
}

void MainWindow::on_LRN_triggered()
{
   QString filepath = QFileDialog::getSaveFileName(this, "Сохранить в тектовый документ.",
QDir::homePath(), tr("Treap(*.txt)"));

        QFile file(filepath);
        if(!file.open(QFile::WriteOnly | QFile::WriteOnly))
        {
           QMessageBox::warning(this, "Внимание!", "Файл не открыт.");
        }
        else
        {
           treap.save(file, TreapTraversal::LRN);
        }

        file.close();
}
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMessageBox>
#include <QMainWindow>
#include <QInputDialog>
#include <QGraphicsScene>

#include "treap.h"
#include "graphicsviewzoom.h"

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_from_file_triggered();

    void on_from_string_triggered();

    void on_add_item_triggered();

    void on_delete_item_triggered();

    void on_horizontalSlider_sliderMoved(int position);

    void on_generate_triggered();

    void on_NLR_triggered();

    void on_LNR_triggered();

    void on_LRN_triggered();
private:
    Ui::MainWindow* ui;
    QGraphicsScene* scene;

    Treap<int> treap;

    QVector<std::pair<Treap<int>*, QString>> list;

    graphicsviewzoom* zoom;
};
#endif // MAINWINDOW_H
```

Название файла: treap.h

```
#ifndef TREAP_H
#define TREAP_H

#include <cmath>
#include <iostream>
#include <cstdlib>

#include <QAction>
#include <QFileDialog>
#include <QTextBrowser>
#include <QApplication>
#include <QGraphicsScene>
#include <QGraphicsTextItem>

#define NODE_WIDTH 20
#define NODE_HEIGHT 20

enum class TreapTraversal
{
   NLR,
   LNR,
   LRN
};

template <typename T>
class Treap
{
public:
   Treap() {};
   Treap(T key, int priority) : key(key), priority(priority), left(nullptr), right(nullptr) {};

   void insert(T key, QVector<std::pair<Treap<T>*, QString>>* list = nullptr)
   {
      _insert(root, new Treap<T>(key, rand()), list);
   }

   void erase(T key, QVector<std::pair<Treap<T>*, QString>>* list = nullptr)
   {
      _erase(root, key, list);
      list->push_back(std::make_pair(root->copy(), "Получившееся дерево: "));
   }

   void draw(QGraphicsScene* scene)
   {
      this->_drawTree(scene, pow(2, this->maxDepth()) * 10, pow(2, this->maxDepth()) * 10);
   }

   int maxDepth() const
   {
      int lDepth = this->left != nullptr ? this->left->maxDepth() : 0;
```

```
        int rDepth = this->right != nullptr ? this->right->maxDepth() : 0;

        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);

    }

    void save(QFile& file, TreapTraversal traversal)
    {
        _save(root, file, traversal);
    }

    QString getNodeData()
    {
        QString data;
        data.append("Ключ (");
        data.append(QString::fromStdString(std::to_string(this->key)));
        data.append("). Приоритет (");
        data.append(QString::fromStdString(std::to_string(this->priority)));
        data.append(") \n");
        return data;
    }

    Treap<T>* copy() const
    {
        Treap<T>* newTreap = new Treap<T>(this->key, this->priority);

        if (this->left != nullptr)
            newTreap->left = this->left->copy();

        if (this->right != nullptr)
            newTreap->right = this->right->copy();

        return newTreap;
    }

private:
    void split(Treap* t, T& key, Treap*& left, Treap*& right,QVector<std::pair<Treap<T>*,
QString>>* list)
    {
        if (t == nullptr)
        {
            left = right = nullptr;
        }
        else if (t->key > key)
        {

            QString info;
            info.append("Ключ новой ноды (");
            info.append(QString::fromStdString(std::to_string(key)));
            info.append(") <= Ключа текущей ноды! (");
```

```
            info.append(QString::fromStdString(std::to_string(t->key)));
            info.append(")\nИдем в правую ветвь!");
            list->push_back(std::make_pair(t->copy(), info));

            split(t->left, key, left, t->left, list);
            right = t;
        }
        else
        {
            QString info;
            info.append("Ключ новой ноды (");
            info.append(QString::fromStdString(std::to_string(key)));
            info.append(") > Ключа текущей ноды! (");
            info.append(QString::fromStdString(std::to_string(t->key)));
            info.append(")\nИдем в левую ветвь!");
            list->push_back(std::make_pair(t->copy(), info));

            split(t->right, key, t->right, right, list);
            left = t;
        }
    }

    void merge(Treap*& t, Treap* left, Treap* right, QVector<std::pair<Treap<T>*, QString>>*
list)
    {
        if (!left)
        {
            t = right;
            return;
        }
        if (!right)
        {
            t = left;
            return;
        }

        if (left->priority >= right->priority)
        {
            QString info;
            info.append("Приоритет левой ноды (");
            info.append(QString::fromStdString(std::to_string(left->priority)));
            info.append(") > Приоритета правой ноды! (");
            info.append(QString::fromStdString(std::to_string(right->priority)));
            info.append(")\nИдем в правую ветвь!");
            list->push_back(std::make_pair(t->copy(), info));

            merge(left->right, left->right, right, list);
            t = left;
        }
        else
        {
            QString info;
```

```
        info.append("Приоритет левой ноды (");
        info.append(QString::fromStdString(std::to_string(left->priority)));
        info.append(") <= Приоритета правой ноды! (");
        info.append(QString::fromStdString(std::to_string(right->priority)));
        info.append(")\nИдем в левую ветвь!");
        list->push_back(std::make_pair(t->copy(), info));

        merge(right->left, left, right->left, list);
        t = right;
    }
}

void _insert(Treap*& t, Treap* it, QVector<std::pair<Treap<T>*, QString>>* list)
{
    if (t == nullptr)
    {
        t = it;
        list->push_back(std::make_pair(t->copy(), "Добавляем элемент в дерево: "));
        return;
    }

    if (it->priority > t->priority)
    {
        QString info;
        info.append("Приоритет новой ноды ");
        info.append(it->getNodeData());
        info.append("> Приоритета текущей ноды! ");
        info.append(t->getNodeData());
        info.append("\nВызываем метод split для перестраивания дерева!");
        list->push_back(std::make_pair(t->copy(), info));

        split(t, it->key, it->left, it->right, list);
        t = it;

        list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
    }
    else
    {
        if (it->key < t->key)
        {
            QString info;
            info.append("Ключ новой ноды ");
            info.append(it->getNodeData());
            info.append("< Ключа текущей ноды! ");
            info.append(t->getNodeData());
            info.append("\nИдем в левую ветвь.");
            list->push_back(std::make_pair(t->copy(), info));

            _insert(t->left, it, list);

            list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
        }
```

```
        else
        {
            QString info;
            info.append("Ключ новой ноды ");
            info.append(it->getNodeData());
            info.append(">= Ключа текущей ноды! ");
            info.append(t->getNodeData());
            info.append("\nИдем в правую ветвь.");
            list->push_back(std::make_pair(t->copy(), info));

            _insert(t->right, it, list);

            list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
        }
    }
}

void _erase (Treap*& t, T key, QVector<std::pair<Treap<T>*, QString>>* list)
{
    if (t == nullptr)
    {
        //list->back().second.append("\nНода с таким значением не найдена!");
        return;
    }
    if (t->key == key)
    {
        QString info;
        info.append("Ключ (");
        info.append(key);
        info.append(") Найден! Это нода: ");
        info.append(t->getNodeData());
        list->push_back(std::make_pair(t->copy(), info));

        merge(t, t->left, t->right, list);
    }
    else if (t->key > key)
    {
        QString info;
        info.append("Ключ этой ноды ");
        info.append(t->getNodeData());
        info.append("> искомого ключа! (");
        info.append(key);
        info.append(")\nИдем в левую ветвь.");
        list->push_back(std::make_pair(t->copy(), info));

        _erase(t->left, key, list);

        //list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
    }
    else
    {
        QString info;
```

```
            info.append("Ключ этой ноды ");
            info.append(t->getNodeData());
            info.append("< искомого ключа! (");
            info.append(key);
            info.append(")\nИдем в правую ветвь.");
            list->push_back(std::make_pair(t->copy(), info));

            _erase(t->right, key, list);

            //list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
        }

        //list->push_back(std::make_pair(t->copy(), "Получившееся дерево: "));
    }

    void _drawTree(QGraphicsScene* scene, int width, int lineSize, int depth = 0, bool circle = true)
    {
        QPen pen;
        if (this == nullptr)
        {
            return;
        }
        else
        {
            if (circle)
                pen.setBrush(Qt::red);
            else
                pen.setBrush(Qt::green);

            pen.setWidth(4);

            if (this->left != nullptr)
                scene->addLine(width + NODE_WIDTH / 2, depth + NODE_HEIGHT / 2, width -
lineSize / 2 + NODE_WIDTH / 2, depth + 60 + NODE_HEIGHT / 2, pen);

            if (this->right != nullptr)
                scene->addLine(width + NODE_WIDTH / 2, depth + NODE_HEIGHT / 2, width +
lineSize / 2 + NODE_WIDTH / 2, depth + 60 + NODE_HEIGHT / 2, pen);

            scene->addEllipse(width, depth, NODE_WIDTH, NODE_HEIGHT, pen,
QBrush(Qt::gray));

            QString nodeKey, nodePriotity;
            nodeKey = QString::fromStdString(std::to_string(this->key));
            nodePriotity = QString::fromStdString(std::to_string(this->priority));

            QGraphicsTextItem* textKey = new QGraphicsTextItem;
            QGraphicsTextItem* textPriority = new QGraphicsTextItem;

            const QColor myTextColor = QColor(Qt::black);

            textKey->setDefaultTextColor(myTextColor);
```

```
            textKey->setPlainText(nodeKey);
            textKey->setPos(width + nodeKey.size() / 3, depth + 17);
            scene->addItem(textKey);

            textPriority->setDefaultTextColor(myTextColor);
            textPriority->setPlainText(nodePriotity);
            textPriority->setPos(width + nodePriotity.size() / 3, depth + 30);
            scene->addItem(textPriority);

            if (this->left != nullptr)
               this->left->_drawTree(scene, width - lineSize / 2, lineSize / 2, depth + 60, false);

            if (this->right != nullptr)
               this->right->_drawTree(scene, width + lineSize / 2, lineSize / 2, depth + 60, false);
        }
    }

    void _save(Treap*& t, QFile& file, TreapTraversal traversal)
    {
       if (traversal == TreapTraversal::NLR)
          file.write(t->getNodeData().toUtf8());

        if (t->left != nullptr)
          _save(t->left, file, traversal);

        if (traversal == TreapTraversal::LNR)
           file.write(t->getNodeData().toUtf8());

        if (t->right != nullptr)
          _save(t->right, file, traversal);

        if (traversal == TreapTraversal::LRN)
           file.write(t->getNodeData().toUtf8());
    }

private:
    T key;
    int priority;
    Treap* left, * right;
    Treap* root = nullptr;

};

#endif // TREAP_H
```

Название файла: graphicsviewzoom.cpp

```
#include "graphicsviewzoom.h"
#include <QMouseEvent>
#include <QApplication>
#include <QScrollBar>
#include <qmath.h>

graphicsviewzoom::graphicsviewzoom(QGraphicsView* view)
  : QObject(view), _view(view)
{
  _view->viewport()->installEventFilter(this);
  _view->setMouseTracking(true);
  _modifiers = Qt::ControlModifier;
  _zoom_factor_base = 1.0015;
}

void graphicsviewzoom::gentle_zoom(double factor)
{
  _view->scale(factor, factor);
  _view->centerOn(target_scene_pos);
  QPointF delta_viewport_pos = target_viewport_pos - QPointF(_view->viewport()->width() / 2.0,
_view->viewport()->height() / 2.0);
  QPointF viewport_center = _view->mapFromScene(target_scene_pos) - delta_viewport_pos;
  _view->centerOn(_view->mapToScene(viewport_center.toPoint()));
  emit zoomed();
}

void graphicsviewzoom::set_modifiers(Qt::KeyboardModifiers modifiers)
{
  _modifiers = modifiers;

}

void graphicsviewzoom::set_zoom_factor_base(double value)
{
  _zoom_factor_base = value;
}

bool graphicsviewzoom::eventFilter(QObject *object, QEvent *event)
{
  if (event->type() == QEvent::MouseMove)
  {
    QMouseEvent* mouse_event = static_cast<QMouseEvent*>(event);
    QPointF delta = target_viewport_pos - mouse_event->pos();
    if (qAbs(delta.x()) > 5 || qAbs(delta.y()) > 5)
    {
      target_viewport_pos = mouse_event->pos();
      target_scene_pos = _view->mapToScene(mouse_event->pos());
    }
  }
```

```
  else if (event->type() == QEvent::Wheel)
  {
    QWheelEvent* wheel_event = static_cast<QWheelEvent*>(event);
    if (QApplication::keyboardModifiers() == _modifiers)
    {
      if (wheel_event)
      {
        double angle = wheel_event->angleDelta().y();
        double factor = qPow(_zoom_factor_base, angle);
        gentle_zoom(factor);
        return true;
      }
    }
  }
  Q_UNUSED(object)
  return false;
}
```

Название файла: graphicsviewzoom.h

```
#include <QObject>
#include <QGraphicsView>

class graphicsviewzoom : public QObject {
  Q_OBJECT
public:
  graphicsviewzoom(QGraphicsView* view);
  void gentle_zoom(double factor);
  void set_modifiers(Qt::KeyboardModifiers modifiers);
  void set_zoom_factor_base(double value);

private:
  QGraphicsView* _view;
  Qt::KeyboardModifiers _modifiers;
  double _zoom_factor_base;
  QPointF target_scene_pos, target_viewport_pos;
  bool eventFilter(QObject* object, QEvent* event);

signals:
  void zoomed();
};
```