

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

КУРСОВАЯ РАБОТА
по дисциплине «Алгоритмы и структуры данных»
Тема: Случайные БДП с рандомизацией.

Студентка гр. 8381

Бердникова А.А.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2019

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Студентка Бердникова А.А.

Группа 8381

Тема работы: Случайные БДП с рандомизацией.

Исходные данные: необходимо сделать демонстрацию алгоритма построения случайного БДП дерева с рандомизацией с поддержкой пошагового режима и сохранения текущего состояния в файл.

Содержание пояснительной записки:

«Содержание», «Введение», «Задание работы», «Ход выполнения работы»
«Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 28.10.2019

Дата сдачи реферата:

Дата защиты реферата:

Студентка

Бердникова А.А.

Преподаватель

Жангиров Т. Р.

Аннотация

В ходе выполнения курсовой работы была написана программа на фреймворке Qt, которая может обрабатывать входную строку, строить по ней бинарное дерево. Возможен ввод из файла и из поля ввода(значения корней подаются через пробел), так же есть возможность построить рандомное случайное БДП. Осуществлена визуализация данных, демонстрация алгоритма вставки и удаления элемента дерева, а также пошаговая демонстрация алгоритма вставки и пошаговая демонстрация алгоритма удаления элемента из случайного БДП.

Summary

During the course work, a program was written on the Qt framework, which can process the input string and build a binary tree on it. It is possible to enter from a file and from an input field (root values are given through a space), it is also possible to build a random random BJP. Data visualization, a demonstration of the algorithm for inserting and deleting a tree element, as well as a step-by-step demonstration of the insertion algorithm and a step-by-step demonstration of the algorithm for removing an element from a random BJP were performed.

СОДЕРЖАНИЕ

	Введение	5
1.	Описание программы	6
1.1	Описание интерфейса пользователя	6
1.2	Описание основных функций для случайного БДП с рандомизацией	7
1.3	Описание алгоритма вставки в случайное БДП с рандомизацией	8
1.4	Описание алгоритма удаления из случайного БДП	8
1.5	Описание методов демонстрации	9
2.	Демонстрация	10
2.1	Вид программы	11
2.2	Демонстрация рандомизированной вставки элемента	12
2.3	Демонстрация удаления элемента	12
	Заключение	14
	Список использованных источников	15
	Приложение А. Исходный код программы. main.cpp	16
	Приложение Б. Исходный код программы. mainwindow.h	17
	Приложение В. Исходный код программы. mainwindow.cpp	19
	Приложение Г. Исходный код программы. bintree.h	24
	Приложение Д. Исходный код программы. bintree.cpp	25
	Приложение Е. Исходный код программы. array_list.h	33
	Приложение Ж. Исходный код программы. array_list.cpp	34
	Приложение З. Исходный код программы. mainwindow.ui	37

ВВЕДЕНИЕ

Цель работы

Необходимо продемонстрировать случайное БДП с рандомизацией.

Основные задачи

Визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

Выполнение исследуемых алгоритмов, в том числе, в пошаговом режиме.

Методы решения

Разработка программы велась на базе операционной системы Windows 10 в среде разработки QtCreator [1]. Для создания графической оболочки использовался редактор интерфейса в QtCreator и система сигналов-слотов Qt[2]. Для графического отображения результатов исследования использовались классы QGraphicsView и QGraphicsScene.

1. ОПИСАНИЕ ПРОГРАММЫ

1.1. Описание интерфейса пользователя

Интерфейс программы [4] разделён на две части: верхняя панель ввода данных, построения дерева, установления промежутка возможного количества данных в рандомно-построенном дереве и управления ходом демонстрации; нижняя панель центральный виджет визуализации.

Основные виджеты и их назначение представлены в табл. 1.

Таблица 1 - Основные виджеты интерфейса

Класс объекта	Название виджета	Назначение
QPushButton	buildButton	Кнопка построения дерева
QPushButton	fileButton	Кнопка для ввода из файла
QLineEdit	inputEdit	Поле ввода
QPushButton	randButton	Кнопка для построения рандомного дерева
QSpinBox	randCountBox	Ввод количества элементов в рандомном дереве
QSpinBox	randMaxBox	Максимальное количество допустимых элементов в рандомном дереве
QLineEdit	elementEdit	Поле для ввода удаляемого/вставляемого элемента
QPushButton	insertButton	Кнопка вставки
QPushButton	nextButton	Кнопка далее
QPushButton	removeButton	Кнопка удаления

1.2. Описание основных функций для случайного БДП с рандомизацией.

Для реализации случайного БДП были созданы шаблонные функции для узла дерева - Node. Структура Node содержит поле для данных любого типа, цвет для визуализации, а также указатели на левый и правый дочерние узлы. Файл bintree.cpp содержит в себе ряд функций для работы с деревом и его узлами, они представлены в табл. 2.

Таблица 2 - Основные функции работы с деревом

Функция	Назначение
<code>void insert_old();</code> <code>void insert_step();</code>	Обычная вставка/Вставка пошагово
<code>void remove_old();</code> <code>void remove_step();</code>	Обычное удаление/Удаление пошагово
<code>int max_depth();</code>	Узнает глубину
<code>int getsize();</code>	Возвращает размер
<code>void fixsize();</code>	Установление корректного размера дерева
<code>Node* copy();</code>	Копирование дерева
<code>Node* rotateright();</code> <code>Node* rotateleft();</code>	Правый и левый поворот вокруг узла
<code>Node* insert_classic();</code>	Классическая вставка нового узла
<code>Node* insertroot();</code>	Вставка нового узла в корень
<code>Node* insert();</code>	Рандомизированная вставка узла в дерево
<code>Node* join();</code>	Объединение двух деревьев

1.3. Описание алгоритма вставки в случайное БДП с рандомизацией

Основное свойство дерева поиска — любой ключ в левом поддереве меньше корневого ключа, а в правом поддереве — больше корневого ключа (будем считать, что все ключи различны). Это свойство позволяет нам очень просто организовать вставку ключа, перемещаясь от корня вправо или влево в зависимости от значения корневого ключа, когда мы упираемся в пустой указатель, мы создаем новый узел и подвешиваем его к тому месту, где мы обнаружили тупик.

Немного о вставке в корень, которая используется в рандомизированной вставке. 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.

Из двух функции вставки (простой и в корень) можно составить рандомизированную вставку. Мы выполняем с $1/(n+1)$ вероятностью вставку в корень, а с вероятностью $1-1/(n+1)$ — рекурсивную вставку в правое или левое поддерево в зависимости от значения ключа в корне.

1.4. Описание алгоритма удаления из случайного БДП

Удаление происходит по ключу — ищем узел с заданным ключом и удаляем этот узел из дерева. Стадия поиска такая же как и при вставке, далее объединяем левое и правое поддерева найденного узла, удаляем узел, возвращаем корень объединенного дерева.

1.5. Описание методов демонстрации

Все методы визуализации располагаются в классе `mainwindow`.
Ниже перечислены основные из них и представлены в табл. 3.

Таблица 3 - Основные методы визуализации дерева

<code>int MainWindow::read(bool& ok);</code>	Считывает элемент из строки ввода
<code>void MainWindow::draw(QGraphicsScene * scene, Node *n, int maxdepth, int depth, int x, int y);</code>	Рекурсивно рисует на <code>graphicsscene</code> все дочерние узлы, включая текущий
<code>void MainWindow::visualize();</code>	Обновляет <code>graphicsscene</code> , визуализируя текущее состояние дерева

Особенностями данной визуализации являются цветовая маркировка проходящих в данный момент элементов, масштабирование приложения и выбор различных способов вставки и удаления.

2. ДЕМОНСТРАЦИЯ

2.1. Вид программы

Программа представляет собой окно с графическим интерфейсом, запускающимся в свернутом в окно режиме с уже введенными данными для тестирования. Вид программы после запуска представлен на рис. 1.

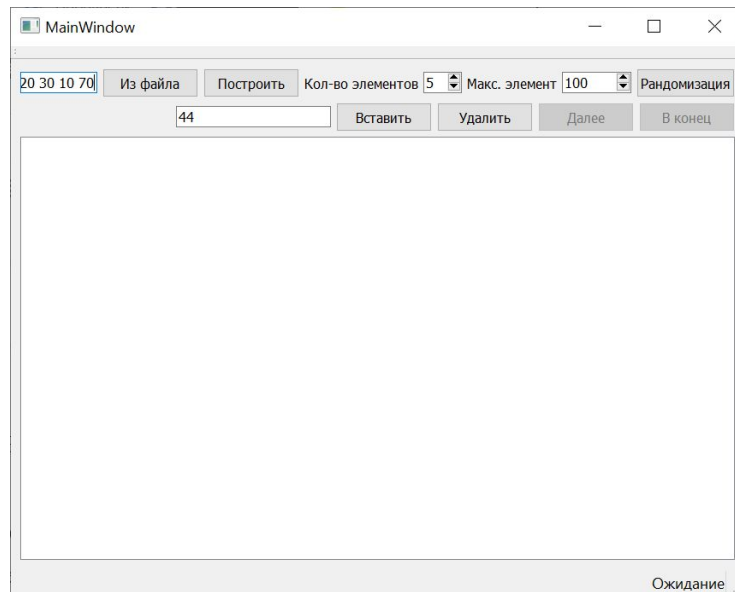


Рисунок 1 - Вид программы после запуска

Далее можно в крайнее левое окно ввести свои значения или выбрать ввод из файла, при нажатии построить строится заданное дерево.

Вид программы после создания БДП представлен на рис. 2.

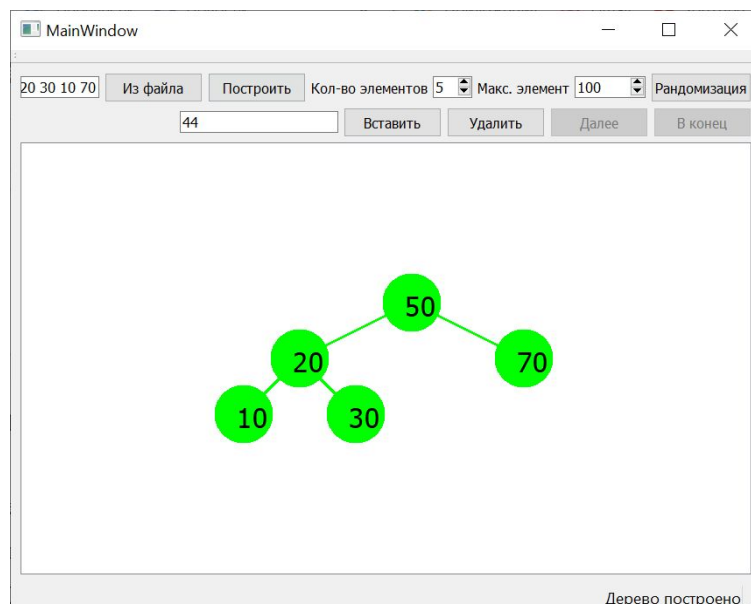


Рисунок 2 - Вид программы после создания БДП

2.2. Демонстрация рандомизированной вставки элемента

На рис. 3, 4, 5 представлен процесс вставки элемента

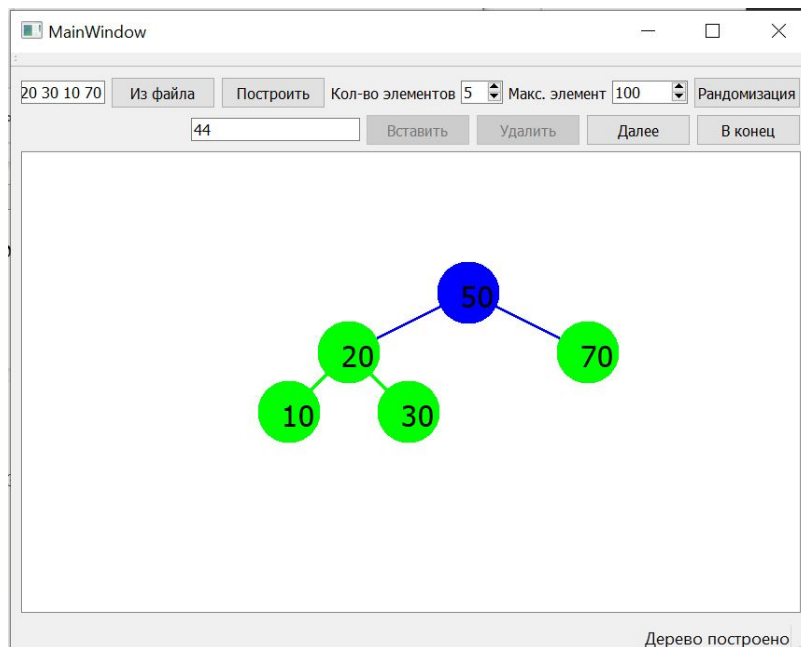


Рисунок 3 - Прохождение по элементам до места вставки

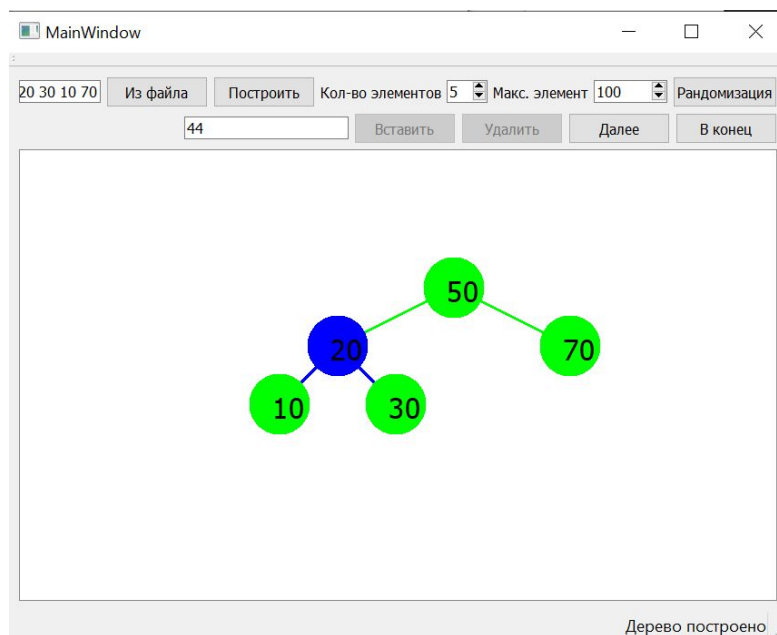


Рисунок 4 - Прохождение по элементам до места вставки

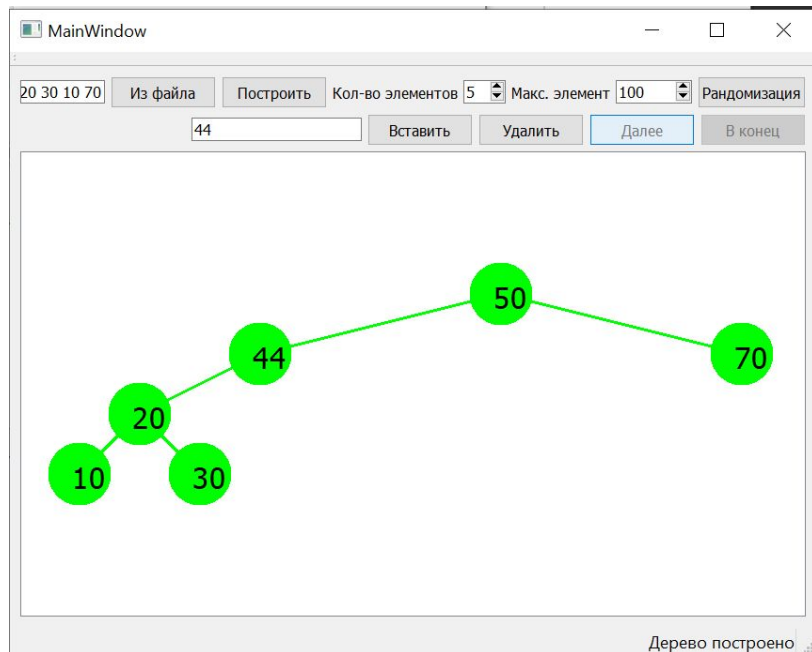


Рисунок 5 - Вставка элемента

2.3. Демонстрация удаления элемента

На рис. 6, 7, 8 представлен процесс удаления элемента

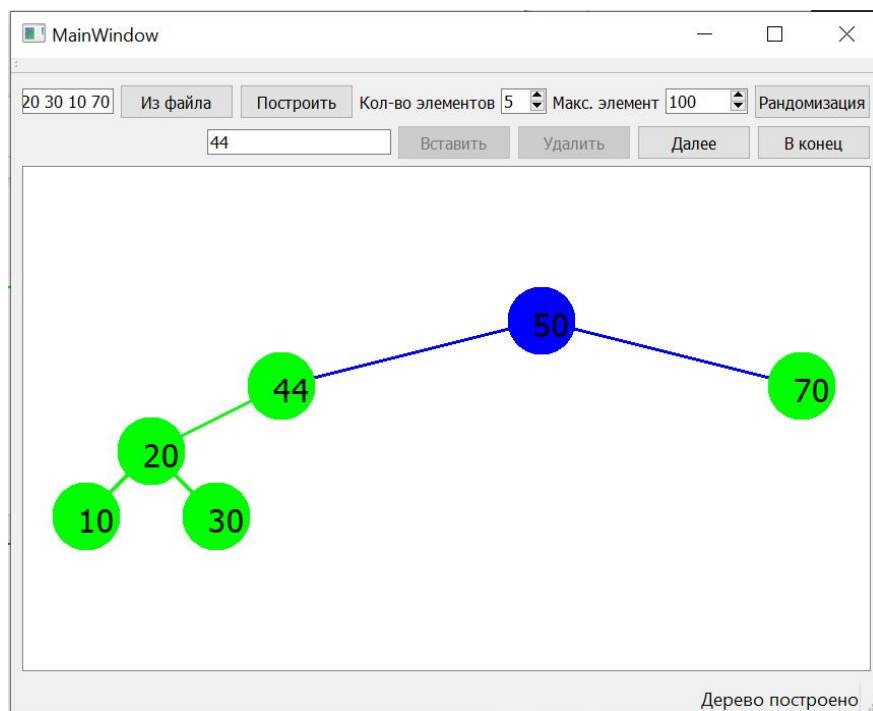


Рисунок 6 - Поиск удаляемого элемента

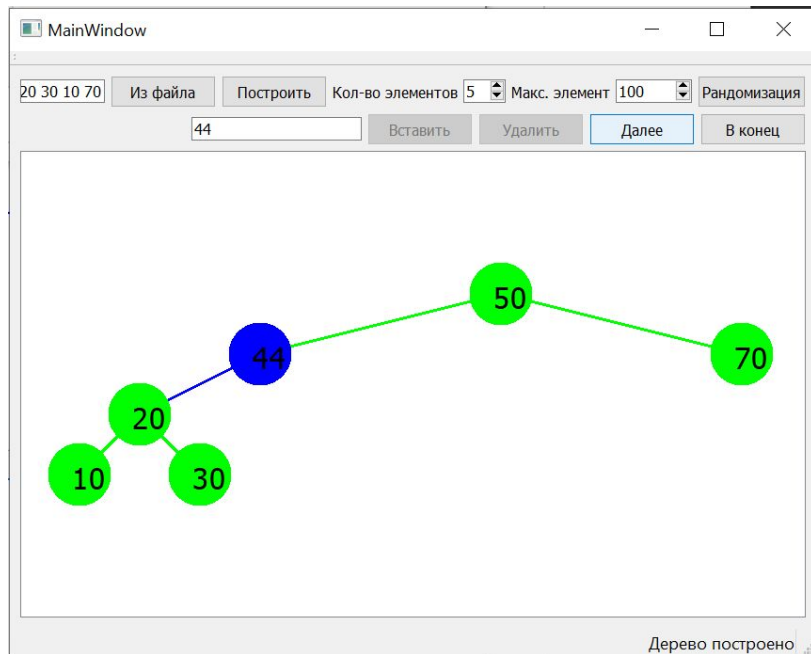


Рисунок 7 - Поиск удаляемого элемента

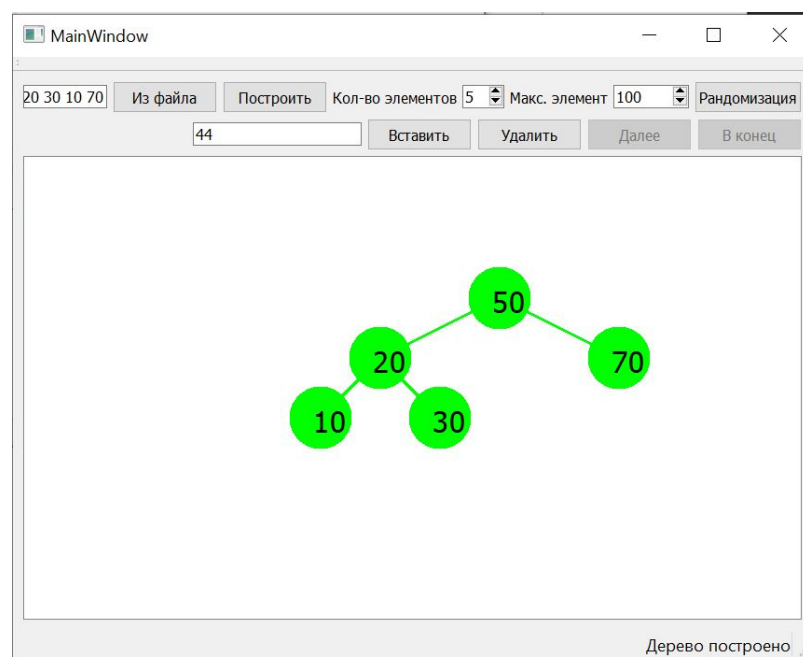


Рисунок 8 - Вид дерева после удаления элемента

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа, которая обладает следующей функциональностью: создание случайного БДП с рандомизацией по входным данным из разных источников ввода, рандомизированное и классическое добавление и исключение элементов, генерация рандомного случайного БДП, демонстрация пошаговых действий, что сопровождается цветовой маркировкой.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роберт Седжвик, Алгоритмы на C++, М.: Вильямс, 2011 г.
2. Martinez, Conrado; Roura, Salvador (1998), Randomized binary search trees, Journal of the ACM (ACM Press) 45 (2): 288–323
3. Reed, Bruce (2003), The height of a random binary search tree, Journal of the ACM 50 (3): 306–332
4. <https://habr.com/ru/post/145388/>
5. Qt Documentation // Qt. URL: <https://doc.qt.io/qt-5/index.html> (дата обращения: 20.12.2019)
6. Qt Documentation // Qt. URL: <https://doc.qt.io/qt-5/index.html> (дата обращения: 20.12.2019)

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAIN.CPP

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

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

    return a.exec();
}
```


ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.H

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include <QFileDialog>
#include <QMessageBox>
#include <QTextStream>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include "bintree.h"

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

    void reset();
    void visualize();
    void draw(QGraphicsScene* scene, Node *n, int maxdepth, int depth = 0, int
x = 0, int y = 0);
    int read(bool& ok);

    QGraphicsScene* graphicsScene;
    Node* tree = nullptr;
    array_list* log = nullptr;
    int index = 0;

    int mode = 0; // 0 - nothing, 1 - inserting, 2 - removing
    int rand_count = 5;
    int rand_max = 99;

    QLabel* status;
```

```
private slots:
    void on_fileButton_clicked();

    void on_buildButton_clicked();

    void on_randCountBox_valueChanged(int arg1);

    void on_randMaxBox_valueChanged(int arg1);

    void on_randButton_clicked();

    void on_insertButton_clicked();

    void on_removeButton_clicked();

    void on_nextButton_clicked();

    void on_endButton_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.CPP

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

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    status = new QLabel();
    ui->statusBar->addPermanentWidget(status);
    status->setText("Ожидание");
    graphicsScene = new QGraphicsScene();
    ui->graphicsView->setScene(graphicsScene);
    srand(time(NULL));
}

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

void MainWindow::reset()
{
    srand(time(nullptr));
    tree = nullptr;
    mode = 0;
    ui->insertButton->setEnabled(true);
    ui->removeButton->setEnabled(true);
    ui->nextButton->setEnabled(false);
    ui->endButton->setEnabled(false);
    visualize();
}

void MainWindow::visualize()
{
    graphicsScene->clear();
    if (tree) {
        draw(graphicsScene, tree, max_depth(tree, 1));
    }
}
```

```

void MainWindow::draw(QGraphicsScene* scene, Node *n, int maxdepth, int
depth, int x, int y)
{
    if (n == nullptr) return;
    int offset = pow(2, maxdepth + 3 - depth);
    // Lines
    QPen pen(n->color, 3);
    if (n->left) scene->addLine(x + 32, y + 32, x - offset + 32, y + 64 + 32, pen);
    if (n->right) scene->addLine(x + 32, y + 32, x + offset + 32, y + 64 + 32, pen);
    // Ellipse
    QBrush brush(n->color);
    scene->addEllipse(x, y, 64, 64, pen, brush);
    // Text
    QGraphicsTextItem *numb = new QGraphicsTextItem();
    numb->setPlainText(QString::number(n->data));
    numb->setDefaultTextColor(Qt::black);
    numb->setScale(2);
    numb->setPos(x + 16, y + 8);
    scene->addItem(numb);
    // Next
    draw(scene, n->left, maxdepth, depth + 1, x - offset, y + 64);
    draw(scene, n->right, maxdepth, depth + 1, x + offset, y + 64);
}

int MainWindow::read(bool& ok)
{
    ok = true;
    if (ui->elementEdit->text().isEmpty())
    {
        status->setText("Введите элемент!");
        ok = false;
        return 0;
    }
    bool isok;
    int res = ui->elementEdit->text().toInt(&isok);
    if (!isok)
    {
        status->setText("Элемент - не число!");
        ui->elementEdit->setText("");
        ok = false;
        return 0;
    }
    return res;
}

```

```

void MainWindow::on_fileButton_clicked()
{
    QString inputStr;
    QString fileName = QFileDialog::getOpenFileName(this, "Open TXT File",
QDir::homePath(), "TXT text (*.txt);;All Files (*)");
    if (fileName == nullptr)
    {
        status->setText("Имя файла не было выбрано");
        return;
    }
    QFile file(fileName);
    if(file.open(QIODevice::ReadOnly | QIODevice::Text)) {
        QTextStream stream(&file);
        inputStr = stream.readAll();
    }
    if(inputStr.isEmpty())
        return;
    file.close();
    ui->inputEdit->setText(inputStr);
}

```

```

void MainWindow::on_buildButton_clicked()
{
    reset();
    int i = 0;
    string raw = ui->inputEdit->text().toStdString();
    if (parse_tree(tree, raw, i))
    {
        tree = nullptr;
        status->setText("Ошибка обработки дерева");
        return;
    }
    visualize();
    status->setText("Дерево построено");
}

```

```

void MainWindow::on_randCountBox_valueChanged(int arg1)
{
    rand_count = arg1;
}

```

```

void MainWindow::on_randMaxBox_valueChanged(int arg1)
{
    rand_max = arg1;
}

```

```

void MainWindow::on_randButton_clicked()
{
    reset();
    for (int i = 0; i < rand_count; i++)
    {
        int r = rand() % rand_max;
        insert_old(tree, r);
    }
    visualize();
    status->setText("Рандомизация прошла успешно");
}

```

```

void MainWindow::on_insertButton_clicked()
{
    bool ok;
    int val = read(ok);
    if (!ok) return;
    if (tree) {
        log = new array_list();
        mode = 1;
        index = 0;
        ui->insertButton->setEnabled(false);
        ui->removeButton->setEnabled(false);
        ui->nextButton->setEnabled(true);
        ui->endButton->setEnabled(true);
        Node* ncopy = copy(tree);
        ncopy = insert(log, ncopy, ncopy, val);
        log->push_back(ncopy);
    }
    visualize();
}

```

```

void MainWindow::on_removeButton_clicked()
{
    bool ok;
    int val = read(ok);
    if (!ok) return;
    if (tree) {
        log = new array_list();
        mode = 2;
        index = 0;
        ui->insertButton->setEnabled(false);
        ui->removeButton->setEnabled(false);
        ui->nextButton->setEnabled(true);
        ui->endButton->setEnabled(true);
        Node* ncopy = copy(tree);

```

```

        ncopy = remove(log, ncopy, ncopy, val);
        log->push_back(ncopy);
    }
    visualize();
}

void MainWindow::on_nextButton_clicked()
{
    if (index >= log->size() - 1)
    {
        on_endButton_clicked();
    }
    else
    {
        tree = log->at(index);
        index++;
        visualize();
    }
}

void MainWindow::on_endButton_clicked()
{
    tree = log->at(log->size() - 1);
    mode = 0;
    ui->insertButton->setEnabled(true);
    ui->removeButton->setEnabled(true);
    ui->nextButton->setEnabled(false);
    ui->endButton->setEnabled(false);
    visualize();
}

```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.H

```
#ifndef BINTREE_H
#define BINTREE_H

#include <string>
#include <QColor>

#include "node.h"
#include "array_list.h"

using namespace std;

int max_depth(Node *n, int i);
bool parse_tree(Node*& n, std::string &s, int &i);
void insert_old(Node*& n, int data);
void insert_step(array_list*& arr, Node*& root, Node*& n, int data);
void remove_old(Node*& n, int data);
void remove_step(array_list*& arr, Node*& root, Node*& n, int data);
Node* copy(Node* n);
string into_string(Node* n);

int getsize(Node* p);
void fixsize(Node* p);
Node* rotateright(Node* p);
Node* rotateleft(Node* q);
Node* insert_classic(Node* p, int k);
Node* insertroot(Node* p, int k);
Node* insert(array_list*& arr, Node*& root, Node* p, int k);
Node* join(Node* p, Node* q);
Node* remove(array_list*& arr, Node*& root, Node* p, int k);

#endif // BINTREE_H
```


ПРИЛОЖЕНИЕ Д

ИСХОДНЫЙ КОД ПРОГРАММЫ. BINTREE.CPP

```
#include "bintree.h"

int max_depth(Node *n, int i)
{
    if (!n) return i;
    int l = max_depth(n->left, i + 1);
    int r = max_depth(n->right, i + 1);
    if (l > r) return l;
    else return r;
}

bool parse_tree(Node*& n, std::string &s, int &i) {
    if (s.size() < 1) return true;
    std::size_t current, previous = 0;
    current = s.find(' ');
    int num;
    while (current != std::string::npos) {
        try
        {
            num = stoi(s.substr(previous, current - previous));
        }
        catch (...)
        {
            return true;
        }
        previous = current + 1;
        current = s.find(' ', previous);
        insert_old(n, num);
    }
    try
    {
        num = stoi(s.substr(previous, current - previous));
    }
    catch (...)
    {
        return true;
    }
    insert_old(n, num);
    return false;
}
```

```

void insert_old(Node*& n, int data)
{
    if (!n)
    {
        n = new Node();
        n->data = data;
    }
    else if (n->data > data)
    {
        insert_old(n->left, data);
    }
    else if (n->data < data)
    {
        insert_old(n->right, data);
    }
}

void remove_old(Node *&n, int data)
{
    if (!n) return;
    if (data < n->data)
    {
        remove_old(n->left, data);
    }
    else if (data > n->data)
    {
        remove_old(n->right, data);
    }
    else
    {
        if (!n->left && n->right)
        {
            Node* temp = n->right;
            delete n;
            n = temp;
        }
        else if (!n->right && n->left)
        {
            Node* temp = n->left;
            delete n;
            n = temp;
        }
        else if (!n->right && !n->left)
        {
            delete n;
            n = nullptr;
        }
    }
}

```

```

    }
    else
    {
        Node* min = n->right;
        if (!min->left)
        {
            n->right = nullptr;
        }
        else
        {
            Node* t = min;
            while(t->left->left)
            {
                t = t->left;
            }
            min = t->left;
            t->left = nullptr;
        }
        n->data = min->data;
        delete min;
    }
}
}

```

```

Node *copy(Node *n)
{
    if (!n) return nullptr;
    Node* co = new Node();
    co->data = n->data;
    co->left = copy(n->left);
    co->right = copy(n->right);
    co->color = n->color;
    return co;
}

```

```

string into_string(Node *n)
{
    if (!n) return "";
    else return "(" + to_string(n->data) + into_string(n->left) +
into_string(n->right) + ")";
}

```

```

void insert_step(array_list *&arr, Node*& root, Node *&n, int data)
{
    if (!n)
    {

```

```

    n = new Node();
    n->data = data;
    arr->push_back(copy(root));
}
else if (n->data > data)
{
    n->color = QColor::fromRgb(0, 0, 255);
    arr->push_back(copy(root));
    n->color = QColor::fromRgb(0, 255, 0);
    insert_step(arr, root, n->left, data);
}
else if (n->data < data)
{
    n->color = QColor::fromRgb(0, 0, 255);
    arr->push_back(copy(root));
    n->color = QColor::fromRgb(0, 255, 0);
    insert_step(arr, root, n->right, data);
}
else
{
    arr->push_back(copy(root));
}
}

```

```

void remove_step(array_list *&arr, Node*& root, Node *&n, int data)
{
    if (!n) return;
    if (data < n->data)
    {
        n->color = QColor::fromRgb(0, 0, 255);
        arr->push_back(copy(root));
        n->color = QColor::fromRgb(0, 255, 0);
        remove_step(arr, root, n->left, data);
    }
    else if (data > n->data)
    {
        n->color = QColor::fromRgb(0, 0, 255);
        arr->push_back(copy(root));
        n->color = QColor::fromRgb(0, 255, 0);
        remove_step(arr, root, n->right, data);
    }
    else
    {
        if (!n->left && n->right)
        {
            Node* temp = n->right;

```

```

        delete n;
        n = temp;
    }
    else if (!n->right && n->left)
    {
        Node* temp = n->left;
        delete n;
        n = temp;
    }
    else if (!n->right && !n->left)
    {
        delete n;
        n = nullptr;
    }
    else
    {
        Node* min = n->right;
        if (!min->left)
        {
            n->right = nullptr;
        }
        else
        {
            Node* t = min;
            while(t->left->left)
            {
                t = t->left;
            }
            min = t->left;
            t->left = nullptr;
        }
        n->data = min->data;
        delete min;
    }
    arr->push_back(copy(root));
}
}

```

```

int getsize(Node* p) // обертка для поля size, работает с пустыми деревьями
(t=NULL)
{
    if( !p ) return 0;
    return p->size;
}

```

```

void fixsize(Node* p) // установление корректного размера дерева

```

```

{
    p->size = getsize(p->left)+getsize(p->right)+1;
}

```

Node* rotateright(Node* p) // правый поворот вокруг узла p

```

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

```

Node* insert_classic(Node* p, int k) // классическая вставка нового узла с ключом k в дерево p

```

{
    if( !p ) return new Node(k);
    if( p->data > k )
        p->left = insert_classic(p->left,k);
    else
        p->right = insert_classic(p->right,k);
    fixsize(p);
    return p;
}

```

Node* insertroot(Node* p, int k) // вставка нового узла с ключом k в корень дерева p

```

{
    if( !p ) return new Node(k);
    if( k < p->data )
    {
        p->left = insertroot(p->left,k);
        return rotateright(p);
    }
}

```

```

    }
    else if ( k > p->data )
    {
        p->right = insertroot(p->right,k);
        return rotateleft(p);
    }
    else
    {
        return p;
    }
}

```

Node* insert(array_list*& arr, Node*& root, Node* p, int k) // рандомизированная вставка нового узла с ключом k в дерево p

```

{
    if( !p ) return new Node(k);
    p->color = QColor::fromRgb(0, 0, 255);
    arr->push_back(copy(root));
    p->color = QColor::fromRgb(0, 255, 0);
    if( rand()%(p->size+1)==0 )
    {
        return insertroot(p,k);
    }
    if( p->data>k )
        p->left = insert(arr, root, p->left,k);
    else if ( p->data<k )
        p->right = insert(arr, root, p->right,k);
    fixsize(p);
    return p;
}
//

```

Node* 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* remove(array_list*& arr, Node*& root, Node* p, int k) // удаление из дерева p первого найденного узла с ключом k

```

{
    if( !p ) return p;
    p->color = QColor::fromRgb(0, 0, 255);
    arr->push_back(copy(root));
    p->color = QColor::fromRgb(0, 255, 0);
    if( p->data==k )
    {
        Node* q = join(p->left,p->right);
        delete p;
        return q;
    }
    else if( k<p->data )
    {
        p->left = remove(arr, root, p->left,k);
    }
    else if( k>p->data )
    {
        p->right = remove(arr, root, p->right,k);
    }
    return p;
}

```


ПРИЛОЖЕНИЕ Е

ИСХОДНЫЙ КОД ПРОГРАММЫ. ARRAY_LIST.H

```
#ifndef ARRAY_LIST_H
#define ARRAY_LIST_H
#include <string>
#include "node.h"

class array_list
{
private:
    Node** array;
    int capacity;
    int count;
    void resize(int new_capacity);

public:
    array_list(int start_capacity = 4);
    Node* at (int index);
    void clean();
    void insert(int index, Node* element);
    Node* remove(int index);

    Node* back();
    void push_back(Node* element);
    Node* pop_back();

    Node* front();
    void push_front(Node* element);
    Node* pop_front();

    int size();
    bool empty();
    ~array_list();
};

#endif // ARRAY_LIST_H
```

ПРИЛОЖЕНИЕ Ж

ИСХОДНЫЙ КОД ПРОГРАММЫ. ARRAY_LIST.CPP

```
#ifndef UTILS_VECTOR_H
#define UTILS_VECTOR_H
#include "array_list.h"

void array_list::resize(int new_capacity)
{
    auto *arr = new Node*[count];
    for (int i = 0; i < count; ++i)
    {
        arr[i] = array[i];
    }
    delete [] array;
    array = new Node*[new_capacity];
    for (int i = 0 ; i < count; ++i)
    {
        array[i] = arr[i];
    }
    delete [] arr;
    capacity = new_capacity;
}

array_list::array_list(int start_capacity)
{
    capacity = start_capacity;
    count = 0;
    array = new Node*[capacity];
}

Node* array_list::at (int index)
{
    return array[index];
}

void array_list::clean ()
{
    count = 0;
}

void array_list::insert(int index, Node* element)
{

```

```

    if (capacity == count)
    {
        resize(count + 8);
    }
    if (count > 0) {
        for (int i = count; i > index; i--)
        {
            array[i] = array[i - 1];
        }
    }
    count++;
    array[index] = element;
}

```

```

Node* array_list::remove(int index)
{
    auto temp = array[index];
    for (int i = index; i < count - 1; i++)
    {
        array[i] = array[i + 1];
    }
    count--;
    return temp;
}

```

```

Node* array_list::back()
{
    return array[count - 1];
}

```

```

void array_list::push_back(Node* element)
{
    if (capacity == count)
    {
        resize(count + 8);
    }
    array[count] = element;
    count++;
}

```

```

Node* array_list::pop_back()
{
    return array[--count];
}

```

```

Node* array_list::front()

```

```

{
    return *array;
}

void array_list::push_front(Node* element)
{
    insert(0, element);
}

Node* array_list::pop_front()
{
    return remove(0);
}

int array_list::size()
{
    return count;
}

bool array_list::empty()
{
    return !count;
}

array_list::~~array_list()
{
    delete [] array;
}

#endif //VECTOR_VECTOR_H

```

ПРИЛОЖЕНИЕ 3

ИСХОДНЫЙ КОД ПРОГРАММЫ. MAINWINDOW.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>857</width>
        <height>644</height>
      </rect>
    </property>
    <property name="windowTitle">
      <string>MainWindow</string>
    </property>
    <property name="styleSheet">
      <string notr="true"/>
    </property>
    <widget class="QWidget" name="centralWidget">
      <layout class="QVBoxLayout" name="verticalLayout_2">
        <item>
          <layout class="QHBoxLayout" name="horizontalLayout">
            <item>
              <widget class="QLineEdit" name="inputEdit">
                <property name="text">
                  <string>50 20 30 10 70</string>
                </property>
              </widget>
            </item>
            <item>
              <widget class="QPushButton" name="fileButton">
                <property name="text">
                  <string>Из файла</string>
                </property>
              </widget>
            </item>
            <item>
              <widget class="QPushButton" name="buildButton">
                <property name="text">
                  <string>Построить</string>
                </property>
              </widget>
            </item>
          </layout>
        </item>
      </layout>
    </widget>
  </widget>
</ui>
```

```

</item>
<item>
  <spacer name="horizontalSpacer">
    <property name="orientation">
      <enum>Qt::Horizontal</enum>
    </property>
    <property name="sizeHint" stdset="0">
      <size>
        <width>40</width>
        <height>20</height>
      </size>
    </property>
  </spacer>
</item>
<item>
  <widget class="QLabel" name="label">
    <property name="text">
      <string>Кол-во элементов</string>
    </property>
  </widget>
</item>
<item>
  <widget class="QSpinBox" name="randCountBox">
    <property name="value">
      <number>5</number>
    </property>
  </widget>
</item>
<item>
  <widget class="QLabel" name="label_2">
    <property name="text">
      <string>Макс. элемент</string>
    </property>
  </widget>
</item>
<item>
  <widget class="QSpinBox" name="randMaxBox">
    <property name="maximum">
      <number>100000</number>
    </property>
    <property name="value">
      <number>100</number>
    </property>
  </widget>
</item>
<item>

```

```

<widget class="QPushButton" name="randButton">
  <property name="text">
    <string>Рандомизация</string>
  </property>
</widget>
</item>
</layout>
</item>
<item>
<layout class="QHBoxLayout" name="horizontalLayout_2">
  <item>
    <spacer name="horizontalSpacer_2">
      <property name="orientation">
        <enum>Qt::Horizontal</enum>
      </property>
      <property name="sizeHint" stdset="0">
        <size>
          <width>40</width>
          <height>20</height>
        </size>
      </property>
    </spacer>
  </item>
  <item>
    <widget class="QLineEdit" name="elementEdit">
      <property name="text">
        <string>44</string>
      </property>
    </widget>
  </item>
  <item>
    <widget class="QPushButton" name="insertButton">
      <property name="text">
        <string>Вставить</string>
      </property>
    </widget>
  </item>
  <item>
    <widget class="QPushButton" name="removeButton">
      <property name="text">
        <string>Удалить</string>
      </property>
    </widget>
  </item>
  <item>
    <widget class="QPushButton" name="nextButton">

```

```

    <property name="enabled">
      <bool>false</bool>
    </property>
    <property name="text">
      <string>Далее</string>
    </property>
  </widget>
</item>
<item>
  <widget class="QPushButton" name="endButton">
    <property name="enabled">
      <bool>false</bool>
    </property>
    <property name="text">
      <string>В конец</string>
    </property>
  </widget>
</item>
</layout>
</item>
<item>
  <widget class="QGraphicsView" name="graphicsView"/>
</item>
</layout>
</widget>
<widget class="QMenuBar" name="menuBar">
  <property name="geometry">
    <rect>
      <x>0</x>
      <y>0</y>
      <width>857</width>
      <height>17</height>
    </rect>
  </property>
</widget>
<widget class="QToolBar" name="mainToolBar">
  <attribute name="toolBarArea">
    <enum>TopToolBarArea</enum>
  </attribute>
  <attribute name="toolBarBreak">
    <bool>false</bool>
  </attribute>
</widget>
<widget class="QStatusBar" name="statusBar"/>
</widget>
<layoutdefault spacing="6" margin="11"/>

```



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