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.

Студент гр. 9384 _____ Прашутинский К.И.

Преподаватель _____ Ефремов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Прашутинский К.И.

Группа 9384

Тема работы: Рандомизированные дерамиды поиска – вставка и исключение.

Исходные данные:
Решение разрабатывалось с графической реализацией под фреймворком “Qt”. Для пользователя предоставлен дружеский интерфейс, в котором он выполнить задания прохождения по дереву и вставку элемента.

Содержание пояснительной записки:
"Введение”, “Описание алгоритма”, “Описание структур данных и функций”, “Описание интерфейса пользователя”, “Заключение”.

Предполагаемый объем пояснительной записки:
Не менее 17 страниц.

Дата выдачи задания: 1.12.2020

Дата сдачи реферата: 31.12.2020

Дата защиты реферата: 31.12.2020

Студент _____ Прашутинский К.И.

Преподаватель _____ Ефремов М.А.

АННОТАЦИЯ

Для разработки решений использовался фреймворк “Qt”. У пользователя есть возможность выбрать вероятность распределения, влияющая на рандомизацию дерева. При написании кода создается структура данных, представляющее собой бинарное дерево, к которой добавляется метод вставки в корень. Этот метод позволяет с некоторой вероятностью повернуть влево или вправо дерево относительно данного узла. Пользователю будут даны задания для получения или закрепления знаний по проходу и вставки в дерамиду поиска.

SUMMARY

For the development of solutions, the "Qt" framework was used. The user has the ability to choose the probability of a distribution that affects the randomization of the tree. When you write the code, a data structure is created, which is a binary tree, to which an insert method is added to the root. This method allows, with some probability, to rotate the tree to the left or to the right relative to a given node. The user will be given the task to obtain or consolidate the knowledge of the aisle treap and insert into it elements.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: сделать задания для подкрепления и приобретению знаний по проходу рандомизированной дерамиды поиска и вставке в него нового элемента.

Для подкрепления и приобретению знаний есть два задания:

1. Записать обход дерева по КЛП, ЛКП, ЛПК.
2. Записать обход дерева по КЛП, ЛКП, ЛПК при условии, что вставлен элемент в дерево.

Задания проверяются автоматически и выводятся на последнем окне.

1. ОПИСАНИЕ АЛГОРИТМА

1.1. Создается класс treap.

Создается класс treap. В котором хранятся указатели на правое и левое поддеревья и ключ, а также методы для вставки, удаления и обхода дерева. Необходимо хранить информацию о классе treap для дальнейшей визуализации, поэтому создается класс mainwindow, в котором идет отрисовка сцен, алгоритм зума, алгоритм прохода по программе и генерации заданий.

1.2. Окно mainwindow.

Окно mainwindow содержит виджет QStackedWidget в котором содержатся страницы приложения: начальная страница, первое задание, второе задание, результаты. На начальной странице, первом задании, втором задании содержатся кнопки перехода на следующую страницу. На странице результата есть кнопка выхода.

2. ОПИСАНИЕ СТРУКТУРЫ ДАННЫХ И ФУНКЦИЙ

2.1. Класс treap

Конструктор создает новый элемент с заданными параметрами.

Функция insert представляет собой вставку очередного элемента.

Функция Create2task вставляет рандомный ключ с приоритетом 0.

Функция maxDepth вычисляет глубину.

Функция draw рисует дерево.

Функция KLR формирует строку обхода по дереву КЛП.

Функция LKR формирует строку обхода по дереву ЛКП.

Функция LRK формирует строку обхода по дереву ЛПК.

Функция split разделяет поддеревья.

Функция merge объединяет поддеревья.

2.2. Класс mainwindow

Конструктор создает окно, устанавливает зум для графических виджетов, устанавливает место отрисовки деревьев, устанавливает место начала программы.

Функция on_actionCreate_variant_triggered создает дерево с введенным пользователем числом элементов. Активируется на кнопку Create variant.

Функция on_Open_triggered создает дерево по данным выбранного файла. Активируется на кнопку Open.

Функция on_Enter_triggered создает дерево по введенным данным. Активируется на кнопку Enter.

Функция on_actionEnter_Key_triggered добавляет один элемент в дерево.

Функция on_actionErase_Key_triggered удаляет один элемент в дерево.

Функция on_FINISH_THE_TASK_clicked считывает ответы веденные пользователи, вычисляет результат, пишет результат, создает следующее задание и переключает на следующую страницу.

Функция on_FINISH_THE_TASK_3_clicked считывает ответы веденные пользователи, вычисляет результат, пишет результат и переключает на следующую страницу.

Функция on_pushButton_clicked создает следующее задание и переключает на следующую страницу.

Функция on_pushButton_2_clicked закрывает приложение.

Функция on_pushButton_3_clicked переходит на первую страницу, затирает прошлую попытку и дает возможность повторить попытку.

Функция create2task создает 2-е задание.

2.3. Класс graphics_view_zoom

Класс отвечающий за зум графика которые увеличивает удобство пользования с большими деревьями.

3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

3.1. Переход по страницам.

Переход по страницам происходит по кнопкам.

3.2. Пояснительная записка.

При запуске программы появляется пояснительная записка, которая поясняет правила выполнения заданий и объяснений как делаются задания.

3.3. Задания.

Для двух заданий даны по 3 подзадания. Ответы нужно вносить под условие названия.

ЗАКЛЮЧЕНИЕ

При подведении итогов следует акцентировать внимание на удобство и понятность интерфейса программы. Настраиваемый пользователем уровень сложности увеличивает полезность программы т.к. этим приложением можно пользоваться и для получения начального понимания обхода дерева и вставки в него символа, так и для закрепления знаний и улучшения навыков.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация // https://doc.qt.io/ URL: https://doc.qt.io/qt5/q3dsurface.html (дата обращения: 27.12.2020).
2. Форум StackOverflow https://stackoverflow.com/
3. Документация c++ // https://www.cplusplus.com/doc/tutorial/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
          srand(time(0));
          QApplication a(argc, argv);
          MainWindow w;
          w.show();
          return a.exec();
}
```

Название файла: mainwindow.cpp

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

#include <QMessageBox>

#define KEY_MAX 1000

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    z = new Graphics_view_zoom(ui->graphicsView);
    z->set_modifiers(Qt::NoModifier);
    z_2 = new Graphics_view_zoom(ui->graphicsView_3);
    z_2->set_modifiers(Qt::NoModifier);
    scene = new QGraphicsScene();
    ui->graphicsView->setScene(scene);
    scene_2 = new QGraphicsScene();
    ui->graphicsView_3->setScene(scene_2);
    ui->stackedWidget->setCurrentIndex(0);
}

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

```
    }

    void MainWindow::on_actionCreate_variant_triggered(){
        int data = QInputDialog::getInt(this, "Create variant.", "Enter
amount of elements: ");
        int random;
        for (int i = 0; i < KEY_MAX; i++) {
            treap.erase(i);
        }
        for(int i = 0; i < data; i++){
            random = rand() % KEY_MAX;
            treap.insert(random);
        }
        scene->clear();
        treap.draw(scene);
    }

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

        QFile file(filepath);
        if(!file.open(QFile::ReadOnly | QFile::ReadOnly))
        {
            QMessageBox::warning(this, "Warning!", "File not open.");
        }
        else
        {
            QTextStream in(&file);
            QString str = in.readLine();

            QStringList lst = str.split(" ");

            QVector<int> data;

            for (qsizetype index = 0; index < lst.size(); index++)
            {
                QString num = lst[index];
                data.push_back(num.toInt());
            }

            for(QVector<int>::iterator  iter  =  data.begin();iter  !=
data.end();iter++)
```

```
            {
                    treap.insert(*iter);
            }

            scene->clear();
            treap.draw(scene);
        }

        file.close();
    }

    void MainWindow::on_Enter_triggered()
    {
        QString str = QInputDialog::getText(this, "Enter Traversal.",
"Enter Preorder Traversal: ");

        QStringList lst = str.split(" ");

        QVector<int> data;

        for (qsizetype index = 0; index < lst.size(); index++)
        {
            QString num = lst[index];
            data.push_back(num.toInt());
        }

        for(QVector<int>::iterator iter = data.begin();iter !=
data.end();iter++)
        {
                treap.insert(*iter);
        }

        scene->clear();
        treap.draw(scene);
    }

    void MainWindow::on_actionEnter_Key_triggered()
    {
        int data = QInputDialog::getInt(this, "Enter Key.", "Enter Key:
");
        treap.insert(data);
        scene->clear();
        treap.draw(scene);
    }
```

```
    void MainWindow::on_actionErase_Key_triggered()
    {
        int data = QInputDialog::getInt(this, "Erase Key.", "Enter Key:
");
        treap.erase(data);
        scene->clear();
        treap.draw(scene);
    }

    void MainWindow::on_FINISH_THE_TASK_clicked()
    {
        QMessageBox::StandardButton reply = QMessageBox::question(this,
"The confirmation.", "Are you sure you want to finish trying?",
QMessageBox::Yes|QMessageBox::No);
        if(reply == QMessageBox::Yes){
            ui->stackedWidget->setCurrentIndex(2);
            QString str;
            unsigned count = 0;
            double percent = 100;
            treap.KLR(str);
            str.chop(1);
            if(MainWindow::ui->lineEditKLR->text() == str){
                count++;
            }
            MainWindow::ui->AllResults->append("KLR\n");
            MainWindow::ui->AllResults->append("You     ansver:\t"     +
MainWindow::ui->lineEditKLR->text() + "\n");
            MainWindow::ui->AllResults->append("Right ansver:\t" + str
+ "\n");
            str.clear();
            treap.LKR(str);
            str.chop(1);
            if(MainWindow::ui->lineEditLKR->text() == str){
                count++;
            }
            MainWindow::ui->AllResults->append("LKR\n");
            MainWindow::ui->AllResults->append("You     ansver:\t"     +
MainWindow::ui->lineEditLKR->text() + "\n");
            MainWindow::ui->AllResults->append("Right ansver:\t" + str
+ "\n");
            str.clear();
            treap.LRK(str);
            str.chop(1);
```

```
            if(MainWindow::ui->lineEditLRK->text() == str){
                count++;
            }
            MainWindow::ui->AllResults->append("LRK\n");
            MainWindow::ui->AllResults->append("You     ansver:\t"     +
MainWindow::ui->lineEditLRK->text() + "\n");
            MainWindow::ui->AllResults->append("Right  ansver:\t"  +  str
+ "\n");
            str.clear();
            str.push_back("Your results: ");
            str.push_back(QString::number(count));
            str.push_back("/3 (");
            str.push_back(QString::number(count*percent/3));
            str.push_back("%)\n");
            MainWindow::ui->AllResults->append(str);


            create2task();
        }
    }

    void MainWindow::on_pushButton_clicked()
    {
        QMessageBox::StandardButton GoTo1 = QMessageBox::question(this,
"Test.",     "Do     you     want     to     start     the     test?",
QMessageBox::Yes|QMessageBox::No);
        if(GoTo1 == QMessageBox::Yes){
            ui->stackedWidget->setCurrentIndex(1);
            on_actionCreate_variant_triggered();
        }
    }

    void MainWindow::on_FINISH_THE_TASK_3_clicked()
    {
        QMessageBox::StandardButton reply = QMessageBox::question(this,
"The  confirmation.",  "Are  you  sure  you  want  to  finish  trying?",
QMessageBox::Yes|QMessageBox::No);
        if(reply == QMessageBox::Yes){
            ui->stackedWidget->setCurrentIndex(3);
            QString str;
            unsigned count = 0;
            double percent = 100;
            treap.KLR(str);
            str.chop(1);
```

```
            if(MainWindow::ui->lineEditKLR_4->text() == str){
                count++;
            }
            MainWindow::ui->AllResults_2->append("KLR\n");
            MainWindow::ui->AllResults_2->append("You    ansver:\t"    +
MainWindow::ui->lineEditKLR_4->text() + "\n");
            MainWindow::ui->AllResults_2->append("Right   ansver:\t"   +
str + "\n");
            str.clear();
            treap.LKR(str);
            str.chop(1);
            if(MainWindow::ui->lineEditLKR_4->text() == str){
                count++;
            }
            MainWindow::ui->AllResults_2->append("LKR\n");
            MainWindow::ui->AllResults_2->append("You    ansver:\t"    +
MainWindow::ui->lineEditLKR_4->text() + "\n");
            MainWindow::ui->AllResults_2->append("Right   ansver:\t"   +
str + "\n");
            str.clear();
            treap.LRK(str);
            str.chop(1);
            if(MainWindow::ui->lineEditLRK_4->text() == str){
                count++;
            }
            MainWindow::ui->AllResults_2->append("LRK\n");
            MainWindow::ui->AllResults_2->append("You    ansver:\t"    +
MainWindow::ui->lineEditLRK_4->text() + "\n");
            MainWindow::ui->AllResults_2->append("Right   ansver:\t"   +
str + "\n");
            str.clear();
            str.push_back("Your results: ");
            str.push_back(QString::number(count));
            str.push_back("/3 (");
            str.push_back(QString::number(count*percent/3));
            str.push_back("%)\n");
            MainWindow::ui->AllResults_2->append(str);
        }
    }
    void MainWindow::on_pushButton_2_clicked()
    {
        close();
    }
```

```
void MainWindow::create2task(){
    int random;
    unsigned data = QInputDialog::getInt(this, "Create variant.",
"Enter amount of elements: ");
    for (int i = 0; i < KEY_MAX; i++) {
        treap.erase(i);
    }
    for(unsigned i = 0; i < data; i++){
        random = rand() % KEY_MAX;
        treap.insert(random);
    }

    scene->clear();
    treap.draw(scene_2);

    int keyreturn = treap.Create2task();
    MainWindow::ui->FinishResults_2->setText("Key = " +
QString::number(keyreturn) + "\tpriority = 0");
}

void MainWindow::on_pushButton_3_clicked()
{
    MainWindow::ui->FinishResults_2->clear();
    MainWindow::ui->FinishResults->clear();
    MainWindow::ui->FinishResults_2->clear();
    MainWindow::ui->lineEditLRK->clear();
    MainWindow::ui->lineEditLKR->clear();
    MainWindow::ui->lineEditKLR->clear();
    MainWindow::ui->lineEditLRK_4->clear();
    MainWindow::ui->lineEditLKR_4->clear();
    MainWindow::ui->lineEditKLR_4->clear();
    MainWindow::scene->clear();
    MainWindow::scene_2->clear();
    ui->stackedWidget->setCurrentIndex(0);
}
```

Название файла: graphics_view_zoom.cpp

```
#include "graphics_view_zoom.h"

#include <QMouseEvent>
#include <QApplication>
#include <QScrollBar>
#include <QWheelEvent>
#include <qmath.h>
```

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

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

}

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

bool Graphics_view_zoom::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->orientation() == Qt::Vertical) {
                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: treap.h

```
#ifndef TREAP_H
#define TREAP_H

#include <iostream>
#include <cstdlib>

#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <math.h>

#define NODE_WIDTH 40
#define NODE_HEIGHT 40

#define KEY_MAX 1000

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

    void insert(T key)
    {
        _insert(root, new Treap<T>(key, rand()));
    }

    void erase(T key)
    {
        _erase(root, key);
```

```
        }

        void draw(QGraphicsScene* scene)
        {
            _draw(scene, root, pow(2, maxDepth(root)) * 10, pow(2,
maxDepth(root)) * 10);
        }

        int Create2task(){
            int r = rand() % KEY_MAX;
            _insert(root,new Treap<T>(r, 0));
            return r;
        }

        int maxDepth(Treap*& t)
        {
            if (t == nullptr)
                return 0;
            else
            {
                /* compute the depth of each subtree */
                int lDepth = maxDepth(t->left);
                int rDepth = maxDepth(t->right);

                /* use the larger one */
                if (lDepth > rDepth)
                    return(lDepth + 1);
                else return(rDepth + 1);
            }
        }

        void KLR(QString& str){
            if(this->root)
                this->root->_KLR(str);
        }

        void LKR(QString& str){
            if(this->root)
                this->root->_LKR(str);
        }

        void LRK(QString& str){
            if(this->root)
                this->root->_LRK(str);
```

```
    }

    void _KLR(QString& str){
        str.push_back(QString::number(key) + ' ');
        if(this->left)
            this->left->_KLR(str);
        if(this->right)
            this->right->_KLR(str);
    }


    void _LKR(QString& str){
        if(this->left)
            this->left->_LKR(str);
        str.push_back(QString::number(key) + ' ');
        if(this->right)
            this->right->_LKR(str);
    }


    void _LRK(QString& str){
        if(this->left)
            this->left->_LRK(str);
        if(this->right)
            this->right->_LRK(str);
        str.push_back(QString::number(key) + ' ');
    }

    /*bool TestCheck(QString string){
        QString CorrectAnswer;
        KLR(CorrectAnswer);

    }*/
private:
    void split(Treap* t, T& key, Treap*& left, Treap*& right)
    {
        if (t == nullptr)
        {
            left = right = nullptr;
        }
        else if (key < t->key)
        {
            split(t->left, key, left, t->left);
            right = t;
```

```
    }
    else
    {
        split(t->right, key, t->right, right);
        left = t;
    }
}

void merge(Treap*& t, Treap* left, Treap* right)
{
    if (!left || !right)
        t = left ? left : right;
    else if (left->priority >= right->priority)
    {
        merge(left->right, left->right, right);
        t = left;
    }
    else
    {
        merge(right->left, left, right->left);
        t = right;
    }
}

void _insert(Treap*& t, Treap* it)
{
    if (t == nullptr)
    {
        t = it;
        return;
    }

    if (it->priority > t->priority)
    {
        split(t, it->key, it->left, it->right);
        t = it;
    }
    else
    {
        _insert(it->key < t->key ? t->left : t->right, it);
    }
}

void _erase (Treap*& t, T key)
```

```
        {
            if (t == nullptr)
                return;
            if (t->key == key)
                this->merge(t, t->left, t->right);
            else if (t->key > key)
                this->_erase(t->left, key);
            else
                this->_erase(t->right, key);
        }

        void _draw(QGraphicsScene* scene, Treap*& t, int width, int
lineSize, int depth = 0)
        {
            QPen pen;
            if (t == nullptr)
            {
                return;
            }
            else
            {
                pen.setBrush(Qt::black);
                pen.setWidth(4);

                if (t->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 (t->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(t-
>key));
                nodePriotity = QString::fromStdString(std::to_string(t-
>priority));

                QGraphicsTextItem* textKey = new QGraphicsTextItem;
```

```
                QGraphicsTextItem*      textPriority      =      new
QGraphicsTextItem;

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

                // TODO: Выравнивание текста.
                textKey->setDefaultTextColor(myTextColor);
                textKey->setPlainText(nodeKey);
                textKey->setPos(width + nodeKey.size() / 2, depth);
                scene->addItem(textKey);

                // TODO: Выравнивание текста.
                textPriority->setDefaultTextColor(myTextColor);
                textPriority->setPlainText(nodePriotity);
                textPriority->setPos(width  +  nodePriotity.size()  /  2,
depth + 10);
                scene->addItem(textPriority);

                _draw(scene, t->left, width - lineSize / 2, lineSize /
2, depth + 60);
                _draw(scene, t->right, width + lineSize / 2, lineSize /
2, depth + 60);
            }

        }

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

    };

#endif // TREAP_H
```

Название файла: mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QVector>
#include <QMainWindow>
#include <QMessageBox>
#include <QFileDialog>
```

```
#include <QInputDialog>
#include <QTextStream>
#include <QWizard>
#include "graphics_view_zoom.h"

#include "treap.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_actionCreate_variant_triggered();

    void on_Open_triggered();

    void on_Enter_triggered();

    void on_actionEnter_Key_triggered();

    void on_actionErase_Key_triggered();

    void on_FINISH_THE_TASK_clicked();

    void on_pushButton_clicked();

    void on_FINISH_THE_TASK_3_clicked();

    void on_pushButton_2_clicked();

    void create2task();

    void on_pushButton_3_clicked();

private:
```

```
        Treap<int> treap;

        Ui::MainWindow *ui;
        QGraphicsScene* scene;
        QGraphicsScene* scene_2;
        Graphics_view_zoom* z;
        Graphics_view_zoom* z_2;
    };
#endif // MAINWINDOW_H
```

Название файла: graphics_view_zoom.h

```
#ifndef GRAPHICS_VIEW_ZOOM_H
#define GRAPHICS_VIEW_ZOOM_H


#include <QObject>
#include <QGraphicsView>

class Graphics_view_zoom : public QObject {
      Q_OBJECT
public:
      Graphics_view_zoom(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();
};

#endif // GRAPHICS_VIEW_ZOOM_H
```