

**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: Случайные БДП с рандомизацией**

Студентка гр. 9381

\_\_\_\_\_

Москаленко Е.М.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

Санкт-Петербург

2020

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студентка Москаленко Е.М.

Группа 9381

Тема работы:

Вариант 10. Случайные БДП с рандомизацией. Демонстрация

Исходные данные:

Целые числа для добавления в БДП

Содержание пояснительной записки:

«Содержание», «Введение», «Задание», «Описание алгоритма», «Описание структур данных и функций», «Описание интерфейса пользователя», «Тестирование», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 14.12.2020

Дата защиты реферата: 16.12.2020

Студентка

\_\_\_\_\_

Москаленко Е.М.

Преподаватель

\_\_\_\_\_

Фирсов М.А.

## **АННОТАЦИЯ**

В данной курсовой работе на языке программирования C++ реализована такая структура данных, как "случайное бинарное дерево с рандомизацией". Вставка и удаление элементов дерева продемонстрированы с помощью средств фреймворка Qt.

## **SUMMARY**

In this course work, in the C ++ programming language, such a data structure as a "random binary tree with randomization" is implemented. Inserting and deleting tree elements is demonstrated using the tools of the Qt framework.

## СОДЕРЖАНИЕ

Введение	5
1.Задание	6
2.Описание алгоритма	6
2.1.Структура данных	6
2.2. Ввод данных	6
2.3. Добавление элементов в дерево	6
2.4. Удаление элемента из дерева	7
3. Описание структур данных и функций	7
3.1. Структура Node	7
3.2. Класс BinTree	8
3.3. Класс MainWindow	10
4. Описание интерфейса пользователя	12
5. Тестирование	13
Заключение	16
Список использованных источников	17
Приложение А. Исходный код программы	18

## ВВЕДЕНИЕ

Целью работы является написание программы, реализующей случайное БДП с рандомизацией. Для этого необходимо изучить соответствующую структуру данных и операции вставки и удаления в ней.

Ключевая идея случайных БДП с рандомизацией состоит в чередовании обычной вставки в дерево поиска и вставки в корень. Чередование происходит случайным (рандомизированным) образом с использованием компьютерного генератора псевдослучайных чисел. Цель такого чередования – сохранить хорошие свойства случайного БДП в среднем и исключить (сделать маловероятным) появление «худшего случая» (поддеревьев большой высоты).

Операция вставки в корень. Если дерево пусто, необходимо создать новый узел со значением  $key$ , если  $key(tree) > key$ , выполнить вставку в корень в левом поддереве  $tree$  и правое вращение, иначе — вставку в корень в правом поддереве и левое вращение. Таким образом узел со значением  $key$  становится корнем дерева.

Рандомизированная вставка элемента со значением  $key$  в дерево  $tree$ . Пусть в дереве имеется  $n$  узлов. После добавления еще одного узла любой узел с равной вероятностью может быть корнем дерева. Тогда, с вероятностью  $1/(n+1)$  осуществляется вставка в корень, иначе рекурсивно используется рандомизированная вставка в левое или правое поддерево в зависимости от значения ключа  $key$ .

## 1. ЗАДАНИЕ

Реализовать структуру данных «Случайное бинарное дерево поиска с рандомизацией» на языке программирования C++. Создать графический интерфейс для демонстрации операций вставки и удаления элементов дерева. На каждом шаге визуализации бинарного дерева программа должна выводить текстовые пояснения того, что происходит.

## 2. ОПИСАНИЕ АЛГОРИТМА

### 2.1. Структура данных.

Для реализации случайного бинарного дерева поиска были созданы структура одного элемента дерева и класс всего дерева, полем которого является корень.

### 2.2. Ввод данных.

Пользователю предлагается выбрать способ ввода данных: ввести данные в строку элемента графического интерфейса QLineEdit или считать файла. В зависимости от выбора, вызывается функция `checkStr` (проверка строки на корректность) или `fillBdpFile` (считывание с файла). Обе функции вызывают `fillBdp`, в которой происходит вызов функций заполнения дерева и его визуализации на QGraphicsScene.

### 2.3. Добавление элементов в дерево.

Дерево рекурсивно заполняется элементами с помощью метода рандомизированной вставки `insert` класса `BinTree`. Рандомизация происходит с помощью функции `rand()` стандартной библиотеки C++. С вероятностью  $1/n+1$ , где  $n$  – размер дерева до вставки, произойдет вставка элемента в корень дерева, и будет вызван метод `insertRoot`, использующий левое или правое вращение. Иначе с вероятностью  $1-1/(n+1)$  произойдет рекурсивная вставка в правое или левое поддерево в зависимости от значения ключа в корне.

### 2.4. Удаление элемента из дерева.

Для удаления элемента в дереве необходимо соединять два дерева. Любой ключ первого дерева меньше любого ключа во втором дереве. В качестве корня нового соединенного дерева можно взять любой из двух корней, пусть это будет  $p$ . Тогда левое поддерево  $p$  можно оставить как есть, а справа к  $p$  подвесить объединение двух деревьев — правого поддерева  $p$  и всего дерева  $q$ .

С тем же успехом мы можем сделать корнем нового дерева узел  $q$ . В рандомизированной реализации выбор между этими альтернативами делается случайным образом. Пусть размер левого дерева равен  $n$ , правого —  $m$ . Тогда  $p$  выбирается новым корнем с вероятностью  $n/(n+m)$ , а  $q$  — с вероятностью  $m/(n+m)$ .

Удаление элемента происходит по ключу — программа ищет узел с заданным ключом и удаляет этот узел из дерева. Стадия поиска такая же как и при поиске и добавлении нового элемента, а дальше программа объединяет левое и правое поддерева найденного узла, удаляет узел и возвращает корень объединенного дерева.

### 3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

#### 3.1. Структура Node.

Для хранения одного элемента реализована структура Node со следующими полями:

int key; - значение элемента узла

int size; - размер дерева с корнем в данном узле

int amount; - количество попыток ввести элемент со значением данного узла

Node \*left; - левое поддерево

Node \*right; - правое поддерево

При создании структуры:

key = k (переданное значение)

```
size = 1;  
amount = 1;  
left = right = nullptr;
```

### 3.2. Класс BinTree.

Для реализации бинарного дерева поиска создан класс BinTree.

Поле класса BinTree:

**Node\* head** – вершина дерева

Методы класса Node:

1) **BinTree::BinTree()** – конструктор класса BinTree

*head* в начале = nullptr.

2) **int getSize(Node\* p)** – возвращает размер дерева, корнем которого является переданный в аргументах узел

*Node \*p* – указатель на корень дерева

3) **void fixSize(Node\* p)** – метод, корректирующий размер дерева (размер левого поддерева + размер правого поддерева + 1)

*Node \*p* – указатель на корень дерева

4) **Node\* rotateLeft(Node\* p)** – метод левого вращения дерева, вызывается при вставке элемента в корень (из правого поддерева). Возвращает указатель на корень измененного дерева.

*Node \*p* – указатель на корень дерева

5) **Node\* rotateRight(Node\* p)** – метод правого вращения дерева, вызывается при вставке элемента в корень (из левого поддерева). Возвращает указатель на корень измененного дерева.

*Node \*p* – указатель на корень дерева



6) ***Node\* insert(Node\* p, int k, string\* str)*** – выполняет с вероятностью  $1/(n+1)$ , где  $n$  – размер дерева в узлах, вставку в корень, а с вероятностью  $1-1/(n+1)$  — рекурсивную вставку в правое или левое поддерево в зависимости от значения ключа в корне. Возвращает указатель на корень измененного дерева.

*Node \*p* – указатель на корень дерева

*int k* - значение вставляемого элемента

*string\* str* – строка с промежуточными выводами

7) ***Node\* insertRoot(Node\* p, int k, , string\* str)*** – вставка в корень дерева. Сначала рекурсивно вставляется новый ключ в корень левого или правого поддеревьев (в зависимости от результата сравнения с корневым ключом) и выполняется правый (левый) поворот, который поднимает нужный узел в корень дерева.

*Node \*p* – указатель на корень дерева

*int k* - значение вставляемого элемента

*string\* str* – строка с промежуточными выводами

8) ***void print2DUtil(Node \*root, int space, string\* str)*** – запись «лежащего» дерева в строку.

*Node \*root* – указатель на корень дерева

*int space* – количество пробелов (расстояние) между уровнями

*string\* str* – строка для записи

9) ***int maxDepth(Node \*p)*** – возвращает максимальную глубину дерева.

*Node \*p* – указатель на корень дерева

10) ***Node\* join(Node\* p, Node\* q)*** – объединяет два бинарных дерева в одно.

*Node \*p* – указатель на корень первого дерева

*Node \*q* – указатель на корень второго дерева

**11)        *Node\* remove(Node\* p, int k, string\* str)*** – метод удаления элемента из дерева. Возвращает указатель на корень обновленного дерева.

*Node \*p* – указатель на корень дерева

*int k* - значение вставляемого элемента

*string\* str* – строка с промежуточными выводами

**12)        *~BinTree()*** – деструктор класса BinTree. Вызывает метод *destroy*.

**13)        *void destroy(Node\* p)*** – рекурсивный метод удаления дерева.

*Node \*p* – указатель на корень дерева

### 3.3. Класс MainWindow.

Класс, контролирующий окно приложения, нажатие на кнопки, визуализацию дерева и демонстрацию промежуточных выводов.

*Поля класса MainWindow:*

***BinTree\* tree*** – объект бинарного дерева.

***QGraphicsScene\* scene*** – графическая сцена для отрисовки дерева.

*Обработка слотов кнопок:*

**1) *void on\_printTree\_clicked()*** – обработка нажатия на кнопку добавления элемента(ов)

2) *void on\_deleteElement\_clicked()* – обработка нажатия на кнопку удаления элемента

3) *void on\_newTree\_clicked()* – обработка нажатия на кнопку создания нового дерева

4) *void on\_openFile\_clicked()* – обработка нажатия на кнопку выбора файла для считывания данных

Методы класса *MainWindow*:

1) *vector<int> checkStr()* – проверка строки из поля ввода на корректность и создание вектора с числами для заполнения дерева.

2) *Node\* fillBdp(vector<int> arr)* – заполнение дерева данными из вектора целых чисел посредством вызова метода *insert* класса *Bintree*

*vector<int> arr* – вектор целых чисел для заполнения дерева

3) *Node\* fillMas(BinTree\* tree, string name, int count)* - одна из двух функций рекурсивного заполнения БДП при считывании с файла. Создается целочисленный массив для *count* элементов, затем в него считываются числа из файла, а после этого все значения из массива *x* с помощью метода *fillBdp* добавляются в дерево. Функция возвращает указатель на корень измененного дерева.

*Node \*p* – указатель на корень дерева

*string name* – полный путь до файла

*int count* – количество чисел в строке

4) *Node\* fillBdpFile(BinTree\* tree, string name)* - одна из двух функций рекурсивного заполнения БДП при считывании с файла. Если файла с

переданным именем не существует, то возвращается пустое дерево. Иначе в файле посчитывается количество чисел в строке, и возвращается значение функции *fillBdpFile(Node\* p, string name)*.

*Node \*p* – указатель на корень дерева

*string name* – полный путь до файла

Функции отрисовки дерева:

1) ***int treePainter(QGraphicsScene \*scene, Node\* node, int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth, int colour)*** – рекурсивный алгоритм обхода дерева и рисования узлов в заданном объекте.

2) ***QGraphicsScene \*graphic(Node \*tree, QGraphicsScene \*scene, int depth)*** – рисование в объекте QGraphicsScene по заданному бинарному дереву.

#### 4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Для взаимодействия пользователя с программой был разработан графический интерфейс.

Добавлены две строки QLineEdit для ввода добавляемых и удаляемых элементов.

*Кнопки:*

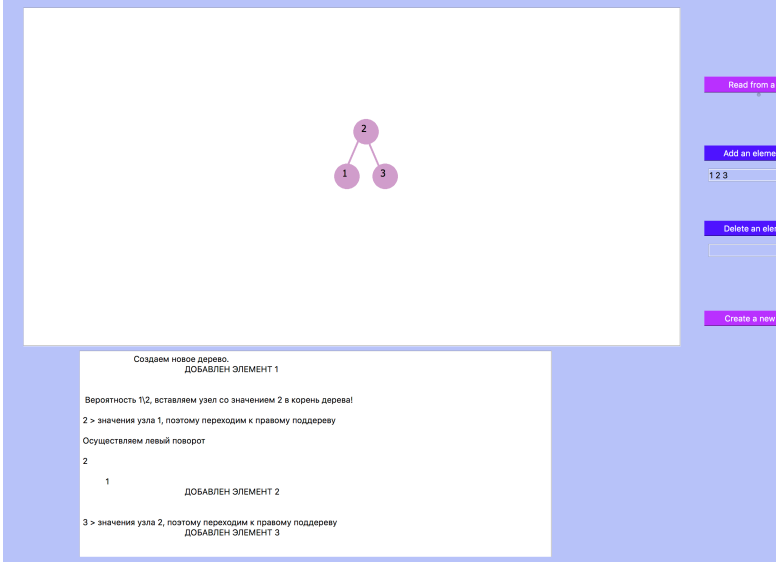
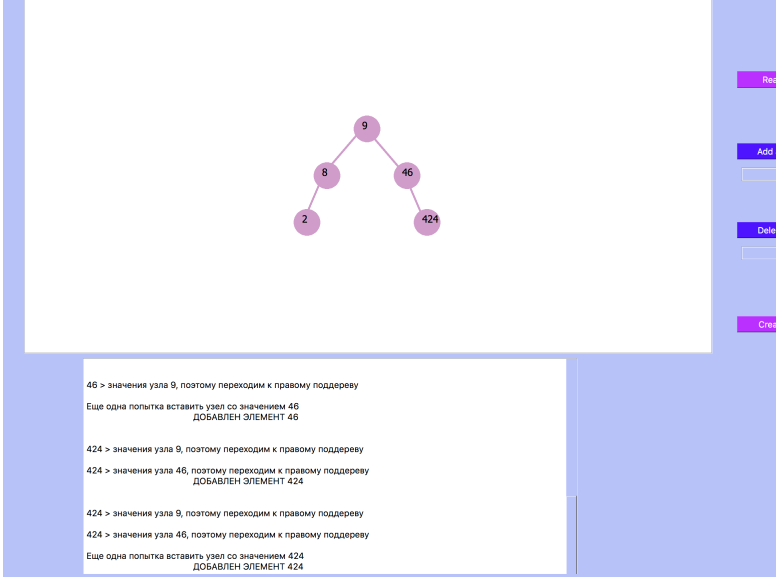
***Read from a file*** – пользователь может выбрать файл для считывания данных

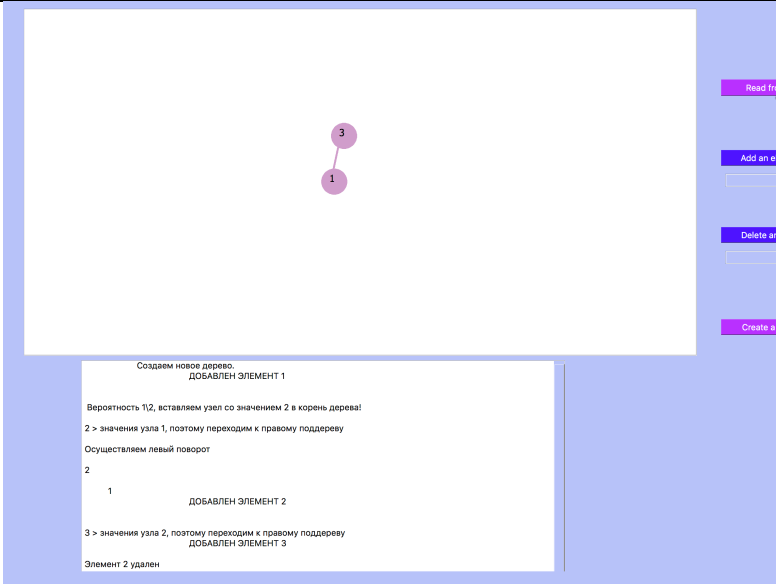
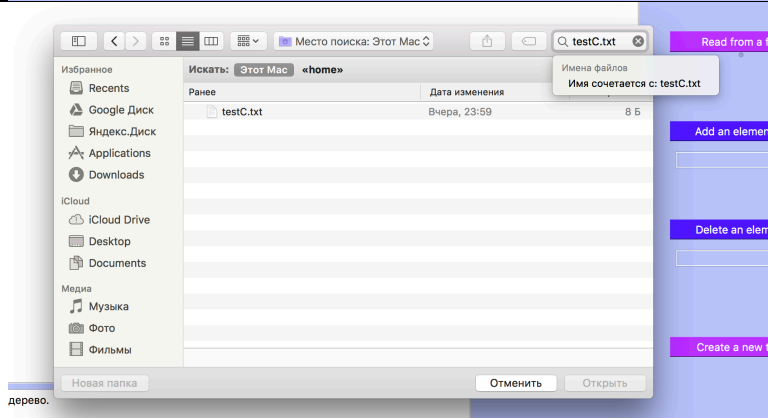
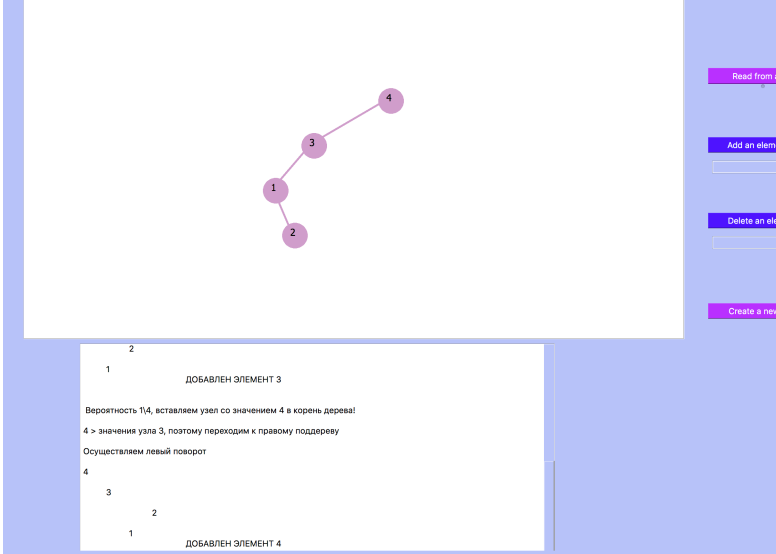
***Add an element(s)*** – добавление в дерево введенных пользователем данных в QLineEdit

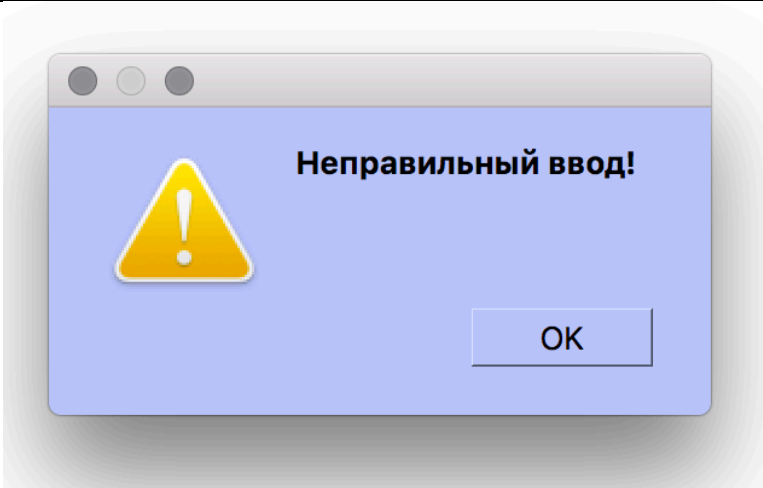
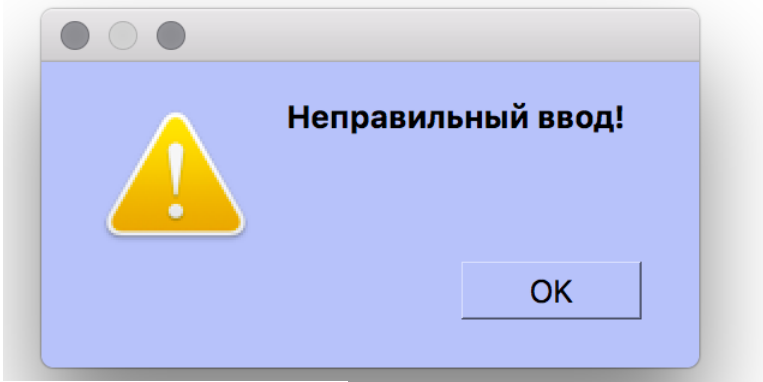
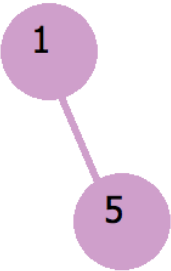
***Delete an element*** – удаление элемента, введенного пользователем в QLineEdit

*Create a new tree* – удаление старого дерева и создание нового. Происходит очистка сцены.

## 5. ТЕСТИРОВАНИЕ

№	Входные данные	Результат	Комментарий
1	1 2 3		На скриншоте продемонстрировано создание дерева с элементами 1 2 3
2	2 8 9 46 424		На скриншоте продемонстрировано создание дерева с элементами 2 8 9 46 424

3	2 (удалить)	 <p>Создаем новое дерево. ДОБАВЛЕН ЭЛЕМЕНТ 1</p> <p>Вероятность 1/2, вставляем узел со значением 2 в корень дерева! 2 &gt; значения узла 1, поэтому переходим к правому поддереву Осуществляем левый поворот</p> <pre> 2  1     ДОБАВЛЕН ЭЛЕМЕНТ 2 3 &gt; значения узла 2, поэтому переходим к правому поддереву     ДОБАВЛЕН ЭЛЕМЕНТ 3 Элемент 2 удален </pre>	На скриншоте продемонстрировано удаление элемента 2 из теста №1
4	testC.txt		Открытие файла testC.txt для считывания данных
5	1 2 3 4	 <pre> 2  1     ДОБАВЛЕН ЭЛЕМЕНТ 3 Вероятность 1/4, вставляем узел со значением 4 в корень дерева! 4 &gt; значения узла 3, поэтому переходим к правому поддереву Осуществляем левый поворот 4  3     2       1         ДОБАВЛЕН ЭЛЕМЕНТ 4 </pre>	Дерево, построенное по данным из файла testC.txt

6	рва		Обработка некорректных данных
7	Дерево было: 1 Ввод: 5лыа	 	Обработка некорректных данных + считывания числа из некорректно введенной строки.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения курсовой работы была написана программа на языке программирования C++, реализующая случайное бинарное дерево поиска с рандомизацией. Реализованы функции добавления и удаления элементов дерева. Для наглядной демонстрации работы алгоритма был написан графический интерфейс в фреймворке Qt.



## **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Рандомизированные деревья поиска // URL: <https://habr.com/ru/post/145388/> (дата обращения: 08.12.2020).

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД ПРОГРАММЫ

#### Файл main.cpp

```
#include "mainwindow.h"

#include <QApplication>

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

#### Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include "node.h"
#include <QMainWindow>
#include <QMessageBox>
#include <QFileDialog>
#include <QTimer>

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_printTree_clicked();

    void on_deleteElement_clicked();

    void on_newTree_clicked();

    void on_openFile_clicked();

private:
    BinTree* tree;
    vector<int> checkStr();
    Node* fillBdp(vector<int> arr);
    Node* fillMas(BinTree* tree, string name, int count);
    Node* fillBdpFile(BinTree* tree, string name);
    QGraphicsScene* scene;
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

## Файл node.h

```
#ifndef NODE_H
#define NODE_H
#include <iostream>
#include <vector>
#include <sstream>
#include <QGraphicsScene>
using namespace std;
#define COUNT 10

struct Node {
    int key;    // значение элемента узла
    int size;   //размер дерева с корнем в данном узле
    int amount; //количество попыток ввести элемент со значением данного
    узла
    Node *left; //левое поддерево
    Node *right; // правое поддерево
    Node(int k) { key = k; size = 1; amount = 1; left = right = nullptr; }
};

class BinTree{
public:
    Node* head;
    BinTree();
    ~BinTree();
    int find(Node *p, int k);
    int getSize(Node* p);
    int maxDepth(Node *p);
    void fixSize(Node* p);
    Node* rotateLeft(Node* p);
    Node* rotateRight(Node* p);
    Node* insert(Node* p, int k, string* str);
    Node* insertRoot(Node* p, int k, string* str);
    Node* getLeft();
    Node* getRight();
    Node* join(Node* p, Node* q);
    Node* remove(Node* p, int k, string* str);
    void print2DUtil(Node *root, int space, string* str);
    void recTreePrint(Node* p);
    void destroy(Node* p);
};
#endif // NODE_H
```

## Файл drawTree.cpp

```
#include "node.h"
#include <QGraphicsTextItem>
#include <cmath>

int treePainter(QGraphicsScene *scene, Node* node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth, int
colour);

QGraphicsScene *graphic(Node *tree, QGraphicsScene *scene, int depth)
{
```

```

    if (!tree)
        return scene;
    scene->clear();
    QPen pen;
    QColor color;
    color.setRgb(220, 220, 220);
    pen.setColor(color);
    int color_c = 220;
    color.setRgb(220, 220, 220);
    QBrush brush (color);
    QFont font;
    font.setFamily("Tahoma");
    pen.setWidth(3);
    int wDeep = static_cast<int>(pow(2, depth));
    int hDelta = 70;
    int wDelta = 15;
    font.setPointSize(wDelta);
    int width = (wDelta*wDeep)/2;
    treePainter(scene, tree, width/2, hDelta, wDelta, hDelta, pen, brush,
font, wDeep, color_c);
    return scene;
}

int treePainter(QGraphicsScene *scene, Node *node, int w, int h, int
wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int depth, int
colour)
{
    if (node == nullptr)
        return 0;
    QString out;
    out.append(QString::number(node->key));
    QColor color(200, 162, 200);
    brush.setColor(color);
    pen.setColor(color);
    QGraphicsTextItem *textItem = new QGraphicsTextItem;
    if (node->left != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    if (node->right != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    textItem->setPos(w, h);
    textItem->setPlainText(out);
    textItem->setFont(font);
    scene->addEllipse(w-wDelta/2, h, wDelta*5/2, wDelta*5/2, pen, brush);

    scene->addItem(textItem);
    treePainter(scene, node->left, w-(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2, colour - 30);
    treePainter(scene, node->right, w+(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2, colour+ 50);
    return 0;
}

```

### Файл mainwindow.cpp

```

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

```

```

#include <fstream>
#include <QDesktopWidget>
#include <QGraphicsScene>
#include <QValidator>
#include "drawTree.cpp"
#include <QThread>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    scene = new QGraphicsScene();
    ui->graphicsView->setScene(scene);
    ui->deleteInput->setValidator (new QIntValidator (this));
    tree = new BinTree();
}

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

Node* MainWindow::fillBdp(vector<int> arr){
    for(int val : arr) { //добавляем элементы из вектора в БДП
        string k = "";
        tree->head = tree->insert(tree->head, val, &k);
        k += "\t\tДОБАВЛЕН ЭЛЕМЕНТ " + std::to_string(val) + "\n";
        ui->result->append(QString::fromLocal8Bit(k.c_str()));
        int depth = tree->maxDepth(tree->head);
        scene = graphic(tree->head, scene, depth);
        QEventLoop loop;
        QTimer::singleShot(1700, &loop, SLOT(quit()));
        loop.exec();
    }
    return tree->head;
}

void MainWindow::on_printTree_clicked()
{
    scene->clear();
    vector<int> arr = checkStr();
    tree->head = fillBdp(arr);
    ui->inputTree->clear();
}

vector<int> MainWindow::checkStr(){
    QString data = ui->inputTree->text();
    string str = data.toUtf8().constData();
    istringstream ss(str);
    vector<int>arr; //вектор введенных чисел
    int x;
    while(ss >> x){
        arr.push_back(x);
        if(ss.peek() == ' ') //игнорируем пробелы
            ss.ignore();
    }
}

```

```

        if (!ss.eof())    //если были введены символы кроме цифер
        {
            QMessageBox::warning(this, "Error", "Неправильный ввод!");
        }
        return arr;
    }

void MainWindow::on_deleteElement_clicked()
{
    if (ui->deleteInput->text().isEmpty())
        return;
    QString data = ui->deleteInput->text();
    string str = data.toUtf8().constData();
    for (auto i: str)
    {
        if (!isdigit(i)){
            QMessageBox::warning(this, "Error", "Неправильный ввод!");
            return;
        }
    }
    int x = std::stoi(str);
    string k = "";
    tree->head = tree->remove(tree->head, x, &k);
    ui->deleteInput->clear();
    int depth = tree->maxDepth(tree->head);
    ui->result->append(QString::fromLocal8Bit(k.c_str()));
    scene = graphic(tree->head, scene, depth-1);
    if (!tree->head)
        scene->clear();
}

void MainWindow::on_newTree_clicked()
{
    scene->clear();
    delete tree;
    tree = new BinTree();
    ui->result->clear();
    ui->result->append(QString::fromLocal8Bit("\tСоздаем новое
дерево."));
}

void MainWindow::on_openFile_clicked()
{
    QString str = QFileDialog::getOpenFileName(nullptr, "Выберите файл
для открытия", "/home/user", "*.txt");
    if (str == nullptr)
        return;
    tree->head = fillBdpFile(tree,
str.toLocal8Bit().constData());
}

Node* MainWindow::fillMas(BinTree* tree, string name, int count){ //
заполнение БДП с файла
    ifstream in(name); //открываем файл
    if (!in.is_open()) {

```

```

        cout << "Файл не может быть открыт!\n";
        return tree->head;
    }

    int x[count];
    vector<int>arr;
    for (int i = 0; i < count - 1; i++){
        in >> x[i];
        arr.push_back(x[i]);
    }

    tree->head = fillBdp(arr);
    in.close(); //под конец закроем файла
    return tree->head;
}

Node* MainWindow::fillBdpFile(BinTree* tree, string name){ //подсчет
количества чисел в файле
    //Создаем файловый поток и связываем его с файлом
    ifstream in(name);
    if (!in.is_open()) {
        cout << "Файл не может быть открыт!\n";
        return tree->head;
    }

    int count = 0; // количество чисел в файле
    int temp; //Временная переменная

    while (!in.eof()) // пробегаем пока не встретим конец файла eof
    {
        in >> temp; //в пустоту считываем из файла числа
        count++; // увеличиваем счетчик количества чисел
    }

    in.close();
    return fillMas(tree, name, count);
}

```

### Файл node.cpp

```

#include "node.h"

Node* BinTree::getLeft(){ //возвращает левое поддереву
    return head->left;
}

Node* BinTree::getRight(){ //возвращает правое поддереву
    return head->right;
}

BinTree::BinTree(){
    head = nullptr;
}

int BinTree::find(Node* p, int k) // поиск ключа k в дереве p
{
    if(!p)
        return false; // в пустом дереве можно не искать
    if( k == p->key ) {

```

```

        cout << "\n\033[31m  Найден узел с совпадающим
значением!\033[0m\n";
        return p->amount;          //если нашли, возвращаем поле amount -
количество вхождений
    }
    if( k < p->key ) { //если меньше, ищем в левом поддереве
        cout << "\nЗначение искомого узла меньше " << p->key << ",
переходим к левому поддереву\n";
        return find(p->left, k);
    }
    else { //если больше, ищем в правом поддереве
        cout << "\nЗначение искомого узла больше " << p->key << ",
переходим к правому поддереву\n";
        return find(p->right, k);
    }
}

int BinTree::getSize(Node* p) // возвращает размер дерева
{
    return (p)? p->size : 0;
}

void BinTree::fixSize(Node* p) // установление корректного размера дерева
{
    p->size = getSize(p->left) + getSize(p->right) + 1; //размер =
размер левого поддерева + правого + 1
}

Node* BinTree::rotateLeft(Node* p) // левый поворот вокруг узла p
{
    if (!p || !p->right) //если узла нет, или правого поддерева, то
поворот не происходит
        return p;

    Node* q = p->right;
    p->right = q->left;
    q->left = p;
    q->size = p->size;
    fixSize(p);
    fixSize(q);
    return q;
}

Node* BinTree::rotateRight(Node* p) // правый поворот вокруг узла p
{
    if (!p || !p->left) //если узла нет, или левого поддерева, то
поворот не происходит
        return p;

    Node* q = p->left;
    p->left = q->right;
    q->right = p;
    q->size = p->size;
    fixSize(p);
    fixSize(q);
    return q;
}

```



```

Node* BinTree::insertRoot(Node* p, int k, string* str) // вставка нового
узла с ключом k в корень дерева p
{
    if(!p) {
        return new Node(k);
    }

    if( k < p->key )          // если значение k < значения узла, то
переходим в левое поддерево
    {
        *str += "\n" + std::to_string(k) + " < "
            + "значения узла " + std::to_string(p->key) + ", поэтому
переходим к левому поддереву\n";
        p->left = insertRoot(p->left, k, str);
        *str += "\nОсуществляем правый поворот\n";
        p = rotateRight(p);    //осуществляем правый поворот
        print2DUtil(p, 0, str);
    }

    else if (k > p->key)       // если значение k > значения узла, то
переходим в правое поддерево
    {
        *str += "\n" + std::to_string(k) + " > "
            + "значения узла " + std::to_string(p->key) + ", поэтому
переходим к правому поддереву\n";
        p->right = insertRoot(p->right, k, str);
        *str += "\nОсуществляем левый поворот\n";
        p = rotateLeft(p);    //осуществляем левый поворот
        print2DUtil(p, 0, str);
    }
    return p;
}

Node* BinTree::insert(Node* p, int k, string* str) // рандомизированная
вставка нового узла с ключом k в дерево p
{
    if(!p){
        return new Node(k); //если корня дерева не существует, то
создаем его в конструкторе со значением k
    }

    if(k == p->key){//если узел с таким значением уже есть в дереве,
увеличиваем значение поля amount
        *str += "\nЕще одна попытка вставить узел со значением " +
std::to_string(k) + "\n";
        p->amount++;
        return p;
    }

    // *str += "\nВероятность элемента " + std::to_string(k) + " оказаться
в корне: " + std::to_string(r) + "/" + std::to_string(p->size+1) + "\n";
    if(rand() % (p->size+1) == 0) { // вставка в корень происходит с
вероятностью 1/(n+1),
        // где n - размер дерева до вставки
        *str += "\n Вероятность " + std::to_string(1) + "\\\" +
std::to_string(p->size+1) +

```

```

        ", вставляем узел со значением " + std::to_string(k) + "
в корень дерева!\n";
        return
            insertRoot(p, k, str);
    }

    if(p->key > k) { // иначе происходит обычная вставка в правое или
левое поддерево в зависимости от значения k
        *str += "\n" + std::to_string(k) + " < " + "значения узла "
            + std::to_string(p->key) + ", поэтому переходим к левому
поддереву\n";
        p->left = insert(p->left, k, str);
    }
    else {
        *str += "\n" + std::to_string(k) + " > " + "значения узла "
            + std::to_string(p->key) + ", поэтому переходим к правому
поддереву\n";
        p->right = insert(p->right, k, str);
    }
    fixSize(p); //регулируется размер дерева
    return p;
}

void BinTree::print2DUtil(Node *root, int space, string* str) //печать
лежащего 2D дерева
{
    if (!root)
        return;

    space += COUNT; //счетчик дистанции между уровнями
    print2DUtil(root->right, space, str); //обработка правого поддерева
    *str += "\n";
    for (int i = COUNT; i < space; i++)
        *str += " ";
    *str += std::to_string(root->key) + "\n"; //печать значения узла
    print2DUtil(root->left, space, str); //обработка левого поддерева
}

int BinTree::maxDepth(Node *p){
    if (!p)
        return 0;
    else{
        int lDepth = maxDepth(p->left);
        int rDepth = maxDepth(p->right);
        if (lDepth > rDepth)
            return(lDepth + 1);
        else return(rDepth + 1);
    }
}

Node* BinTree::join(Node* p, Node* q) // объединение двух деревьев
{
    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;
    }
}

Node* BinTree::remove(Node* p, int k, string* str) // удаление из дерева
p первого найденного узла с ключом k
{
    if(!p)
        return p;
    if( p->key == k )
    {
        *str += "Элемент " + std::to_string(k) + " удален\n";
        Node* q = join(p->left, p->right);
        delete p;
        return q;
    }
    else if( k < p->key ){
        *str += std::to_string(k) + " < " + "значения узла "
            + std::to_string(p->key) + ", поэтому переходим к левому
поддереву\n";
        p->left = remove(p->left,k, str);
    }
    else{
        *str += "\n" + std::to_string(k) + " > " + "значения узла "
            + std::to_string(p->key) + ", поэтому переходим к правому
поддереву\n";
        p->right = remove(p->right, k, str);
    }
    return p;
}

BinTree::~BinTree(){
    destroy(head);
}

void BinTree::destroy(Node* p)    //удаление дерева
{
    if(!p)
        return;
    destroy(p->left);
    destroy(p->right);
    delete p;
}

```

## Файлmainwindow.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>1283</width>
      <height>966</height>
    </rect>
  </property>
  <property name="windowTitle">
    <string>MainWindow</string>
  </property>
  <property name="autoFillBackground">
    <bool>>false</bool>
  </property>
  <property name="styleSheet">
    <string notr="true">background-color:rgb(185, 195, 246)</string>
  </property>
  <widget class="QWidget" name="centralwidget">
    <widget class="QGraphicsView" name="graphicsView">
      <property name="geometry">
        <rect>
          <x>32</x>
          <y>12</y>
          <width>1031</width>
          <height>531</height>
        </rect>
      </property>
      <property name="styleSheet">
        <string notr="true">background-color:rgb(255, 255, 255);</string>
      </property>
      <property name="verticalScrollBarPolicy">
        <enum>Qt::ScrollBarAsNeeded</enum>
      </property>
    </widget>
    <widget class="QTextBrowser" name="result">
      <property name="geometry">
        <rect>
          <x>120</x>
          <y>550</y>
          <width>741</width>
          <height>341</height>
        </rect>
      </property>
      <property name="styleSheet">
        <string notr="true">background-color:rgb(255, 255, 255)</string>
      </property>
    </widget>
    <widget class="QSplitter" name="splitter">
      <property name="geometry">
        <rect>
          <x>1100</x>
          <y>120</y>
          <width>171</width>
          <height>391</height>
        </rect>
      </property>
    </widget>
  </widget>
</widget>

```

```

    <property name="orientation">
      <enum>Qt::Vertical</enum>
    </property>
    <widget class="QPushButton" name="openFile">
      <property name="styleSheet">
        <string notr="true">background-color:rgb(172, 70, 255);
color:white</string>
      </property>
      <property name="text">
        <string>Read from a file</string>
      </property>
    </widget>
    <widget class="QWidget" name="layoutWidget">
      <layout class="QVBoxLayout" name="verticalLayout">
        <item>
          <spacer name="verticalSpacer">
            <property name="orientation">
              <enum>Qt::Vertical</enum>
            </property>
            <property name="sizeHint" stdset="0">
              <size>
                <width>20</width>
                <height>80</height>
              </size>
            </property>
          </spacer>
        </item>
        <item>
          <widget class="QPushButton" name="printTree">
            <property name="styleSheet">
              <string notr="true">background-color:rgb(74, 48, 255);
color:white</string>
            </property>
            <property name="text">
              <string>Add an element(s)</string>
            </property>
          </widget>
        </item>
        <item>
          <widget class="QLineEdit" name="inputTree">
            <property name="cursorMoveStyle">
              <enum>Qt::LogicalMoveStyle</enum>
            </property>
          </widget>
        </item>
        <item>
          <spacer name="verticalSpacer_3">
            <property name="orientation">
              <enum>Qt::Vertical</enum>
            </property>
            <property name="sizeHint" stdset="0">
              <size>
                <width>20</width>
                <height>40</height>
              </size>
            </property>
          </spacer>
        </item>
      </layout>
    </widget>
  </property>
</object>

```

```

</item>
<item>
  <widget class="QPushButton" name="deleteElement">
    <property name="styleSheet">
      <string notr="true">background-color:rgb(74, 48, 255);
color:white</string>
    </property>
    <property name="text">
      <string>Delete an element</string>
    </property>
  </widget>
</item>
<item>
  <widget class="QLineEdit" name="deleteInput"/>
</item>
<item>
  <layout class="QVBoxLayout" name="verticalLayout_2"/>
</item>
<item>
  <spacer name="verticalSpacer_2">
    <property name="orientation">
      <enum>Qt::Vertical</enum>
    </property>
    <property name="sizeHint" stdset="0">
      <size>
        <width>20</width>
        <height>80</height>
      </size>
    </property>
  </spacer>
</item>
<item>
  <widget class="QPushButton" name="newTree">
    <property name="styleSheet">
      <string notr="true">background-color:rgb(172, 70, 255);
color:white</string>
    </property>
    <property name="text">
      <string>Create a new tree</string>
    </property>
  </widget>
</item>
</layout>
</widget>
</widget>
<widget class="QMenuBar" name="menubar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>1283</width>
      <height>22</height>
    </rect>
  </property>
</widget>
<widget class="QStatusBar" name="statusbar"/>

```

```
</widget>  
<resources/>  
<connections/>  
</ui>
```