Лабораторная работа №5

Выполнила: Аниськова Я.Ю.

Группа: 23ПИнж1з

Задание. Реализовать бинарное поисковое дерево (ООП):

- базовые операции: добавление, поиск, удаление (левое и правое);

- 3 способа обхода дерева: прямой, обратный, симметричный;

- методы для определения глубины, высоты и уровня узла;

- вывод дерева на экран

Реализация:

#include <iostream>

#include <queue>

#include <iomanip>

class TreeNode {

public:

int value;

TreeNode\* left;

TreeNode\* right;

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

};

class BinarySearchTree {

private:

TreeNode\* root;

// Вспомогательные функции для рекурсивных операций

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

if (!node) return new TreeNode(value);

if (value < node->value) node->left = insert(node->left, value);

else if (value > node->value) node->right = insert(node->right, value);

return node;

}

TreeNode\* search(TreeNode\* node, int value) {

if (!node || node->value == value) return node;

if (value < node->value) return search(node->left, value);

return search(node->right, value);

}

TreeNode\* deleteNode(TreeNode\* node, int value) {

if (!node) return node;

if (value < node->value) node->left = deleteNode(node->left, value);

else if (value > node->value) node->right = deleteNode(node->right, value);

else {

// Узел с одним или нулем потомков

if (!node->left) {

TreeNode\* temp = node->right;

delete node;

return temp;

}

else if (!node->right) {

TreeNode\* temp = node->left;

delete node;

return temp;

}

// Узел с двумя потомками

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

node->value = temp->value;

node->right = deleteNode(node->right, temp->value);

}

return node;

}

TreeNode\* findMin(TreeNode\* node) {

while (node && node->left) node = node->left;

return node;

}

void preOrder(TreeNode\* node) {

if (!node) return;

std::cout << node->value << " ";

preOrder(node->left);

preOrder(node->right);

}

void inOrder(TreeNode\* node) {

if (!node) return;

inOrder(node->left);

std::cout << node->value << " ";

inOrder(node->right);

}

void postOrder(TreeNode\* node) {

if (!node) return;

postOrder(node->left);

postOrder(node->right);

std::cout << node->value << " ";

}

int depth(TreeNode\* node) {

if (!node) return 0;

return 1 + std::max(depth(node->left), depth(node->right));

}

int nodeDepth(TreeNode\* node, int value, int depth) {

if (!node) return -1;

if (node->value == value) return depth;

int downlevel = nodeDepth(node->left, value, depth + 1);

if (downlevel != -1) return downlevel;

return nodeDepth(node->right, value, depth + 1);

}

void printTree(TreeNode\* root, int space = 0, int height = 10) {

if (!root) return;

space += height;

printTree(root->right, space);

std::cout << std::endl;

for (int i = height; i < space; i++) std::cout << ' ';

std::cout << root->value << "\n";

printTree(root->left, space);

}

public:

BinarySearchTree() : root(nullptr) {}

void insert(int value) {

root = insert(root, value);

}

TreeNode\* search(int value) {

return search(root, value);

}

void deleteNode(int value) {

root = deleteNode(root, value);

}

void preOrder() {

preOrder(root);

std::cout << std::endl;

}

void inOrder() {

inOrder(root);

std::cout << std::endl;

}

void postOrder() {

postOrder(root);

std::cout << std::endl;

}

int depth() {

return depth(root);

}

int nodeDepth(int value) {

return nodeDepth(root, value, 0);

}

void printTree() {

printTree(root);

}

};

int main() {

setlocale(LC\_ALL, "Russian");

BinarySearchTree bst;

// Добавление узлов

bst.insert(50);

bst.insert(30);

bst.insert(20);

bst.insert(40);

bst.insert(70);

bst.insert(60);

bst.insert(80);

std::cout << "Дерево:\n";

bst.printTree();

std::cout << "Обход в прямом порядке:\n";

bst.preOrder();

std::cout << "Обход в симметричном порядке:\n";

bst.inOrder();

std::cout << "Обход в обратном порядке:\n";

bst.postOrder();

std::cout << "Глубина дерева: " << bst.depth() << std::endl;

int value = 40;

std::cout << "Глубина узла с значением " << value << ": " << bst.nodeDepth(value) << std::endl;

std::cout << "Удаление узла с значением 20\n";

bst.deleteNode(20);

bst.printTree();

std::cout << "Удаление узла с значением 30\n";

bst.deleteNode(30);

bst.printTree();

std::cout << "Удаление узла с значением 50\n";

bst.deleteNode(50);

bst.printTree();

return 0;

}

