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 пирамиды поиска (Treaps) – вставка и исключение. Демонстрация

Студентка гр. 7383 _____ Маркова А.В.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Маркова А.В.

Группа 7383

Тема работы: Рандомизированные пирамиды поиска (Treaps) – вставка и исключение. Демонстрация

Содержание пояснительной записки:

- Содержание
- Введение
- Теоретические сведения. Описание функций
- Решение задачи
- Примеры работы программы
- Заключение
- Список используемых источников
- Приложение А. Исходный код программы

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 19.10.2018

Дата сдачи курсовой работы:

Дата защиты курсовой работы:

Студентка гр. 7383 _____ Маркова А.В.

Преподаватель _____ Размочаева Н.В.

АННОТАЦИЯ

В ходе курсовой работы была реализована программа на высокоуровневом языке программирования C++ с использованием движка Qt, который обеспечивает исключение и добавление элементов в рандомизированную пирамиду поиска (Treaps). Была осуществлена демонстрация построения пирамиды. Также реализован графический пользовательский интерфейс для удобной работы с программой.

SUMMARY

During the course work, a program was implemented in a high-level C ++ programming language using the Qt engine, which provides for the exclusion and addition of elements to the randomized search pyramid (Treaps). A demonstration of the construction of the pyramid was carried out. Also implemented a graphical user interface for convenient work with the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью данной курсовой работы является создание графического интерфейса (GUI) приложения на высокоуровневом языке программирования C++ с использованием фреймворка Qt для исключения и добавления элементов в рандомизированную пирамиду поиска (Treaps).

Для достижения поставленной цели требуется решить следующие задачи:

- Создание класса для построения рандомизированной пирамиды поиска;
- Создание пользовательского интерфейса для удобного взаимодействия с программой;
- Реализация функций демонстрации;
- Сборка и тестирование программы.

1.ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Декартово дерево (treap) – структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу.

Это структура данных, которая хранит пары (*x*,*y*) в виде бинарного дерева таким образом, что она является бинарным деревом поиска по *x* и бинарной пирамидой по *y*. Предполагая, что все *x* и все *y* являются различными (уникальными), 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 $y > y_0$.

Пирамиды были предложены Сиделом и Арагон в 1989 г.

x – являются ключами, *y* – приоритетами. Если бы приоритетов не было, то было бы обычное бинарное дерево поиска по *x*, и заданному набору могло бы соответствовать много деревьев, некоторые из которых являются вырожденными (в виде цепочки), а потому чрезвычайно медленными.

В то же время, приоритеты позволяют однозначно указать дерево, которое будет построено. Теперь очевидно, что если выбрать приоритеты случайно, то этим добьёмся построения невырожденных деревьев в среднем случае, что обеспечит асимптотику в среднем. Отсюда и понятно еще одно название этой структуры данных – рандомизированная пирамида поиска.

2.РЕШЕНИЕ ЗАДАЧИ

В данной курсовой работе было написано несколько функций и структура для работы с пирамилой поиска.

`struct node` – структура, представляющая узел БДП, содержит в себе поля int key для хранения ключа, `int prior` для хранения приоритета, `node* left`, right для хранения указателей на правое и левое поддерево.

`node* rotateright (node* p)` – функция, делающая правый поворот вокруг узла *p*.

`node* rotateleft(node* p)` – функция, делающая левый поворот вокруг узла *p*.

`node* insert(int key, node* root)` – функция, добавляющая узел с ключом *k*, учитывая его приоритет и ключ. Если ключ *k* больше (меньше) ключа рассматриваемого узла, то он вставляется вправо (влево) от этого узла, при надобности делается правый или левый поворот.

`node* merge(node *p, node *q)` – функция, получающая на вход два дерева, которые она объединяет.

`void printPriority(node* root)` – функция, печатающая приоритеты узлов, которые задаются случайным образом.

`node* remove(node* p, int k)` – функция получает на вход дерево и ключ узла. Удаляет узел с заданным ключом, объединяя его правое и левое поддерево с помощью функции merge.

`node* add(node* p, int el)` – функция, добавляющая элемент в пирамиду.

`node* find(node* tree, int key)` – функция для поиска элемента по ключу.

`void printtree(node* treenode, int l)` – функция, печатающая дерево.

`int main()` – головная функция, которая в зависимости от выбора пользователя считывает ключи из файла или с окна ввода, затем создает бинарное дерево, печатает приоритеты ключей, выводит само дерево и удаляет или добавляет узел, с заданным пользователем ключом.

Функция `void MainWindow::addKey()` позволяет добавить элемент в пирамиду, а `void MainWindow::removeKey()` – удалить. `void MainWindow::Calculate()` – функция, вызывающая обработку входной строки. void `MainWindow::inputFromFile()` – для работы с файлом, `void MainWindow::inputFromStr()` – со строкой.

При выборе пустого файла, при попытке создания пустой пирамиды и при некорректных входных данных выводятся сообщения об соответствующих ошибках.

3.ПРИМЕР РАБОТЫ ПРОГРАММЫ

Программа была написана в QtCreator 5.7.0 MinGW 32bit в операционной системе Windows 10. В других системах тестирование не проводилось.

На рис. 1 представлено удачное добавление элемента.

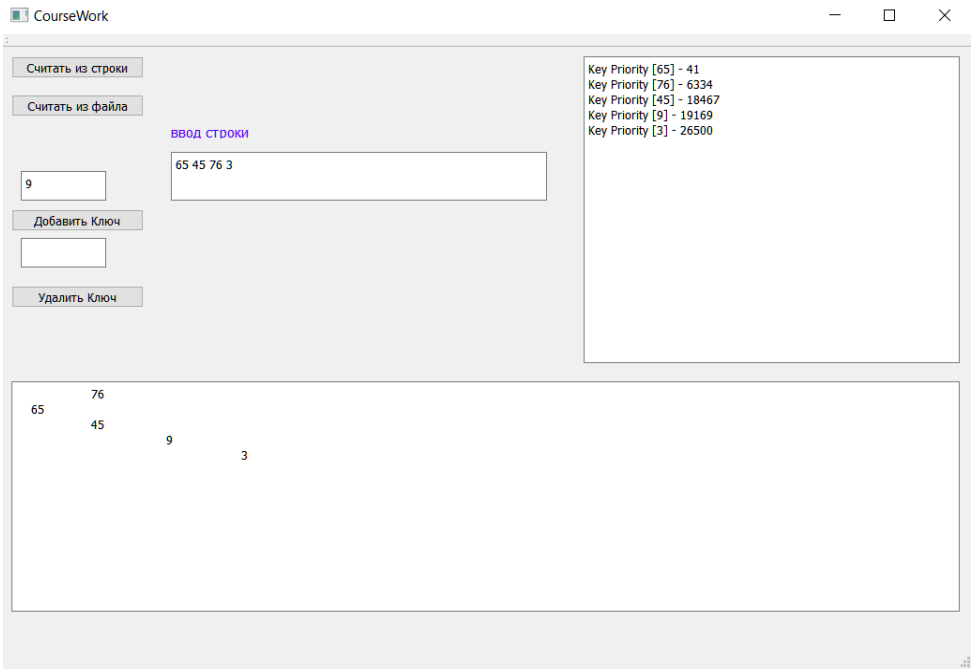

Рисунок 1 – Правильная работа программы

На рис. 2 показано окно выбора файла

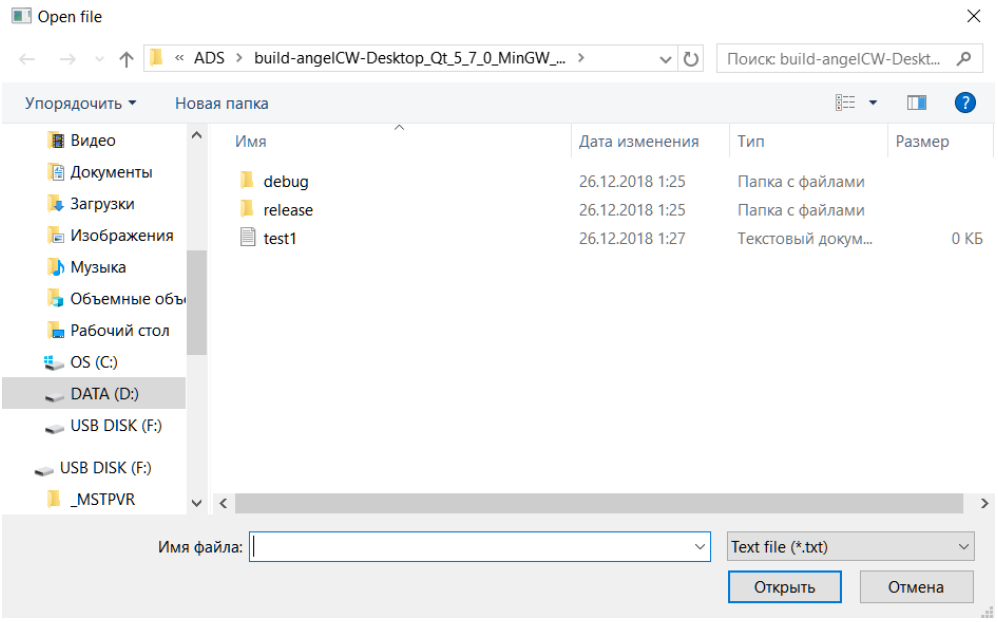

Рисунок 2 – Окно выбора файла

На рис. 3 представлены сообщения об ошибках, которые выводятся в отдельных окнах.

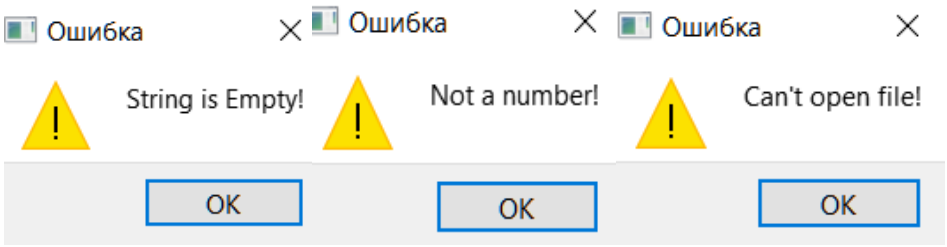

Рисунок 3 – Сообщения об ошибках

ЗАКЛЮЧЕНИЕ

В ходе курсовой работы была реализована программа на высокоуровневом языке программирования C++ с использованием движка Qt Creator для добавления и исключения элемента в рандомизированной пирамиде поиска. Также была реализована демонстрация вывода пирамиды.

Для взаимодействия с программой был реализован графический интерфейс, упрощающий работу пользователя.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

http://doc.qt.io/

http://e-maxx.ru/algo/treap

https://habr.com/post/145388/

https://intellect.ml/derevya-poiska-avl-derevo-splej-derevo-dekartovo-derevo-65

ПРИЛОЖЕНИЕ А ИСХОДНЫЙ КОД ПРОГРАММЫ

angelCW.pro

```
#-------------------------------------------------
#
# Project created by QtCreator 2018-12-24T21:18:07
#
#-------------------------------------------------

QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = PODRUGA5LAB
TEMPLATE = app


SOURCES += main.cpp\
        mainwindow.cpp

HEADERS  += mainwindow.h

FORMS    += mainwindow.ui
```

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <cmath>
#include <stdlib.h>
#include <QMainWindow>

struct node {          // структура для представления узлов дерева
    int key;           // ключ-значение
    long long prior;   // приоритет
    node* left;        // указатель на левое поддерево
    node* right;       // указатель на правое поддерево
    node(int k) {
        key = k;       // инициализация структуры
        left = right = NULL;
        prior = rand()%4294967296; // рандомные числа от 0 до 2^32
    }
```

```
};

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    node* add(node* p, int el,QString &err);
private slots:
    void inputFromStr();

    void inputFromFile();

    void Calculate();

    void addKey();

    void removeKey();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

main.cpp

```
#include "mainwindow.h"
#include <QApplication>

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

    return a.exec();
}
```

mainwindow.cpp

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

```
#include "ui_mainwindow.h"
#include <QFileDialog>
#include <QMessageBox>
#include <QString>
#include <QTextStream>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cstring>
#include <fstream>
#include <iomanip>
#include <cctype>

using namespace std;
QString Qstr = "";

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow){
    ui->setupUi(this);
    connect(ui->inputStr,SIGNAL(clicked()),this,SLOT(inputFromStr()));
    connect(ui-
>inputFile,SIGNAL(clicked()),this,SLOT(inputFromFile()));
    connect(ui->addKey,SIGNAL(clicked()),this,SLOT(addKey()));
    connect(ui->removeKey,SIGNAL(clicked()),this,SLOT(removeKey()));
}

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

node* treap = NULL;

void MainWindow::inputFromStr(){
    Qstr = ui->editStr->toPlainText();
    if (Qstr.isEmpty()){
        ui->treeEdit->setText("");
        ui->treapEdit->setText("");
        QMessageBox::warning(this, "Ошибка", "String is Empty!");
    }
    else {
        ui->treeEdit->setText("");
        ui->treapEdit->setText("");
        Calculate();
```

```
    }
}

void MainWindow::inputFromFile(){
    QString filename = QFileDialog::getOpenFileName(this,tr("Open
file"),"C://Qt//build-kursovik-Dekstop_Qt_5_7_0_MinGW_32bit-
Debug//","Text file (*.txt)");
    QFile file(filename.toStdString().data());
    if (!file.open(QIODevice::ReadOnly)){
        ui->treeEdit->setText("");
        ui->treapEdit->setText("");
        QMessageBox::warning(this, "Ошибка", "Can't open file!");
    }
    else{
        QTextStream in(&file);
        Qstr = in.readLine();
        file.close();
        ui->treeEdit->setText("");
        ui->treapEdit->setText("");
        Calculate();
    }
}


 node* rotateright(node* p){ // правый поворот вокруг узла p
     node* q = p->left;
     if( !q )
         return p;
     p->left = q->right;
     q->right = p;
     return q;
 }

 node* rotateleft(node* q){ // левый поворот вокруг узла q
     node* p = q->right;
     if( !p )
         return q;
     q->right = p->left;
     p->left = q;
     return p;
 }


 node* insert(int key, node* root){ // вставка
```

```
    if(!root) {
        node* p = new node(key);
        return (p);
    }
    if(key <= root->key){
        root->left = insert(key, root->left);
        if(root->left->prior < root->prior)
            root = rotateright(root);
    }
    else{
        root->right = insert(key, root->right);
        if(root->right->prior < root->prior)
            root = rotateleft(root);
    }
    return root;
}
void Delete(node* p){
    if(p==NULL)
        return;
    Delete(p->left);
    Delete(p->right);
    delete p;
}

node* merge(node *p, node *q){
    if (p == NULL) return q;
    if (q == NULL) return p;

    if (p->prior > q->prior){
        p->right = merge(p->right, q);
        return p;
    }
    else {
        q->left = merge(p, q->left);
        return q;
    }
}

node* remove(node* p, int k){ // удаление из дерева p первого
найденного узла
    if( !p )
        return p;
    if( p->key == k ) {
        node* q = merge(p->left,p->right);
```

```
        delete p;
        return q;
    }
    else if( k < p->key )
        p->left = remove(p->left,k);
    else
        p->right = remove(p->right,k);
    return p;
}

void printPriority(node* root,QString &tr){
   if (!root)
       return;
   char st[10],tm[10];
   tr = tr + "Key Priority [" + itoa(root->key,st,10) + "] - " +
itoa(root->prior,tm,10) + "\n";
   printPriority(root->right,tr);
   printPriority(root->left,tr);
}

node* findKey( node* tree, int key){
    if(!tree)
        return NULL;
    if(key == tree->key)
        return tree;
    if(key < tree->key)
        return findKey(tree->left, key);
    else
        return findKey(tree->right, key);
}

node* MainWindow::add(node* p, int el,QString &err){ // добавление
нового элемент
    if (findKey(p,el)) {
        char st[10];
        err = err + "Key [" + itoa(el,st,10) + "] repeats" + "\n";
        return p;
    }
    else{
        p=insert(el, p);
        char st[10],tm[10];
        err = err + "Priority of a new key [" + itoa((findKey(p,el))-
>key,st,10) + "] - " + itoa((findKey(p,el))->prior,tm,10) + "\n";
        return p;
```

```
    }
}

void printtree(node* treenode, int l,QString &out){
    if(treenode==NULL) {
        for(int i = 0; i<l; ++i)
            out = out + "  ";
        return;
    }
    printtree(treenode->right, l+1,out);
    for(int i = 0; i < l; i++)
        out = out + "\t";
    char st[10];
    out = out + itoa(treenode->key,st,10) + "\n";
    printtree(treenode->left,l+1,out);
}

void MainWindow::Calculate(){
        delete[] treap;
        treap = NULL;
        int c;
        if (Qstr.isEmpty()){
           QMessageBox::warning(this, "Ошибка", "String is Empty!");
           return;
        }
        string str = Qstr.toUtf8().constData();
        QString err;
        Qstr.clear();
        char* arr = new char[str.size()+1];
        strcpy(arr, str.c_str()); // запись строки в массив, который
содержит последовательность символов с нулевым завершением
        char* tok;
        tok = strtok(arr, " "); // разделяем строку на цифры - ключи
        while(tok != NULL) {
           c = atoi(tok);      // конвертируем строку в величину типа
int
           if(isalpha(*tok)){
               QMessageBox::warning(this, "Ошибка", "Uncorrect
Data!");
                return;
           }
           if (findKey(treap,c) != NULL){
                 char st[10];
```

```
                err = err + "Key [" + itoa(c,st,10) + "] repeats\n";
// повторение ключа не допустимо, тк должен быть уникальным
                QMessageBox::warning(this, "Ошибка", err);
                //ui->ErrorEdit->setText(err);
                tok = strtok(NULL, " ");
                continue;
              }
           treap = insert(c, treap);
           tok = strtok(NULL, " ");
        }
        QString tr = "";
        printPriority(treap,tr); // печать приоритетов
        ui->treeEdit->setText(tr);
        QString out = "";
        printtree(treap,0,out);
        ui->treapEdit->setText(out);
        delete tok;
        delete[] arr;
   }

void MainWindow::addKey(){
    if (treap == NULL){
        QMessageBox::warning(this, "Ошибка", "Treap is NULL!");
        return;
    }
    QString tmp = ui->addEdit->toPlainText();
    int flag=0;
    for (int i=0; i<tmp.size();i++){
    if (isalpha(tmp.toStdString()[i])){
        flag++;
    }
    }
    if (!flag) {
    int el = atoi(tmp.toStdString().data());
    QString err;
    treap = add(treap, el,err);
    QString out = "";
    printtree(treap,0,out);
    QString tr = "";
    printPriority(treap,tr); // печать приоритетов
    ui->treeEdit->setText(tr);
    ui->treapEdit->setText(out);
    }
    else QMessageBox::warning(this, "Ошибка", "Not a number!");
```

```
}
void MainWindow::removeKey(){
    if (treap == NULL){
        ui->treapEdit->setText("Treap is NULL");
        return;
    }
    QString tmp = ui->removeEdit->toPlainText();
    int el = atoi(tmp.toStdString().data());
    treap = remove(treap, el);
    QString out = "";
    printtree(treap,0,out);
    ui->treapEdit->setText(out);
}
```