Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Пермский национальный исследовательский
политехнический университет»**

Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»
направление подготовки: 09.03.01– «Информатика и вычислительная техника»

Лабораторная работа
по дисциплине
«Теория алгоритмов и структуры данных»
на тему
«Бинарное дерево»
Вариант 2

Выполнил студент гр. ИВТ-23-1б
Попонин Михаил Александрович

Проверил:
Доцент каф. ИТАС
Яруллин Денис Владимирович

(оценка) (подпись)

(дата)

г. Пермь, 2024

Цель и задачи работы

Целью данной работы является получение навыков работы с бинарными деревьями

Вариант 2: Тип информационного поля int. Найти максимальный элемент в дереве

UML диаграмма

На рисунке 1 изображена диаграмма класса

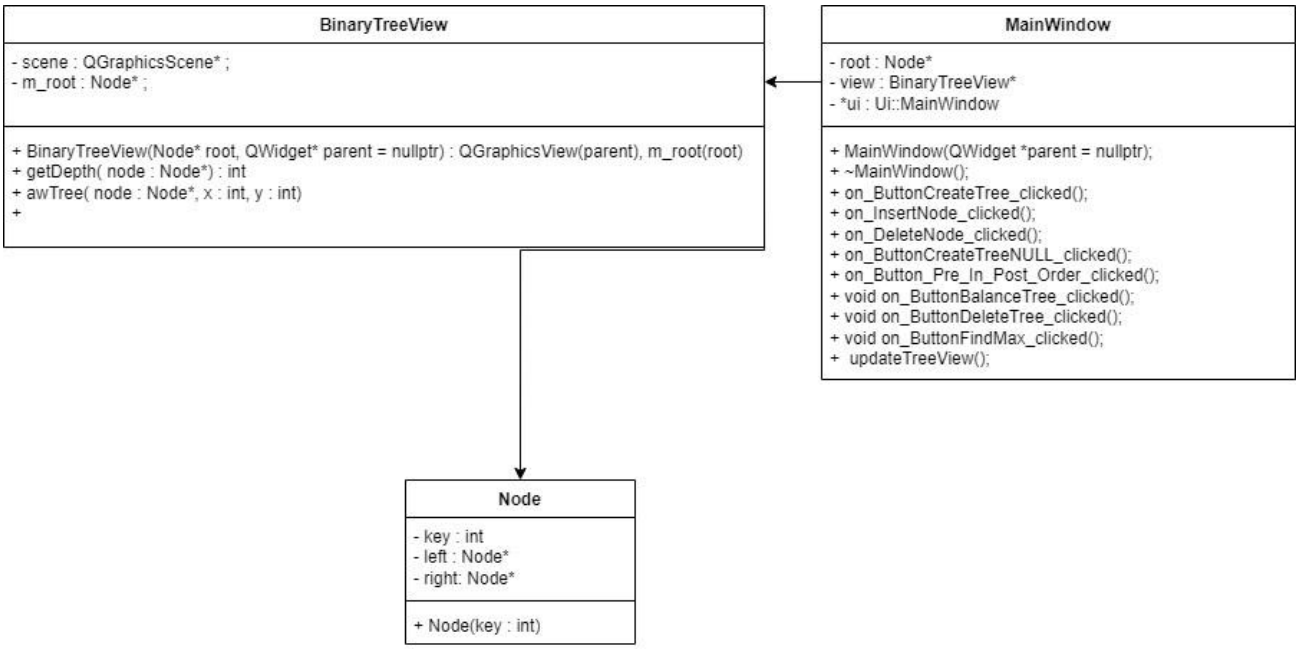

Рисунок 1

Код программы

Таблица 1 – Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent), ui(new
Ui::MainWindow)
{
    ui->setupUi(this);
    view = nullptr;
}
MainWindow::~MainWindow()
{
    delete ui;
    if (view != nullptr)
        delete view;
}
void MainWindow::updateTreeView()
{
    if (view != nullptr)
    {
        delete view;
    }
    view = new BinaryTreeView(root, this);
    view->resize(1000, 530);
    view->show();
}
void preorder(Node* root, QString& result)
{
    if (root == nullptr)
    {
        return;
    }
    result += QString::number(root->key) + " ";
    preorder(root->left, result);
    preorder(root->right, result);
}
void inorder(Node* root, QString& result)
{
    if (root == nullptr)
    {
        return;
    }
    inorder(root->left, result);
    result += QString::number(root->key) + " ";
    inorder(root->right, result);
}
```

```
void postorder(Node* root, QString& result)
{
    if (root == nullptr)
    {
        return;
    }
    postorder(root->left, result);
    postorder(root->right, result);
    result += QString::number(root->key) + " ";
}
void deleteTree(Node* &root)
{
    if (root == nullptr)
    {
        return;
    }
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
    root = nullptr;
}
Node* findMin(Node* node)
{
    Node* current = node;
    while (current->left != nullptr) {
        current = current->left;
    }
    return current;
}
void deleteNode(Node* &root, int key)
{
    if (root == nullptr)
    {
        return;
    }

    if (key < root->key)
    {
        deleteNode(root->left, key);
    } else if (key > root->key)
    {
        deleteNode(root->right, key);
    } else {
        if (root->left == nullptr)
        {
            Node* temp = root->right;
            delete root;
            root = temp;
```

```
        } else if (root->right == nullptr)
        {
            Node* temp = root->left;
            delete root;
            root = temp;
        } else {
            Node* temp = findMin(root->right);
            root->key = temp->key;
            deleteNode(root->right, temp->key);
        }
    }
}
void insert(Node* root, int key)
{
    if (root == nullptr)
    {
        return;
    }
    if (key < root->key)
    {
        if (root->left == nullptr)
        {
            root->left = new Node(key);
        } else
        {
            insert(root->left, key);
        }
    } else
    {
        if (root->right == nullptr)
        {
            root->right = new Node(key);
        } else
        {
            insert(root->right, key);
        }
    }
}
void storeInorder(Node* root, std::vector<int> &values)
{
    if (root == nullptr)
    {
        return;
    }
    storeInorder(root->left, values);
    values.push_back(root->key);
    storeInorder(root->right, values);
}
```

```
Node* buildBalancedTree(const std::vector<int> &values, int start, int end)
{
    if (start > end)
    {
        return nullptr;
    }
    int mid = (start + end) / 2;
    Node* root = new Node(values[mid]);
    root->left = buildBalancedTree(values, start, mid - 1);
    root->right = buildBalancedTree(values, mid + 1, end);
    return root;
}
Node* balanceTree(Node* root)
{
    std::vector<int> values;
    storeInorder(root, values);
    if (values.empty())
    {
        return nullptr;
    }
    std::sort(values.begin(), values.end());
    return buildBalancedTree(values, 0, values.size() - 1);
}
void MainWindow::on_ButtonCreateTree_clicked()
{
    root = new Node(rand()%100-rand()%100);
    for(int i = 0; i < rand()%100; i++)
    {
        insert(root, (rand()%100-rand()%100));
    }
    updateTreeView();
    ui->statusbar->showMessage("Дерево успешно создано!");
}
void MainWindow::on_InsertNode_clicked()
{
    QString value = ui->lineEdit->text();
    int key = value.toInt();
    insert(root, key);
    updateTreeView();
    ui->lineEdit->clear();
    ui->statusbar->showMessage("Узел успешно вставлен!");
}
void MainWindow::on_DeleteNode_clicked()
{
    QString value = ui->lineEdit_2->text();
    int key = value.toInt();
    deleteNode(root, key);
    updateTreeView();
```

```
    ui->lineEdit_2->clear();
    ui->statusbar->showMessage("Узел успешно удалён!");
}
void MainWindow::on_ButtonCreateTreeNULL_clicked()
{
    root = new Node(0);
    updateTreeView();
    ui->statusbar->showMessage("Пустое дерево успешно создано!");
}
void MainWindow::on_Button_Pre_In_Post_Order_clicked()
{
    ui->textBrowser->clear();
    QString result;
    result = "preorder: ";
    preorder(root, result);
    ui->textBrowser->append(result);

    result = "inorder: ";
    inorder(root, result);
    ui->textBrowser->append(result);

    result = "postorder: ";
    postorder(root, result);
    ui->textBrowser->append(result);
    ui->statusbar->showMessage("Алгоритмы: прямой, симметричный, обратный успешно
выполнены!");
}
void MainWindow::on_ButtonBalanceTree_clicked()
{
    if (root == nullptr)
    {
        ui->statusbar->showMessage("Дерево пусто!");
        return;
    }

    Node* newRoot = balanceTree(root);
    if (newRoot != nullptr)
    {
        deleteTree(root);
        root = newRoot;
        updateTreeView();
        ui->statusbar->showMessage("Дерево успешно сбалансировано!");
    }
    else
    {
        ui->statusbar->showMessage("Ошибка при балансировке дерева!");
    }
}
```

```
void MainWindow::on_ButtonDeleteTree_clicked()
{
    deleteTree(root);
    updateTreeView();
    ui->statusbar->showMessage("Дерево успешно уничтожено!");
}
Node* findMax(Node* node)
{
    if (node == nullptr)
    {
        return nullptr;
    }
    Node* current = node;
    while (current->right != nullptr)
    {
        current = current->right;
    }
    return current;
}
void MainWindow::on_ButtonFindMax_clicked()
{
    if (root == nullptr)
    {
        ui->statusbar->showMessage("Дерево пусто!");
        return;
    }
    Node* maxNode = findMax(root);
    if (maxNode != nullptr)
    {
        ui->statusbar->showMessage("Максимальный элемент: " + QString::number(maxNode-
>key));
    }
    else
    {
        ui->statusbar->showMessage("Максимальный элемент не найден!");
    }
}
```

Таблица 2 – Файл main.cpp

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

Таблица 3 – файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <QMainWindow>
#include <QPainter>
#include <QApplication>
#include <QWidget>
#include <QFont>
#include <iostream>
#include <cmath>
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsTextItem>
QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE
class Node
{
public:
    int key;
    Node* left;
    Node* right;
    Node(int key)
    {
        this->key = key;
        this->left = nullptr;
        this->right = nullptr;
    }
};
void insert(Node* root, int key);
void deleteNode(Node* &root, int key);
void preorder(Node* root);
void inorder(Node* root);
void postorder(Node* root);
void deleteTree(Node* &root);
Node* balanceTree(Node* root);
class BinaryTreeView : public QGraphicsView
{
public:
    BinaryTreeView(Node* root, QWidget* parent = nullptr) : QGraphicsView(parent),
m_root(root)
    {
        scene = new QGraphicsScene(this);
        setScene(scene);
        drawTree(m_root, 630, 50);
    }
private:
```

```
    QGraphicsScene* scene;
    Node* m_root;
    int getDepth(Node* node)
    {
        if (node == nullptr)
        {
            return 0;
        }
        int leftDepth = getDepth(node->left);
        int rightDepth = getDepth(node->right);
        return 1 + std::max(leftDepth, rightDepth);
    }
    void drawTree(Node* node, int x, int y)
    {
        if (node == nullptr)
        {
            return;
        }
        QPen linePen(Qt::magenta);
        linePen.setWidth(2);
        QBrush circleBrush(Qt::white);
        scene->addEllipse(x, y, 40, 40, linePen, circleBrush);
        int offsetX_l = 30 * pow(2, getDepth(node->left) - 1);
        int offsetX_r = 30 * pow(2, getDepth(node->right) - 1);
        int offsetY = 70;
        QFont font("Arial", 12);
        QGraphicsTextItem* textItem = scene->addText(QString::number(node->key), font);
        textItem->setPos(x, y);
        if (node->left != nullptr)
        {
            scene->addLine(x + 20, y + 40, x - offsetX_l + 20, y + offsetY, linePen);
            drawTree(node->left, x - offsetX_l, y + offsetY);
        }
        if (node->right != nullptr)
        {
            scene->addLine(x + 40, y + 20, x + offsetX_r + 20, y + offsetY, linePen);
            drawTree(node->right, x + offsetX_r, y + offsetY);
        }
    }
};
class MainWindow : public QMainWindow
{
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
private slots:
    void on_ButtonCreateTree_clicked();
```

```
    void on_InsertNode_clicked();
    void on_DeleteNode_clicked();
    void on_ButtonCreateTreeNULL_clicked();
    void on_Button_Pre_In_Post_Order_clicked();
    void on_ButtonBalanceTree_clicked();
    void on_ButtonDeleteTree_clicked();
    void on_ButtonFindMax_clicked();
private:
    Ui::MainWindow *ui;
    Node* root;
    BinaryTreeView* view;
    void updateTreeView();
};
#endif // MAINWINDOW_H
```

Демонстрация работы программы:

https://youtu.be/RLrhJaB2s7Y?si=9mEkZ8TXPXU4vQgp

Скриншоты работы программы: