Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра **«**Информационные технологии и автоматизированные системы**»**

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Дисциплина: «Основы алгоритмизации и программирования»

Семестр 2

Тема: Деревья

Выполнил работу

Студент группы РИС-22-1Б

Головин М.А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

г. Пермь-2023

**Введение**

Для выполнения лабораторной работы требуется сделать отчет программы. Создание программ – отличный способ практики программирования.

**Постановка задачи**

Требуется реализовать алгоритмы для собственного варианта бинарного дерева поиска, имеющего не менее трёх уровней.

Задания:

1. Необходимо реализовать функции для редактирования дерева:

* Вставка узла.
* Удаление узла.
* Поиск элемента по ключу.

1. Реализовать алгоритмы обхода дерева:

* Прямой
* Симметричный
* Обратный

1. Реализовать алгоритм балансировки дерева.
2. Визуализация дерева должна быть выполнена с использованием любой доступной графической библиотеки – SFML, SDL, OpenGL и подобных.
3. Пользовательский интерфейс по усмотрению разработчика - с условием кроссплатформенности (поощряется использование Qt или иных фреймворков).

**Код программы**

**main.cpp**

#include <QApplication>

#include <QWidget>

#include <QPushButton>

#include <QHBoxLayout>

#include <QVBoxLayout>

#include <QLabel>

#include <QInputDialog>

#include <QPainter>

#include <QDebug>

#include <QMessageBox>

// Структура узла бинарного дерева

struct TreeNode {

int key;

TreeNode\* left;

TreeNode\* right;

TreeNode(int value) : key(value), left(nullptr), right(nullptr) {}

};

class BinaryTreeWidget : public QWidget {

public:

BinaryTreeWidget(QWidget\* parent = nullptr) : QWidget(parent), root(nullptr) {}

void insertNode(int key) {

root = insert(root, key);

update();

}

void deleteNode(int key) {

root = remove(root, key);

update();

}

bool searchNode(int key) {

return search(root, key);

}

void preOrderTraversal() {

preOrder(root);

}

void inOrderTraversal() {

inOrder(root);

}

void postOrderTraversal() {

postOrder(root);

}

void balanceTree() {

root = balance(root);

update();

}

void paintEvent(QPaintEvent\* event) override {

Q\_UNUSED(event);

QPainter painter(this);

painter.setRenderHint(QPainter::Antialiasing);

int initialX = width() / 2;

int initialY = 50;

drawTree(painter, initialX, initialY, root, 1);

}

private:

TreeNode\* root;

TreeNode\* insert(TreeNode\* node, int key) {

if (node == nullptr) {

return new TreeNode(key);

}

if (key < node->key) {

node->left = insert(node->left, key);

} else if (key > node->key) {

node->right = insert(node->right, key);

}

return node;

}

TreeNode\* remove(TreeNode\* node, int key) {

if (node == nullptr) {

return nullptr;

}

if (key < node->key) {

node->left = remove(node->left, key);

} else if (key > node->key) {

node->right = remove(node->right, key);

} else {

if (node->left == nullptr && node->right == nullptr) {

delete node;

return nullptr;

} else if (node->left == nullptr) {

TreeNode\* temp = node->right;

delete node;

return temp;

} else if (node->right == nullptr) {

TreeNode\* temp = node->left;

delete node;

return temp;

} else {

TreeNode\* minNode = findMin(node->right);

node->key = minNode->key;

node->right = remove(node->right, minNode->key);

}

}

return node;

}

bool search(TreeNode\* node, int key) {

if (node == nullptr) {

return false;

}

if (key == node->key) {

return true;

} else if (key < node->key) {

return search(node->left, key);

} else {

return search(node->right, key);

}

}

void preOrder(TreeNode\* node) {

if (node != nullptr) {

qDebug() << node->key;

preOrder(node->left);

preOrder(node->right);

}

}

void inOrder(TreeNode\* node) {

if (node != nullptr) {

inOrder(node->left);

qDebug() << node->key;

inOrder(node->right);

}

}

void postOrder(TreeNode\* node) {

if (node != nullptr) {

postOrder(node->left);

postOrder(node->right);

qDebug() << node->key;

}

}

TreeNode\* balance(TreeNode\* node) {

if (node == nullptr) {

return nullptr;

}

QVector<int> keys;

collectKeys(node, keys);

return buildBalancedTree(keys, 0, keys.size() - 1);

}

void collectKeys(TreeNode\* node, QVector<int>& keys) {

if (node != nullptr) {

collectKeys(node->left, keys);

keys.append(node->key);

collectKeys(node->right, keys);

}

}

TreeNode\* buildBalancedTree(const QVector<int>& keys, int left, int right) {

if (left > right) {

return nullptr;

}

int middle = (left + right) / 2;

TreeNode\* node = new TreeNode(keys[middle]);

node->left = buildBalancedTree(keys, left, middle - 1);

node->right = buildBalancedTree(keys, middle + 1, right);

return node;

}

TreeNode\* findMin(TreeNode\* node) {

while (node->left != nullptr) {

node = node->left;

}

return node;

}

void drawTree(QPainter& painter, int x, int y, TreeNode\* node, int level) {

if (node == nullptr) {

return;

}

int radius = 25;

int spacing = 75;

int levelHeight = 80;

painter.drawEllipse(QPointF(x, y), radius, radius);

painter.drawText(QRectF(x - radius, y - radius, 2 \* radius, 2 \* radius), Qt::AlignCenter, QString::number(node->key));

if (node->left != nullptr) {

int leftX = x - spacing \* (1 << (level - 1));

int leftY = y + levelHeight;

painter.drawLine(x - radius, y + radius, leftX, leftY);

drawTree(painter, leftX, leftY, node->left, level + 1);

}

if (node->right != nullptr) {

int rightX = x + spacing \* (1 << (level - 1));

int rightY = y + levelHeight;

painter.drawLine(x + radius, y + radius, rightX, rightY);

drawTree(painter, rightX, rightY, node->right, level + 1);

}

}

};

int main(int argc, char\* argv[]) {

QApplication app(argc, argv);

BinaryTreeWidget binaryTreeWidget;

QPushButton insertButton("Insert");

QObject::connect(&insertButton, &QPushButton::clicked, [&binaryTreeWidget]() {

bool ok;

int key = QInputDialog::getInt(nullptr, "Insert Node", "Enter key:", 0, INT\_MIN, INT\_MAX, 1, &ok);

if (ok) {

binaryTreeWidget.insertNode(key);

}

});

QPushButton deleteButton("Delete");

QObject::connect(&deleteButton, &QPushButton::clicked, [&binaryTreeWidget]() {

bool ok;

int key = QInputDialog::getInt(nullptr, "Delete Node", "Enter key:", 0, INT\_MIN, INT\_MAX, 1, &ok);

if (ok) {

binaryTreeWidget.deleteNode(key);

}

});

QPushButton searchButton("Search");

QObject::connect(&searchButton, &QPushButton::clicked, [&binaryTreeWidget]() {

bool ok;

int key = QInputDialog::getInt(nullptr, "Search Node", "Enter key:", 0, INT\_MIN, INT\_MAX, 1, &ok);

if (ok) {

bool found = binaryTreeWidget.searchNode(key);

QString message = found ? "Node found!" : "Node not found!";

QMessageBox::information(nullptr, "Search Result", message);

}

});

QPushButton preOrderButton("Pre-order");

QObject::connect(&preOrderButton, &QPushButton::clicked, [&binaryTreeWidget]() {

binaryTreeWidget.preOrderTraversal();

});

QPushButton inOrderButton("In-order");

QObject::connect(&inOrderButton, &QPushButton::clicked, [&binaryTreeWidget]() {

binaryTreeWidget.inOrderTraversal();

});

QPushButton postOrderButton("Post-order");

QObject::connect(&postOrderButton, &QPushButton::clicked, [&binaryTreeWidget]() {

binaryTreeWidget.postOrderTraversal();

});

QPushButton balanceButton("Balance");

QObject::connect(&balanceButton, &QPushButton::clicked, [&binaryTreeWidget]() {

binaryTreeWidget.balanceTree();

});

QVBoxLayout layout;

layout.addWidget(&binaryTreeWidget);

QHBoxLayout buttonLayout;

buttonLayout.addWidget(&insertButton);

buttonLayout.addWidget(&deleteButton);

buttonLayout.addWidget(&searchButton);

buttonLayout.addWidget(&preOrderButton);

buttonLayout.addWidget(&inOrderButton);

buttonLayout.addWidget(&postOrderButton);

buttonLayout.addWidget(&balanceButton);

QWidget mainWidget;

mainWidget.setLayout(&buttonLayout);

layout.addWidget(&mainWidget);

QWidget window;

window.setLayout(&layout);

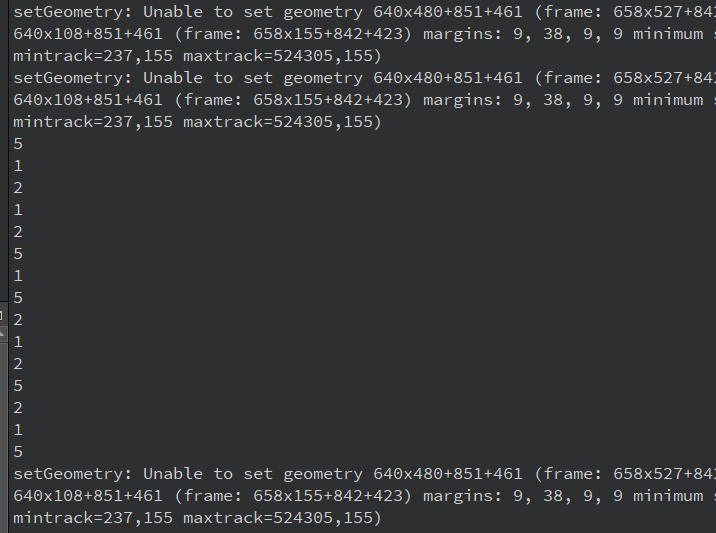
window.setWindowTitle("Binary Tree");

window.show();

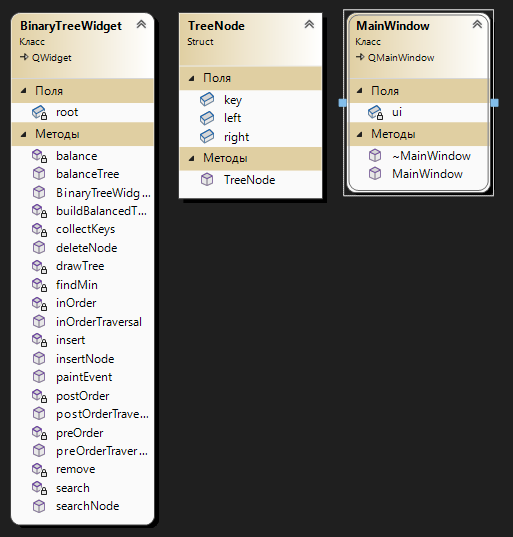
return app.exec();

}

**Вывод программы**

****

**UML диаграмма**

****

**Вывод**

Программа успешно выполняет свою задачу.

**Заключение**

Для решения задачи потребовались знания языка программирования, а конкретнее C++. Программа выполняет те условия, что были указаны в постановке задачи и работает без проблем.