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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: БДП

Студент гр. 8381 _____ Лисок М.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2019

Цель работы.

Изучить основные характеристики и реализовать структуру данных БДП (англ. *treap*). Создать программу, выполняющую визуализацию заданного БДП и выполняющую с ним заданную операцию расщепления.

Постановка задачи.

- по заданному файлу, все элементы которого различны построить БДП;
- выполнить операцию расщепления по заданной структуре данных БДП;

Основные теоретические положения.

Декартово дерево или дерамида (англ. *Treap*) — это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу (отсюда и второе её название: *treap* (*tree* + *heap*) и дерамида (дерево + пирамида), также существует название курево (куча + дерево).

Более строго, это бинарное дерево, в узлах которого хранятся пары (x, y), где *x* — это ключ, а *y* приоритет. Также оно является двоичным деревом поиска по *x* и пирамидой по *y*. Предполагая, что все *x* и *y* являются различными, получаем, что если некоторый элемент дерева содержит (x_0, y_0), то у всех элементов в левом поддереве $x < x_0$, у всех элементов в правом поддереве $x > x_0$, а также и в левом, и в правом поддереве имеем: $y < y_0$.

Выполнение работы.

Написание работы производилось на базе операционной системы Mac OS в среде разработки QtCreator с использованием фреймворка Qt. Сборка, отладка производились в QtCreator при помощи отладчика gdb 8.3.1 и компилятора g++.

В классе treapNode описана структура рандомизированного бинарного дерева и непосредственные методы работы с ним, описание которых представлено в табл. 1.

Таблица 1 – методы, описанные в заголовочном файле treapNode.h

TreapNode* merge(TreapNode*, TreapNode*);	С помощью этого метода можно слить два декартовых дерева в одно. Причём, все ключи в первом(*левом*) дереве должны быть меньше, чем ключи во втором(*правом*).
void split(ul key, TreapNode*& T1, TreapNode*& T2);	Метод, разрезающий исходное дерево по заданному ключу *key*. Возвращать она будет такую пару деревьев *T1* и *T2*, что в *T1* ключи меньше key, а *T2* все остальные.
TreapNode* add(ul);	Метод, добавляющий элемент в не пустое декартово дерево, используя ключ, приоритет в данном случае устанавливается рандомно
TreapNode* add(ul, ul);	Метод, добавляющий элемент в не пустое декартово дерево, используя ключ и приоритет
TreapNode* remove(ul);	Метод, удаляющий элемент по ключу
int exist(ul);	Метод, возращает уровень, на котором располагается элемент или -1 если такого элемента нет
void recalcSizeOf();	Метод, переустанавливающий размер декартово дерева
size_t height();	Метод, определяющий высоту декартова дерева
static void clear(TreapNode*&);	Метод, очищающий декартово дерево
static size_t sizeOf(TreapNode*);	Метод, возращающий колличество элементов в дереве.

Продолжение таблицы 1

size_t levelLenght(size_t);	Метод, возращающий число элементов на определенном уровне

Файл drawing.cpp содержит функции, выполняющие работу по отрисовке графического представления на QGraphicsScene через данные, получаемые от функциональных алгоритмов. Описание представлено в табл. 2.

Таблица 2 - Функции, связующие графический интерфейс и алгоритмы

void MainWindow::graphic(TreapNode<int>* node, bool isRoot);	По заданному бинарному дереву выполняет рисование в объекте QGraphicsScene
void MainWindow::treePainter(TreapNode<int> *node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth);	Рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте QGraphicsScene

Файл mainwindow.cpp содержит основные слоты, отвечающие за выполнение построения дерева, пошаговый режим, расщепление дерева, ввод данных из файла – описание представлено в табл. 3.

Таблица 3 – Методы работы с выражением и слоты взаимодействия с UI

void MainWindow::createTreap()	Вспомогательный метод, отвечающий за построение дерева на основании введенных данных

Продолжение таблицы 3

void MainWindow::onTreapSplitingButtonClicked()	Слот, отвечающий за расщепления дерева
void MainWindow::onFileOpenButtonClicked()	Слот, отвечающий за считывание из файла
void MainWindow::onTreapDrawingButtonClicked();	Слот, отвечающий за отображение дерева
void MainWindow::onStepTreapDrawingButtonClicked();	Слот, отвечающий за отображение дерева в пошаговом режиме

Для реализации пошагового режима в данной лабораторной работе использовался метод задержки выполнения основного потока программы с возможностью взаимодействия с пользовательским интерфейсом. Данные метода из-за применения только для отрисовки бинарного дерева описаны в заголовочном файле drawing.cpp.

Представленный ниже код позволяет “поставить на паузу” выполнение текущей операции и ожидать от пользователя нажатия кнопки следующего шага (смена состояние происходит при помощи изменения флага состояния):

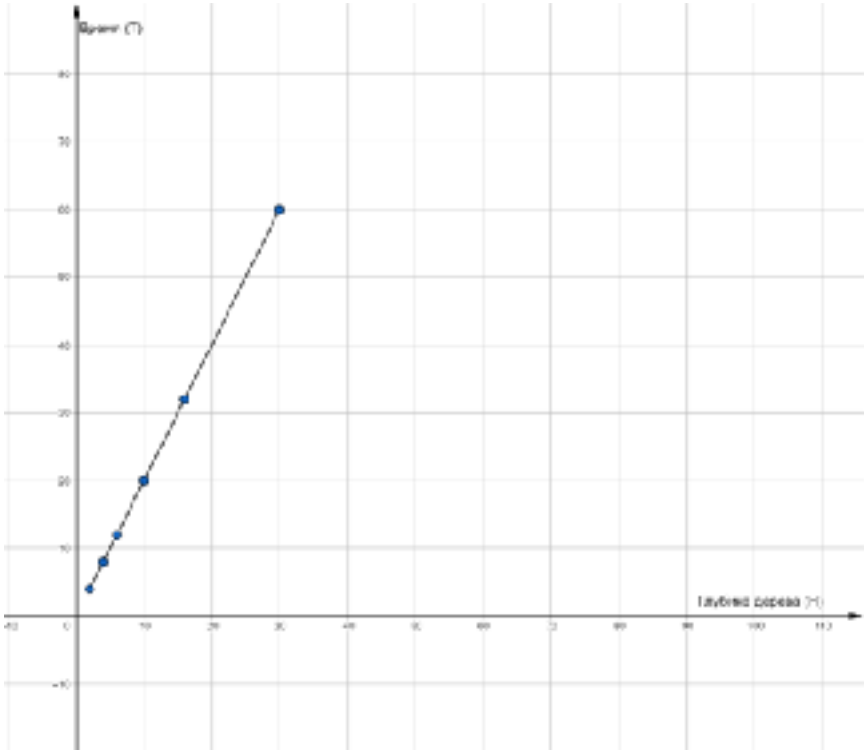
```
    void loopLatency() {
         qDebug() << "Loop latency for step-by-step started" << endl;
        for( ; ; ) {
            QApplication::processEvents();
            if(stepLoopSwitcher == false) break;
        }
        stepLoopSwitcher = true;
    }
```

Проверка начального входа в петлю осуществляется за счёт флага stepButStepEnable, переключаемого благодаря выбору соответствующей опции в UI.

Оценка сложности алгоритма.

Оценим время работы операции `split`. Во время выполнения вызывается одна операция `split` для дерева хотя бы на один меньшей высоты и делается ещё $O(1)$ операций. Тогда итоговая трудоёмкость этой операции равна $O(h)$, где h — высота дерева. Данные рассуждения были подтверждены практически, путём построения графика зависимости времени работы метода класса `Treep` от высоты задаваемого дерева. Результаты приведены на рис. 1.

Рисунок 1 – График зависимости времени от глубины дерева



Все остальные методы работы с пирамидой реализованы через вставку узлов в пирамиду. Вставка узлов состоит из двух методов `merge` и одного `split`. Как было доказано ранее, `merge`, как и `split`, теоретически требуют $O(n)$ операций, где n – высота дерева, так как в них рекурсивно вызывается $O(1)$ операций для каждого дерева меньшей высоты. Операция слияния двух пирамид (ключи в которых не сортированы) требует рекурсивного обхода второй пирамиды, что, теоретически требует $O(k)$ операций, где k – количество элементов во второй пирамиде.

Сложность $O(h)$ может не быть обеспечена в случае вырождения дерева в линейный список при построении. Решение данной проблемы обеспечивается

использованием псевдослучайных чисел при назначении приоритета очередного добавляемого узла, что даёт математическое ожидание (средняя высота) $O(log\ h)$.

Тестирование.

Визуализация операции расщепления дерева по заданному выражению представлена на рис. 2, 3, 4

Рисунок 2 - визуализация дерева по выражению 14 15 18 19 и расщепление по ключу 21

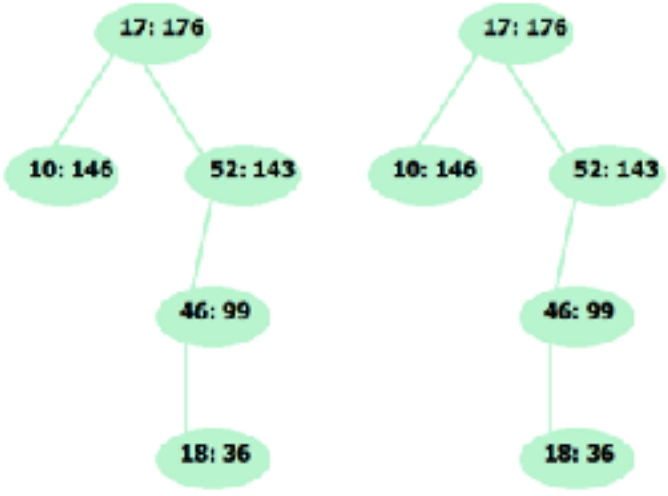


Рисунок 3 - визуализация дерева по выражению 10 17 18 46 52 и расщепление по ключу 5

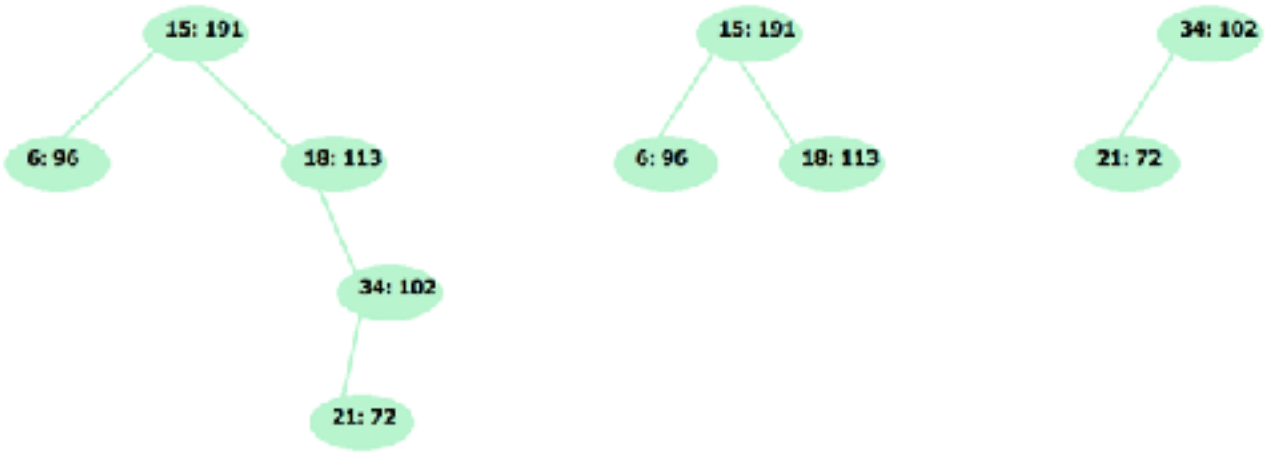


Рисунок 4 - визуализация дерева по выражению 6 18 21 34 15 и расщепление по ключу 18

Вывод.

В ходе выполнения данной работы была изучена и реализована такая структура данных, как декартово дерево. На её основе была составлена программа для построения, расщепления и визуализации декартово дерева.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: headers.h

```
#ifndef HEADERS_H
#define HEADERS_H

#include <vector>
#include <iostream>
#include <string>
#include <fstream>
#include <math.h>

#include <QPushButton>
#include <QVBoxLayout>
#include <QGraphicsView>
#include <QGraphicsItem>
#include <QTextEdit>
#include <QDebug>
#include <QFileDialog>
#include <QDir>
#include <QCheckBox>
#include <QApplication>
#include <QLineEdit>
#include <QRegExpValidator>
#include <QLabel>
#include <QPoint>

using namespace std;

typedef unsigned long ul;

#define RAND_FACTOR 200

#endif // HEADERS_H
```

Название файла: mainnwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "headers.h"
```

```
#include "treapNode.h"


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow {

    Q_OBJECT

    public:
        MainWindow(QWidget *parent = nullptr);
        ~MainWindow();
        void graphic(TreapNode<ul>*, bool) ;
    public slots:
        void onFileOpenButtonClicked();
        void onTreapDrawingButtonClicked();
        void onStepTreapDrawingButtonClicked();
        void onTreapSplitingButtonClicked();

    private:
        Ui::MainWindow *ui;

         void treePainter(TreapNode<ul> *, int, int, int, int,
QPen &, QBrush &, QFont &, int);
        void setUpUI();
        void createTreap();
        QPushButton *fileOpen;
        QPushButton *drawingTreap;
        QPushButton *stepDrawingTreap;
        QPushButton *splitTreap;
        QCheckBox *stepSwitch;

        QGraphicsView *mainGraphicsView;
        QGraphicsScene *mainGraphicsScene;
        QPen pen;
        QColor color;
        QBrush brush;
        QFont font;

        QLineEdit *inputLine;
        QLineEdit *splitKeyInput;
```

```
        QLineEdit *slpitError;
        QLabel *inputExprLabel;
        QLabel *splitElLabel;
        TreapNode<ul>* workingTreap;

        bool isDrawing{false};

};

#endif // MAINWINDOW_H
```

Название файла: treapNode.h

```
#ifndef TREAPNODE_H
#define TREAPNODE_H

#include "headers.h"


template<class T>
class TreapNode{
public:
    friend class MainWindow;
    TreapNode();
      TreapNode(ul, ul priority = std::rand() % RAND_FACTOR +
1, TreapNode* left = nullptr, TreapNode* right = nullptr);

     TreapNode* merge(TreapNode*, TreapNode*);
    void split(ul, TreapNode*&, TreapNode*&);
     TreapNode* add(ul);
    TreapNode* add(ul, ul);
    TreapNode* remove(ul);
     int exist(ul);
    void recalcSizeOf();
    size_t height();
    static void clear(TreapNode*&);
    static size_t sizeOf(TreapNode*);

     ul key(size_t);
     size_t levelLenght(size_t);

     const int DrawingSize{28};
     const int MajorAxis{55};
```

```
    void setStartPos(int, int);
    QPoint getStartPos() const;

    // STEP-BY-STEP

    static void loopLatency();
    static void changeLoopState();
    static void changeStepByStepMode();
    static void setDisableStepMode();
    static bool stepButStepEnable;

    ul getKey() const{
        return mKey;
    }
    ul getPriority() const{
        return mPriority;
    }

    void setWidth(int w){
        width = w;
    }

    int getWidth() const{
        return width;
    }
    void setForErrors(string & str){
       forErrors += str;
    }

    const string& getForErrors() const{
        return forErrors;
    }

private:

    TreapNode* mLeft;
    TreapNode* mRight;

    ul mKey;
    ul mPriority;

    size_t mSizeOf;
    static bool stepLoopSwitcher;
```

```
        // DRAWING
        QPoint startPos{0, 0};
        int width;

        static string forErrors;
    };

    template<class T>

    TreapNode<T>::TreapNode() {}

    template<class T>
    TreapNode<T>::TreapNode(ul key, ul priority, TreapNode* left,
TreapNode* right) : TreapNode() {
        mLeft = left;
        mRight = right;
        mKey = key;
        mPriority = priority;
    }

    template<class T>
    TreapNode<T>* TreapNode<T>::merge(TreapNode* left, TreapNode*
right){
        if(!left) return right;
        if(!right) return left;

        TreapNode<T>* treap;
        if(left->mPriority > right->mPriority) {
                TreapNode<T>* new_subTreap = merge(left->mRight,
right);
              treap = new TreapNode(left->mKey, left->mPriority,
left->mLeft, new_subTreap);
        } else {
                 TreapNode<T>* new_subTreap = merge(left, right-
>mLeft);
             treap = new TreapNode(right->mKey, right->mPriority,
new_subTreap, right->mRight);
        }

        treap->recalcSizeOf();
        return treap;
```

```
    }

    template<class T>
    void TreapNode<T>::split(ul key, TreapNode*& left,
TreapNode*& right){
        TreapNode<T>* newTreap = nullptr;
        if(mKey <= key) {
            if(!mRight)
                right = nullptr;
            else
                mRight->split(key, newTreap, right);
                    left = new TreapNode(mKey, mPriority, mLeft,
newTreap);
            left->recalcSizeOf();
        }
        else {
            if(!mLeft)
                left = nullptr;
            else
                mLeft->split(key, left, newTreap);
                 right = new TreapNode(mKey, mPriority, newTreap,
mRight);
            right->recalcSizeOf();
        }
    }

    template<class T>
    TreapNode<T>* TreapNode<T>::add(ul key){
        TreapNode<T>* left = nullptr;
        TreapNode<T>* right= nullptr;
        split(key, left, right);
        TreapNode<T>* newNode = new TreapNode(key);
        return merge(merge(left, newNode), right);
    }

    template<class T>
    TreapNode<T>* TreapNode<T>::add(ul key,  ul priority){
        if(exist(key) != -1) {
             string buf = "WARNING:Node with that _key is already
existed in Treap.";
            this->setForErrors(buf);
        }else{
        TreapNode<T>* left = nullptr;
```

```
    TreapNode<T>* rigth = nullptr;
    split(key, left, rigth);
    TreapNode<T>* newNode = new TreapNode(key, priority);
    return merge(merge(left, newNode), rigth);
    }
}

template<class T>
TreapNode<T>* TreapNode<T>::remove(ul key){
    TreapNode<T>* r = nullptr;
    TreapNode<T>* l = nullptr;
    TreapNode<T>* el = nullptr;
    split(key, l, r);
    l->split(key-1, l, el);
    delete el;
    return merge(l, r);
}

template<class T>
void TreapNode<T>::clear(TreapNode*& treap){
    if(treap)
    {
        if(treap->mLeft)
            clear(treap->mLeft);
        if(treap->mRight)
            clear(treap->mRight);

        delete treap;
        treap = nullptr;
    }
}


template<class T>
size_t TreapNode<T>::sizeOf(TreapNode* treap){
    return !treap ? 0 : treap->mSizeOf;
}

template<class T>
void TreapNode<T>::recalcSizeOf()
{
    mSizeOf += sizeOf(mLeft) + sizeOf(mRight);
}
```

```
template<class T>
int TreapNode<T>::exist(ul key){
    int index = 0;
    TreapNode<T>* node = this;

    while (node)
    {
        if(node->mKey == key) return index;
        else if(node->mKey < key)
            node = node->mRight;
        else if(node->mKey > key)
            node = node->mLeft;
        index++;
    }
    return -1;
}

template<class T>
size_t TreapNode<T>::height(){
    size_t index = 0;
    TreapNode<T>* node = this;

    while(TreapNode<T>::sizeOf(node) > 1)
    {
                    if(TreapNode<T>::sizeOf(node->mLeft)  >=
TreapNode<T>::sizeOf(node->mRight))
            node = node->mLeft;
        else
            node = node->mRight;

        index++;
    }
    return index;
}

template<class T>
ul TreapNode<T>::key(size_t idx){
    TreapNode<T>* node = this;
    while(node) {
        if(node->sizeOf(node->mLeft) == idx)
            return node->mKey;
```

```
        else if(node->sizeOf(node->mLeft) > idx)
            node = node->mLeft;
        else if (node->sizeOf(node->mLeft) < idx) {
            idx -= sizeOf(node->mLeft) + 1;
            node = node->mRight;
        }
    }
    return 0;
}

template<class T>
size_t TreapNode<T>::levelLenght(size_t level){
    size_t count = 0;
    for(size_t it = 0; it < mSizeOf; it++)
    {
        size_t cur_height = exist(key(it));
        if(cur_height == level)
            count++;
    }
    return count;
}


template<class T>
void TreapNode<T>::setStartPos(int x, int y) {
    startPos.setX(x);
    startPos.setY(y);
}

template<class T>
QPoint TreapNode<T>::getStartPos() const {
    return startPos;
}

template<class T> bool TreapNode<T> ::stepLoopSwitcher;
template<class T> bool TreapNode<T> ::stepButStepEnable;
template<class T> string TreapNode<T> ::forErrors;

template<class T>
void TreapNode<T>::changeLoopState(){
    stepLoopSwitcher = false;
}
```

```
template<class T>
void TreapNode<T>::changeStepByStepMode(){
    stepButStepEnable = true;
}

template<class T>
void TreapNode<T>::setDisableStepMode(){
    stepButStepEnable = false;
}

template<class T>
void TreapNode<T>::loopLatency(){
      qDebug() << "Loop latency for step-by-step started" <<
endl;
    for( ; ; ) {
        QApplication::processEvents();
        if(stepLoopSwitcher == false) break;
    }
    stepLoopSwitcher = true;
}


#endif // TREAPNODE_H
```

Название файла: drawing.cpp

```
#include "mainwindow.h"
#include "headers.h"
#include "treapNode.h"

void MainWindow::graphic(TreapNode<ul>* node, bool isRoot) {
    if (!workingTreap)
        return;
    QPen pen;
    QColor color;
    color.setRgb(196, 242, 209);
    pen.setColor(color);
    QBrush brush (color);
    QFont font;
    font.setFamily("Tahoma");
    pen.setWidth(2);
    int wDeep = static_cast<int>(pow(2, node->height())+2);
    int hDelta = 70;
```

```
        int wDelta = 15;
        font.setPointSize(11);
        font.setBold(true);
        int width = (wDelta*wDeep)/2;
        mainGraphicsView->setScene(mainGraphicsScene);
        if(isRoot){
            node->setStartPos(width/2, hDelta);
            if(wDelta*wDeep > 400)
                node->setWidth(wDelta*wDeep + 2*hDelta);
            else
                node->setWidth(2*(wDelta*wDeep));
        }
        else{
            node->setWidth(4*width+hDelta);
        }
              treePainter(node, node->getStartPos().x(), node-
>getStartPos().y(), wDelta, hDelta, pen, brush, font, wDeep);
    }

    void MainWindow::treePainter(TreapNode<ul> *node, int w, int
h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font,
int depth) {
        if (!node)
            return;
        if(node->stepButStepEnable) node->loopLatency();
        QGraphicsTextItem *textItem = new QGraphicsTextItem;
        textItem->setPos(w, h);
         textItem->setPlainText(QString::number(node->getKey())+":
"+QString::number(node->getPriority()));
        textItem->setFont(font);
              mainGraphicsScene->addEllipse(w-wDelta/2, h, node-
>MajorAxis, node->DrawingSize, pen, brush);
        if (node->mLeft)
               mainGraphicsScene->addLine(w+wDelta/2, h+wDelta, w-
(depth/2)*wDelta+wDelta/2, h+hDelta+wDelta, pen);
        if (node->mRight)
               mainGraphicsScene->addLine(w+wDelta/2, h+wDelta, w+
(depth/2)*wDelta+wDelta/2, h+hDelta+wDelta, pen);
        mainGraphicsScene->addItem(textItem);
           treePainter(node->mLeft, w-(depth/2)*wDelta, h+hDelta,
wDelta, hDelta, pen, brush, font, depth/2);
          treePainter(node->mRight, w+(depth/2)*wDelta, h+hDelta,
wDelta, hDelta, pen, brush, font, depth/2);
```

```
}
```

Название файла: main.cpp

```
#include <QApplication>

#include "mainwindow.h"

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

Название файла: mainnwindow.cpp

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow),
    fileOpen(new QPushButton),
    drawingTreap(new QPushButton),
    stepDrawingTreap(new QPushButton),
    splitTreap(new QPushButton),
    stepSwitch(new QCheckBox),
    mainGraphicsView(new QGraphicsView),
    mainGraphicsScene(new QGraphicsScene),
    inputLine(new QLineEdit),
    splitKeyInput(new QLineEdit),
    slpitError(new QLineEdit),
    inputExprLabel(new QLabel),
    splitElLabel(new QLabel)
{
    setUpUI();
}

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

```
    void MainWindow::createTreap(){
        string str = inputLine->text().toUtf8().constData();
        string token{};
        slpitError->clear();
        for(ul i=0; i<str.length(); i++){
            if(isdigit(str[i])){
                token += str[i];
            }
            else if(!token.empty()){
                ul priority = rand() % RAND_FACTOR + 1;
                if(!workingTreap) {
                                                workingTreap = new
TreapNode<ul>(static_cast<ul>(stoi(token)), priority);
                }
                else{
                                            workingTreap = workingTreap-
>add(static_cast<ul>(stoi(token)), priority);
                }
                if(!(TreapNode<ul>::forErrors.empty())){
                                                    slpitError-
>setText(QString::fromStdString(TreapNode<ul>::forErrors));
                    TreapNode<ul>::forErrors.clear();
                    return;
                }
                token.clear();

            }
        }
        if(!token.empty()){
            ul priority = rand() % RAND_FACTOR + 1;
                        workingTreap = workingTreap-
>add(static_cast<ul>(stoi(token)), priority);
            token.clear();
        }

    }
    void MainWindow::onFileOpenButtonClicked() {
        std::string inputStr;
            QString fileName = QFileDialog::getOpenFileName(this,
                    tr("Open TXT File"),  QDir::homePath(),
                    tr("TXT text (*.txt);;All Files (*)"));
            if (fileName == nullptr)
            {
```

```
                qDebug() << "No file name" << endl;
                return;
            }
            QFile file(fileName);
              if(file.open(QIODevice::ReadOnly | QIODevice::Text))
{
                QTextStream stream(&file);

inputStr.append(stream.readAll().toUtf8().constData());
            }

            if(inputStr.empty())
                return;

            file.close();

                                                    inputLine-
>setText(QString::fromUtf8(inputStr.c_str()));
    }

    void MainWindow::onTreapDrawingButtonClicked() {
        //if(isDrawing && stepSwitch->isChecked()){
          //  TreapNode<int>::changeStepByStepMode();
            //return;
       // }
        if(workingTreap){
            qDebug() << "clear" << endl;
            TreapNode<ul>::clear(workingTreap);
            workingTreap = nullptr;
            mainGraphicsScene->clear();
        }
        if(stepSwitch->isChecked()) {
            stepDrawingTreap->setDisabled(false);
            TreapNode<ul>::changeStepByStepMode();
            isDrawing = true;
        }
        else {
            stepDrawingTreap->setDisabled(true);
            TreapNode<ul>::changeLoopState();
            TreapNode<ul>::setDisableStepMode();
        }

        createTreap();
```

```
        graphic(workingTreap, true);

    }


    void MainWindow::onStepTreapDrawingButtonClicked() {
        TreapNode<ul>::changeLoopState();
    }

    void MainWindow::onTreapSplitingButtonClicked() {
        mainGraphicsScene->clear();
        if(!workingTreap){
            createTreap();
        }
        graphic(workingTreap, true);
        TreapNode<ul>* treapLeft = new TreapNode<ul>();
        TreapNode<ul>* treapRight = new TreapNode<ul>();
        if(!splitKeyInput->text().toUtf8().constData()){
                slpitError->setText(QString::fromStdString("please
input key"));
        }
             ul key = static_cast<unsigned long>(splitKeyInput-
>text().toLong());
        workingTreap->split(key, treapLeft, treapRight);
        if(treapLeft){
                        treapLeft->setStartPos(workingTreap-
>getStartPos().x()+workingTreap->getWidth(), workingTreap-
>getStartPos().y());
            graphic(treapLeft, false);
        }if(treapRight){
            if(treapLeft)
                                treapRight->setStartPos(treapLeft-
>getStartPos().x()+treapLeft->getWidth(), treapLeft-
>getStartPos().y());
            else
                            treapRight->setStartPos(workingTreap-
>getStartPos().x()+workingTreap->getWidth(), workingTreap-
>getStartPos().y());
            graphic(treapRight, false);
        }

    }
```

Название файла: setUpUi.cpp

```
#include "mainwindow.h"
#include "headers.h"

void MainWindow::setUpUI() {
    QWidget *canvas = new QWidget;
    QVBoxLayout *layout = new QVBoxLayout;

    canvas->setLayout(layout);

    layout->setStretch(1, 2);
    layout->setAlignment(Qt::AlignCenter);

    setCentralWidget(canvas);

    fileOpen->setText("file");
    drawingTreap->setText("Draw treap");
    stepDrawingTreap->setText("Step of drawing");
    splitTreap->setText("Split");
    stepSwitch->setText("Step mode");
    inputExprLabel->setText("Input:");
    splitElLabel->setText("Split by: ");
            inputLine->setValidator(new
QRegExpValidator(QRegExp("[0-9()\\s]*"), inputLine));
          splitKeyInput->setValidator(new
QRegExpValidator(QRegExp("[0-9()]*"), splitTreap));
    mainGraphicsView->setMinimumSize(500, 500);


    QHBoxLayout *exprWorkLayout = new QHBoxLayout();
    QVBoxLayout *allButLayout = new QVBoxLayout();
          QVBoxLayout *mainWorkingLayout = new
QVBoxLayout();
    QVBoxLayout *splitLayout = new QVBoxLayout();
    QHBoxLayout *splitMainLayout = new QHBoxLayout();
    splitMainLayout->addWidget(splitElLabel);
    splitMainLayout->addWidget(splitKeyInput);
```

```
        splitMainLayout->addWidget(splitTreap);
        splitLayout->addLayout(splitMainLayout);
        splitLayout->addWidget(slpitError);
             QHBoxLayout *inputTreapLayout = new
QHBoxLayout();
        inputTreapLayout->addWidget(inputExprLabel);
        inputTreapLayout->addWidget(inputLine);
        inputTreapLayout->addWidget(fileOpen);
        QHBoxLayout *drawingLayout = new QHBoxLayout();
        drawingLayout->addWidget(drawingTreap);
        drawingLayout->addWidget(stepSwitch);
        drawingLayout->addWidget(stepDrawingTreap);

        allButLayout->addLayout(inputTreapLayout);
        allButLayout->addLayout(splitLayout);
        allButLayout->addLayout(drawingLayout);

        exprWorkLayout->addLayout(allButLayout);
        exprWorkLayout->addWidget(mainGraphicsView);

        mainWorkingLayout->addLayout(exprWorkLayout);

        layout->addLayout(mainWorkingLayout);

        // default conditions
        stepSwitch->setChecked(false);
        stepDrawingTreap->setDisabled(true);

          connect(drawingTreap, SIGNAL(clicked()), this,
SLOT(onTreapDrawingButtonClicked()));
             connect(fileOpen, SIGNAL(clicked()), this,
SLOT(onFileOpenButtonClicked()));
            connect(stepDrawingTreap, SIGNAL(clicked()),
this, SLOT(onStepTreapDrawingButtonClicked()));
           connect(splitTreap, SIGNAL(clicked()), this,
SLOT(onTreapSplitingButtonClicked()));

    }
```

Название файла: mainnwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>800</width>
    <height>600</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralwidget"/>
  <widget class="QMenuBar" name="menubar">
   <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>800</width>
     <height>22</height>
    </rect>
   </property>
  </widget>
  <widget class="QStatusBar" name="statusbar"/>
 </widget>
 <resources/>
 <connections/>
</ui>
```