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. 7383 _____ Левкович Д.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Левкович Д.В.

Группа 7383

Тема работы: Бинарные деревья поиска

Исходные данные:

Написать программу, генерирующая входные данные и фиксирующая результаты испытаний работы программы.

Содержание пояснительной записки:

- Содержание
- Введение
- Функции для работы с бинарными деревьями поиска
- Тестирование
- Заключение
- Приложение А. Исходный код программы.

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания:

Дата сдачи реферата:

Дата защиты реферата:

Студент _____ Левкович Д.В.

Преподаватель _____ Размочаева Н.В.

АННОТАЦИЯ

В данной курсовой работе была разработана программа на языке программирования C++, которая позволяет создать рандомизированное дерево поиска и были разработаны функции и для работы с данным деревом. Набор функций включает в себя функцию добавления нового узла в дерево, удаление узла, если этот узел есть в дереве, правого и левого поворотов дерева, склеивания двух деревьев, поиска элемента и удаления самого дерева. Для представления бинарного дерева в памяти был использован класс, который хранит в себе информацию о размере дерева, значения в узле дерева и адреса на левое и правое поддеревья.

SUMMARY

In this course work, a program was developed in the programming language C ++, which allows you to create a randomized search tree and functions were developed for working with this tree. The set of functions includes the function of adding a new node to the tree, deleting the node if this node is in the tree, right and left turning the tree, gluing two trees, searching for an element and deleting the tree itself. To represent the binary tree in memory, a class was used that stores information about the size of the tree, values in the tree node and addresses to the left and right subtrees.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Требуется разработать программу, которая создает рандомизированное дерево поиска, удалить из него элемент, если он есть. Провести исследование. Тестирование будет проводиться в Qt 5.11.2 в ОС Windows 10.

1. ФОРМУЛИРОВКА ЗАДАЧИ И ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Написать программу, которая строит бинарное дерево поиска с рандомизацией, генерирует входные данные для использования их для измерения количественной характеристики структуры данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими.

Бинарное дерево поиска с рандомизацией

Определение:

Пусть T — бинарное дерево поиска. Тогда

1. Если T – пусто, то оно является рандомизированным бинарным деревом (в дальнейшем RBST).

2. Если T – непусто (содержит n вершин, $n > 0$), то T – рандомизированное бинарное дерево поиска тогда и только тогда, когда его левое и правое поддеревья оба являются рандомизированными бинарными деревьями поиска, а также выполняется соотношение $P[\text{size}(L) = i] = 1/n$, $i = 1..n$.

Из определения следует, что каждый ключ в RBST размера n может оказаться корнем с вероятностью $1/n$.

Идея RBST состоит в том, что хранимое дерево постоянно является рандомизированным бинарным деревом поиска. Далее подробно будет описана реализация операций над RBST, которая позволит добиться этой цели. Заметим лишь, что хранение RBST в памяти ничем не отличается от хранения обычного дерева поиска: хранится указатель на корень; в каждой вершине хранятся указатели на её сыновей.

2. ФУНКЦИИ И СТРУКТУРЫ ДЛЯ РАБОТЫ С БИНАРНЫМ ДЕРЕВОМ ПОИСКА

2.1. Класс class BST

Класс содержит в себе поле int key для хранения значения в узле, поле для хранения текущего размера дерева int size и указатели BST* right, BST* left на правое и левое поддерево соответственно.

2.2. Конструктор BST(int k)

Конструктор инициализирует поля нашего класса.

2.3. Деструктор ~BST()

Используется для удаления нашего дерева.

2.4. Методы BST* rotateright и BST* rotateleft

Используется для правого и левого поворота дерева при вставке в корень дерева. То есть эти методы позволяют поднять нужный нам узел в корень дерева.

2.5. Метод BST* insertroot

Используется для классической вставки элемента в корень дерева. Сначала рекурсивно вставляем новый ключ в корень левого или правого поддеревьев (в зависимости от результата сравнения с корневым ключом) и выполняем правый (левый) поворот, который поднимает нужный нам узел в корень дерева.

2.6.Метод BST* insertrandom

Используется для рандомизированной вставки нового элемента в дерево, используя метод классической вставки. Если n – размер дерева до вставки, то любой ключ может оказаться корнем с вероятностью $1/(n+1)$, тогда с указанной вероятностью выполняем вставку в корень, а с вероятностью 1 -

$1/(n+1)$ – рекурсивную вставку в правое или левое поддерево в зависимости от ключа в корне.

2.7. Методы int getsize и void fixsize

Используются для хранения размера дерева и для рандомизированной вставки.

2.8. Метод BST* join

Используется при удалении узла из дерева и последующего соединения двух деревьев.

2.9. Метод BST* remove

Используется для удаления узла из дерева по заданному ключу. Если заданный ключ для удаления больше, чем значение в корне тогда выполняем эту же процедуру для правого поддерева, если меньше, то для левого. Используем основное свойство бинарного дерева поиска – любой ключ в левом поддереве меньше, чем в корне, а в правом – больше чем в корне.

3. ТЕСТИРОВАНИЕ

На Рисунок 1 показан пример генерации случайной последовательности ключей и создания дерева.

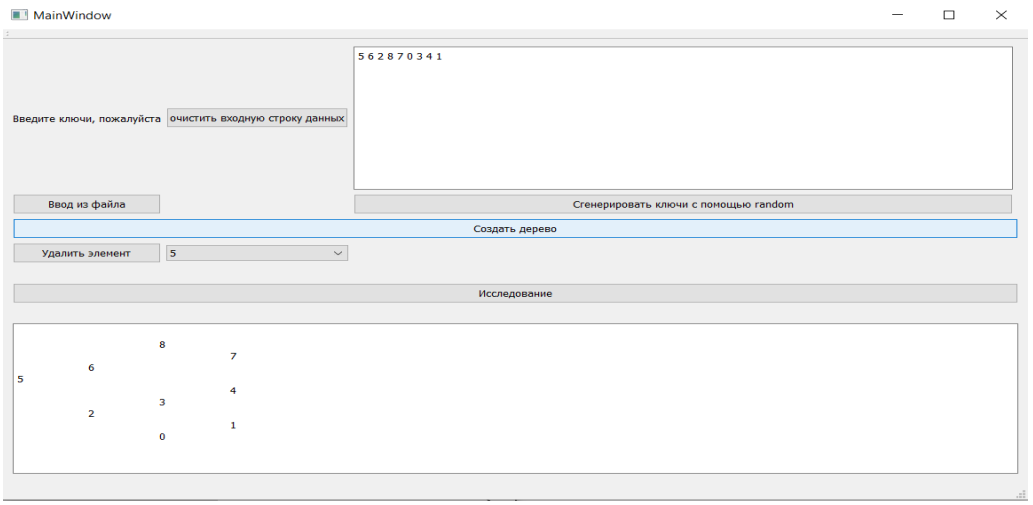

Рисунок 1 – Генерация ключей и создание дерева

На Рисунок 2 показан пример удаления узла с ключом рамным двум в дереве и перестроение самого дерева.

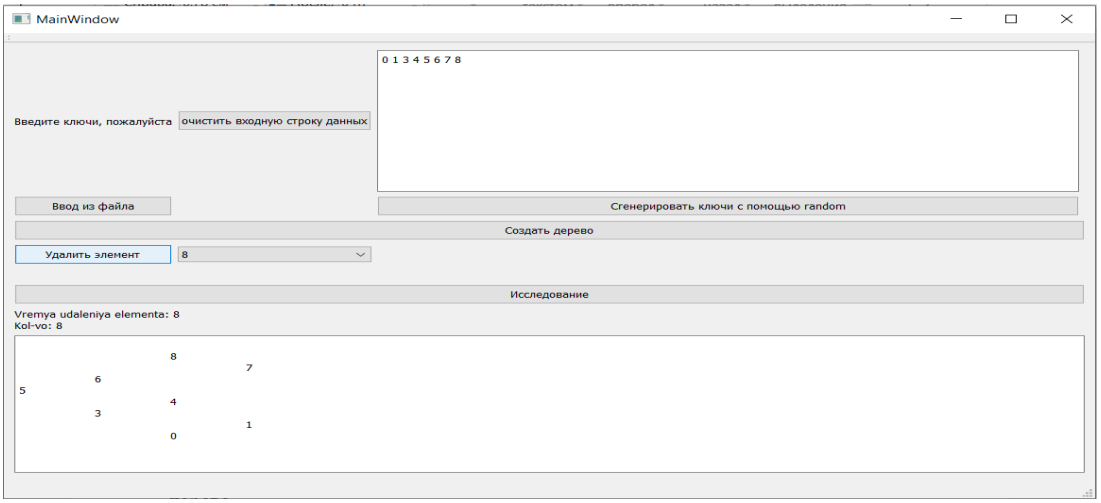

Рисунок 2 – Удаление элемента из дерева

4. ИССЛЕДОВАНИЕ

На рис. 3 приведен график зависимости времени добавления удаления узлов из дерева от количества узлов в дереве.

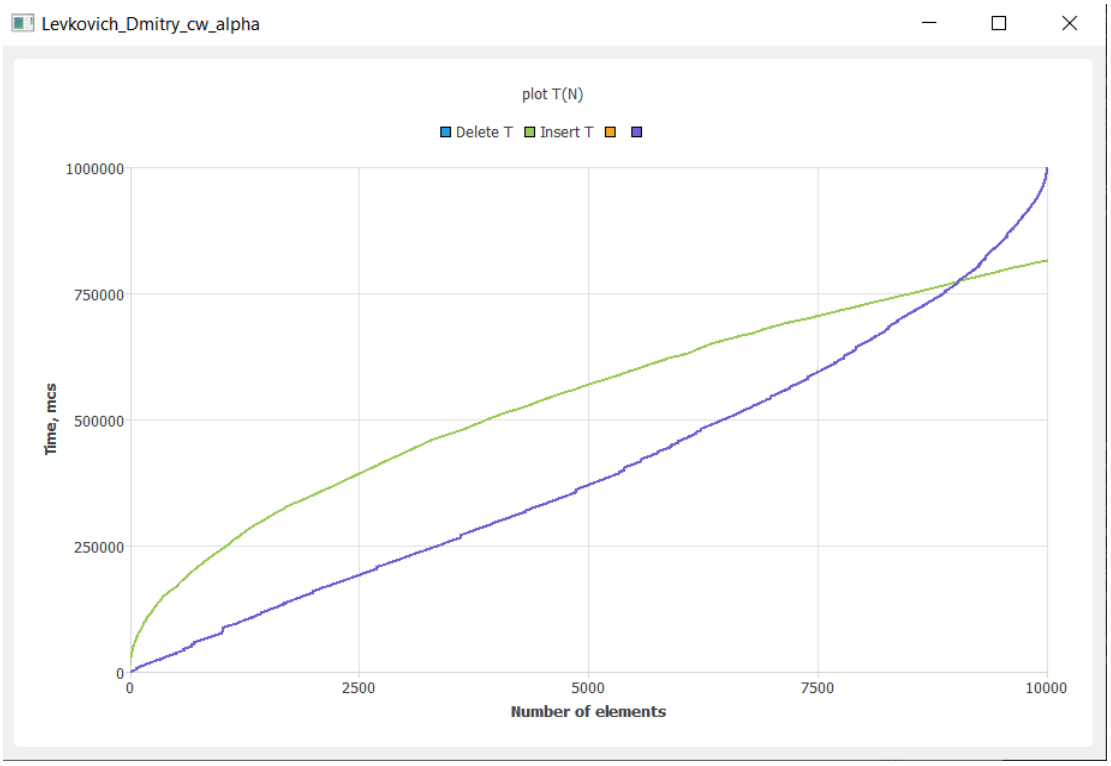

Рисунок 3 - График зависимости времени добавления и удаления узлов из дерева от количества узлов в дереве

На рис. 4 представлен график зависимости количества интераций от количества узлов в дереве в лучшем случае и в худшем случае.

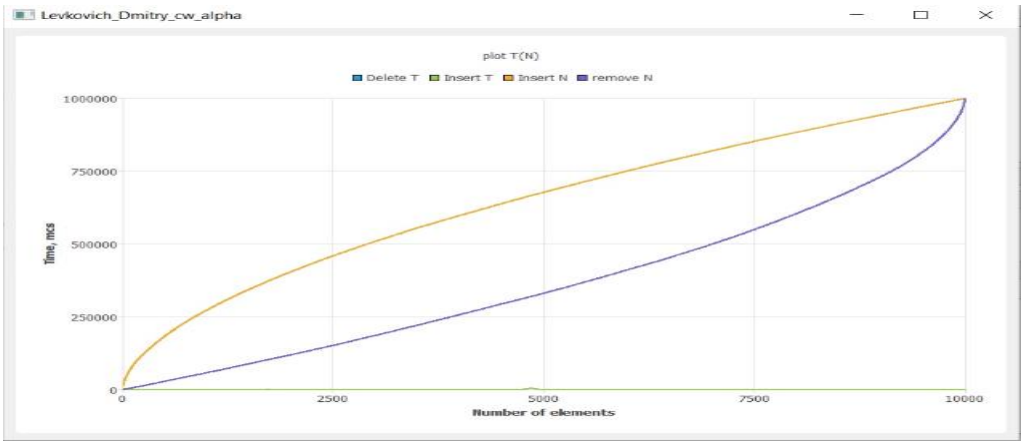

Рисунок 4 - График зависимости количества итераций от количества узлов

Как видно из графиков на Рисунок 3 и Рисунок 4, время удаления элементов требует меньше времени, чем время вставки, это может быть связано с тем, что удаляемый элемент может встретиться раньше, поэтому время удаления меньше, а вставка происходит в конец дерева и после этого происходит перерасчет.

ЗАКЛЮЧЕНИЕ

Как видно из Рисунок 3 и Рисунок 4, в лучшем случае и в среднем график представляет собой график логарифмической функции, из этого можно сделать вывод, что скорость добавления и удаления совпадает с теоретической O(log(N)) как в лучшем случае (см. Рисунок 3 и Рисунок 4), так и в худшем O(N).

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл 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();
}
```

Файл mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <QDebug>
#include "tree.h"
#include <QProcess>
#include <QFile>
#include <QFileDialog>
#include <QTextStream>
#include <QKeyEvent>
#include <windows.h>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLabel>
#include <QMessageBox>
#include<QRegExpValidator>
#include <math.h>
#include <vector>
#include <cstdlib>
#include <algorithm>
#include <QtCharts/QtCharts>
#include <QLogValueAxis>
#include <QLineSeries>
#include <QValueAxis>
#include <QChart>
#include <QChartView>
#include <QChar>
#include <chrono>
QT_CHARTS_USE_NAMESPACE
```

```
using namespace std;

BST* tree = 0;
int numberTreeEl;
vector<int> values(numberTreeEl);

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

MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::on_pushButton_clicked()
{
    if(tree!=nullptr){
        tree=tree->Delete(tree);
        tree==nullptr;
    }
    tree->random_init();
    QString str = ui->textEdit->toPlainText();
    for(int i = 0;i<str.size();i++){
        if(str[i].isDigit() || str[i].isSpace()){
            continue;
        }
        else{
            ui->textEdit->setText(ui->textEdit->toPlainText()+"\nincorrect
input in " + QString::number(i+1)+"th position: "+"'"+str[i]+"'");
            ui->textEdit_2->clear();
            return;
        }
    }
    QStringList list = str.split(QRegExp("\\s"), QString::SkipEmptyParts);
    ui->comboBox->clear();
    for (int i=0; i<list.length(); i++)
    {
        QString x;
        x=list[i];

        if(!(tree->find(tree,x.toInt()))){
            ui->comboBox->addItem(x);
            tree = tree->insertrandom(tree, x.toInt());
            numberTreeEl= tree->counter(tree);
        }
    }
    ui->textEdit_2->clear();
```

```
    print(tree, 0);
}

void MainWindow::on_pushButton2_clicked()
{
    tree->random_init();
    srand(time(nullptr));
    int len = 10;
    vector<int> x(len);
    for(int i = 0;i<len;i++){
        x[i] = i;
    }
    random_shuffle(x.begin(),x.end());
    ui->textEdit->clear();
    for (int i=0; i < len; i++)
    {
        ui->textEdit->setText(ui->textEdit->toPlainText() +
QString::number(x[i]) + " ");

    }

}

void MainWindow::on_pushButton_3_clicked()
{
    QString fileName = QFileDialog::getOpenFileName(this, tr("kek"),
QString(),
                tr("Text Files (*.txt)"));

        if (!fileName.isEmpty())
        {
            QFile file(fileName);
            if (!file.open(QIODevice::ReadOnly))
            {
                QMessageBox::critical(this, tr("lel"), tr("lal"));
                return;
            }
            QTextStream in(&file);
            ui->textEdit->setText(in.readAll());
            file.close();
        }

}

void MainWindow::on_pushButton_2_clicked()
{
    if(tree==nullptr){
        ui->label_3->setText("tree is empty\n");
        return;
    }
    tree->random_init();
```

```
    LARGE_INTEGER t1, t2, f;
    QueryPerformanceFrequency(&f);
    QString string = ui->comboBox->currentText();
    QString x = string;
    int y = x.toInt();
    QueryPerformanceCounter(&t1);
    tree = tree->remove(tree,y);
    QueryPerformanceCounter(&t2);
    numberTreeEl= tree->counter(tree);
    double dt =(t2.QuadPart-t1.QuadPart);
    int i = ui->comboBox->findText(x);
    ui->comboBox->removeItem(i);
    ui->label_3->setText("Vremya udaleniya elementa: " +
QString::number(dt)+"\nKol-vo: "+QString::number(numberTreeEl)+"
"+QString::number(tree->count()));
    tree->zero();
    ui->textEdit_2->clear();
    print(tree, 0);
    ui->textEdit->clear();
    infix(tree);
}

void MainWindow::print(BST* treenode, int l){
    if(treenode==nullptr){
        for(int i = 0;i<l;++i)
            ui->textEdit_2->setText(ui->textEdit_2->toPlainText() + "\t");
        ui->textEdit_2->setText(ui->textEdit_2->toPlainText() + "\r\n");
        return;
    }
    print(treenode->Right(treenode), l+1);
    for(int i = 0; i < l; i++)
        ui->textEdit_2->setText(ui->textEdit_2->toPlainText() + "\t");
    ui->textEdit_2->setText(ui->textEdit_2-
>toPlainText()+QString::number(treenode->Root(treenode)) + "\t");
    print(treenode->Left(treenode),l+1);
}

void MainWindow::infix(BST* b){
    if(b!=nullptr){

        infix(b->Left(b));
        ui->textEdit->setText(ui->textEdit->toPlainText() +
QString::number(b->Root(b))+" ");
        infix(b->Right(b));

    }
}

int k = 0;

void MainWindow::printlevel(BST *t, int level) {
```

```
    if (t == nullptr) {
        // Если дерево пустое, то отображать нечего, выходим
        return;
    } else {
        printlevel(t->Left(t), level - 1);  // С помощью рекурсии посещаем
левое поддерево
        if (level == 0) {
            // level будет равен нулю на нужной глубине, так как при каждом
рекурсивном вызове значение level уменьшается на один
            this->k = t->Root(t);  // Показываем элемент, если он на нужном
нам уровне
            return;
        }
        printlevel(t->Right(t), level - 1);  // С помощью рекурсии посещаем
правое поддерево
    }
}

void MainWindow::on_pushButton_4_clicked()
{
    int len = 10000;
    vector<int> x(len), w(len), q(len);
    QChart *chart = new QChart();
    QLineSeries *deleteEl = new QLineSeries();
    QLineSeries *insert = new QLineSeries();
    QLineSeries *delete1 = new QLineSeries();
    QLineSeries *insert1 = new QLineSeries();
    insert1->setName("Insert N");
    delete1->setName("remove N");
    for(int i = 0;i<len;i++)
        x[i] = i;
    random_shuffle(x.begin(),x.end());
    int ans1 = 0;
    int N = 0;
    for(int i = 0;i<len;i++){
        tree = tree->insertrandom(tree, x[i]);
        ans1 += tree->counter(tree);
        N =tree->count1();
        insert1->append(ans1, N);
    }
    ans1 = 0;
    N = 0;
    for(int i = 0;i<len;i++){
        tree=tree->remove(tree, x[i]);
        ans1 += tree->counter(tree);
        N = tree->count();

        delete1->append(ans1, N);


    }
```

```
    chart->addSeries(deleteEl);
    chart->addSeries(insert);
    chart->addSeries(insert1);
    chart->addSeries(delete1);
    chart->setTitle("plot iterations(N)");
    QValueAxis *axisX = new QValueAxis();
    axisX->setLabelFormat("%d");
    axisX->setTitleText("Number of elements");
    axisX->setRange(0, len);
    chart->addAxis(axisX, Qt::AlignBottom);
    deleteEl->attachAxis(axisX);
    insert->attachAxis(axisX);
    QValueAxis *axisY = new QValueAxis();
    axisY->setTitleText("inerations");
    axisY->setLabelFormat("%lld");
    axisY->setRange(0, 1000000);
    chart->addAxis(axisY, Qt::AlignLeft);
    deleteEl->attachAxis(axisY);
    insert->attachAxis(axisY);
    QChartView* chartView = new QChartView(chart);
    chartView->resize(920, 600);
    chartView->show();
    plot();
}

void MainWindow::plot()
{
    int len = 10000;
    vector<int> x(len);
    QChart *chart = new QChart();
    QLineSeries *deleteEl = new QLineSeries;
    QLineSeries *insert = new QLineSeries();
    QLineSeries *delete1 = new QLineSeries();
    QLineSeries *insert1 = new QLineSeries();
    delete1->setName("Delete T");
    insert->setName("Insert T");
    for(int i = 0;i<len;i++)
        x[i] = i;
    random_shuffle(x.begin(),x.end());
    auto ans = 0;
    int ans1 = 0;
    for(int i = 0;i<len;i++){
        auto begin = std::chrono::steady_clock::now();
        tree = tree->insertrandom(tree, x[i]);
        auto end = std::chrono::steady_clock::now();
        auto elapsed_ns = (
std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count());
        ans += elapsed_ns;
        ans1 += tree->counter(tree);
            insert->append(ans1, 10*ans);
    }
```

```
    ans = 0;
    ans1 = 0;
    for(int i = 0;i<len;i++){
        auto begin = std::chrono::steady_clock::now();
        tree=tree->remove(tree, x[i]);
        auto end = std::chrono::steady_clock::now();
        auto elapsed_ns = (
std::chrono::duration_cast<std::chrono::nanoseconds>(end - begin).count());
        ans += elapsed_ns;
        ans1 += tree->counter(tree);
            delete1->append(ans1, 10*ans);

    }
    chart->addSeries(deleteEl);
    chart->addSeries(insert);
    chart->addSeries(insert1);
    chart->addSeries(delete1);
    chart->setTitle("plot T(N)");
    QValueAxis *axisX = new QValueAxis();
    axisX->setLabelFormat("%d");
    axisX->setTitleText("Number of elements");
    axisX->setRange(0, len);
    chart->addAxis(axisX, Qt::AlignBottom);
    deleteEl->attachAxis(axisX);
    insert->attachAxis(axisX);
    QValueAxis *axisY = new QValueAxis();
    axisY->setTitleText("Time, mcs");
    axisY->setLabelFormat("%lld");
    axisY->setRange(0, 1000000);
    chart->addAxis(axisY, Qt::AlignLeft);
    deleteEl->attachAxis(axisY);
    insert->attachAxis(axisY);
    QChartView* chartView = new QChartView(chart);
    chartView->resize(920, 600);
    chartView->show();

}

void MainWindow::on_pushButton_5_clicked()
{
    ui->textEdit->clear();
}

void MainWindow::on_pushButton_6_clicked()
{
    int h = tree->height(tree);
    printlevel(tree, h-1);
    tree->remove(tree, this->k);
    ui->label_4->setText(QString::number(k)+" "+QString::number(tree-
>count()));
    tree->zero();
```

```
    ui->textEdit_2->clear();
    print(tree, 0);
}
```

Файл mainwindow.h:

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include "tree.h"
class SecondWindow;
namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:
    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

    void on_pushButton3_clicked();

    void on_pushButton2_clicked();

    void on_pushButton_3_clicked();

    void on_label_2_linkActivated();

    void print(BST*, int);

    void printlevel(BST*, int);

    void infix(BST*);

    void on_pushButton_4_clicked();

    void on_pushButton_5_clicked();
    void plot();
    void on_pushButton_6_clicked();

private:
    Ui::MainWindow *ui;
```

```
    SecondWindow *window;
    int k;
};


#endif // MAINWINDOW_H
```

Файл tree.cpp:

```
#include <iostream>
#include "tree.h"
#include <ctime>
#include <cstdlib>
#include "mainwindow.h"
using namespace std;

int N = 0, M = 0;

BST::BST(int k){
    key = k;
    left = right = nullptr;
    size = 1;
}

int BST::Root(BST* b)
{
    if (b == nullptr)
         exit(1);
    else
        return b->key;
}

BST* BST::Left(BST* b)
{
    if (b == nullptr)
        exit(1);
    else
        return b->left;
}

BST* BST::Right(BST* b)
{
    if (b == nullptr)
        exit(1);
    else
        return b->right;
}

int BST::getsize(BST* p){
```

```
    if(!p)
        return 0;
    return p->size;
}

void BST::fixsize(BST* p){
    p->size = getsize(Left(p))+getsize(Right(p))+1;
}

BST* BST::rotateright(BST* p){
    BST* q = p->left;
    if( !q )
        return p;
    p->left = q->right;
    q->right = p;
    q->size = p->size;
    fixsize(p);
    return q;
    }

BST* BST::rotateleft(BST* q){
    BST* p = q->right;
    if( !p )
        return q;
    q->right = p->left;
    p->left = q;
    p->size = q->size;
    fixsize(q);
    return p;
}

BST* BST::insert(BST* p, int k) // классическая вставка нового узла с ключом
k в дерево p
{
    M++;
    if(!p) return new BST(k);
    if( p->key>k )
        p->left = insert(p->left,k);
    else
        p->right = insert(p->right,k);
    fixsize(p);
    return p;
}

BST* BST::insertroot(BST* p, int k){
    M++;
    if( !p ){
        return new BST(k);
    }

    if( k<p->key ){
```

```
        p->left = insertroot(p->left,k);
        return rotateright(p);
    }
    else{
        p->right = insertroot(p->right,k);
        return rateleft(p);
    }
}

BST * BST::insertrandom(BST *p, int k){
   M++;
    if(!p)
        return new BST(k);
    if(rand()%(getsize(p)+1)== 0)
        return insertroot(p,k);
    if( p->key>k )
        p->left = insert(Left(p),k);
    else
        p->right = insert(Right(p),k);
    fixsize(p);
    return p;
}

bool BST::isNull(BST* b){
    return b==nullptr;
}
BST* BST::join(BST* p, BST* q){
    N++;
    if( !p )
        return q;
    if( !q )
        return p;
    if( rand()%(p->size+q->size) < p->size ){
        p->right = join(p->right,q);
        fixsize(p);
        return p;
    }
    else{
        q->left = join(p,q->left);
        fixsize(q);
        return q;
    }
}

int BST::max(BST* p){
    if(p->right==nullptr)
        return p->key;
    else max(p->right);
}

BST* BST::remove(BST* p, int k){
```

```
    N++;
    if( !p )
        return p;
    if( p->key==k ){
        BST* q = join(p->left,p->right);
        p=q;
        return p;
    }
    else if( k<p->key )
        p->left = remove(p->left,k);
    else
        p->right = remove(p->right,k);
    return p;
}


void BST::zero(){
    N = 0;
    return;
}

void BST::zero1(){
    M = 0;
    return;
}

int BST::count(){

    return N;
}

int BST::count1(){

    return M;
}


BST* BST::Delete(BST* p){
    if (p->left)
        delete p->left;
    if (p->right)
        delete p->right;
    delete p;
    return p = nullptr;
}

BST* BST::find( BST* tree, int key){
    if(!tree)
        return nullptr;
    if(key == tree->key)
        return tree;
```

```
    if(key < tree->key)
        return find(tree->left, key);
    else
        return find(tree->right, key);
}

void BST::random_init(){
    srand(time(nullptr));
}


int BST::counter(BST* tree){
    int x = 1;
    if(tree == nullptr)
        return 0;
     x+=counter(tree->left);
     x+=counter(tree->right);
     return x;
}

int BST::height(BST* b) {
    if(b==nullptr)
        return 0;
       if(height(b->left)>height(b->right))
           return height(b->left)+1;
    else
        return height(b->right)+1;
}

BST::~BST(){
    delete left;
    delete right;
}
```

Файл tree.h:

```
#ifndef TREE_H
#define TREE_H
#include <ctime>
#include <cstdlib>
#include <iostream>

using namespace std;

class BST{
public:
    BST(int k);
    int Root(BST* b);
    BST *Left(BST* b);
    BST *Right(BST* b);
    int getsize(BST* p);
```

```
    void fixsize(BST* p);
    BST *rotateright(BST* p);
    BST *rotateleft(BST* q) ;
    BST* insert(BST*, int);
    BST *insertroot(BST* p, int k);
    BST  *insertrandom(BST *p, int k);
    BST* join(BST* p, BST* q);
    BST* remove(BST* p, int k);
    BST* Delete(BST* p);
    BST* find( BST* tree, int key);
    void random_init();
    int counter(BST* tree);
    bool isNull(BST* b);
    int max(BST*);
    int height(BST*);
    int count();
    void zero();
    int count1();
    void zero1();
    ~BST();
private:
    int key;
    int size;
    BST* right;
    BST* left;
};

#endif // TREE_H
```