#include <iostream>

#include <algorithm>

using namespace std;

class Node {

public:

int key;

Node \*left;

Node \*right;

int height;

Node(int k) : key(k), left(nullptr), right(nullptr), height(1) {}

};

class AVLTree {

private:

Node\* root;

// Hàm tính chiều cao của nút

int height(Node\* node) {

if (node == nullptr)

return 0;

return node->height;

}

// Hàm tính hệ số cân bằng

int balanceFactor(Node\* node) {

if (node == nullptr)

return 0;

return height(node->left) - height(node->right);

}

// Cập nhật chiều cao của nút

void updateHeight(Node\* node) {

if (node != nullptr)

node->height = 1 + max(height(node->left), height(node->right));

}

// Quay phải

Node\* rightRotate(Node\* y) {

Node\* x = y->left;

Node\* T2 = x->right;

x->right = y;

y->left = T2;

updateHeight(y);

updateHeight(x);

return x;

}

// Quay trái

Node\* leftRotate(Node\* x) {

Node\* y = x->right;

Node\* T2 = y->left;

y->left = x;

x->right = T2;

updateHeight(x);

updateHeight(y);

return y;

}

// Chèn nút vào cây AVL

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

// 1. Thực hiện chèn như BST thông thường

if (node == nullptr)

return new Node(key);

if (key < node->key)

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

else if (key > node->key)

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

else

return node; // Không cho phép giá trị trùng

// 2. Cập nhật chiều cao của nút cha

updateHeight(node);

// 3. Kiểm tra hệ số cân bằng và cân bằng lại nếu cần

int balance = balanceFactor(node);

// Trường hợp Left Left

if (balance > 1 && key < node->left->key)

return rightRotate(node);

// Trường hợp Right Right

if (balance < -1 && key > node->right->key)

return leftRotate(node);

// Trường hợp Left Right

if (balance > 1 && key > node->left->key) {

node->left = leftRotate(node->left);

return rightRotate(node);

}

// Trường hợp Right Left

if (balance < -1 && key < node->right->key) {

node->right = rightRotate(node->right);

return leftRotate(node);

}

return node;

}

// Tìm nút có giá trị nhỏ nhất

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current->left != nullptr)

current = current->left;

return current;

}

// Xóa nút khỏi cây AVL

Node\* deleteNode(Node\* root, int key) {

// 1. Thực hiện xóa như BST thông thường

if (root == nullptr)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

// Nút có một con hoặc không có con

if ((root->left == nullptr) || (root->right == nullptr)) {

Node\* temp = root->left ? root->left : root->right;

// Trường hợp không có con

if (temp == nullptr) {

temp = root;

root = nullptr;

} else { // Trường hợp một con

\*root = \*temp;

}

delete temp;

} else {

// Nút có hai con: lấy nút nhỏ nhất từ cây con phải

Node\* temp = minValueNode(root->right);

// Sao chép giá trị của nút nhỏ nhất vào nút hiện tại

root->key = temp->key;

// Xóa nút nhỏ nhất

root->right = deleteNode(root->right, temp->key);

}

}

// Nếu cây chỉ có một nút thì trả về

if (root == nullptr)

return root;

// 2. Cập nhật chiều cao của nút hiện tại

updateHeight(root);

// 3. Kiểm tra hệ số cân bằng và cân bằng lại nếu cần

int balance = balanceFactor(root);

// Trường hợp Left Left

if (balance > 1 && balanceFactor(root->left) >= 0)

return rightRotate(root);

// Trường hợp Left Right

if (balance > 1 && balanceFactor(root->left) < 0) {

root->left = leftRotate(root->left);

return rightRotate(root);

}

// Trường hợp Right Right

if (balance < -1 && balanceFactor(root->right) <= 0)

return leftRotate(root);

// Trường hợp Right Left

if (balance < -1 && balanceFactor(root->right) > 0) {

root->right = rightRotate(root->right);

return leftRotate(root);

}

return root;

}

// Tìm kiếm nút trong cây

Node\* search(Node\* root, int key) {

if (root == nullptr || root->key == key)

return root;

if (root->key < key)

return search(root->right, key);

return search(root->left, key);

}

// Duyệt cây theo thứ tự in-order

void inOrder(Node\* root) {

if (root != nullptr) {

inOrder(root->left);

cout << root->key << " ";

inOrder(root->right);

}

}

// Duyệt cây theo thứ tự pre-order

void preOrder(Node\* root) {

if (root != nullptr) {

cout << root->key << " ";

preOrder(root->left);

preOrder(root->right);

}

}

// Duyệt cây theo thứ tự post-order

void postOrder(Node\* root) {

if (root != nullptr) {

postOrder(root->left);

postOrder(root->right);

cout << root->key << " ";

}

}

public:

AVLTree() : root(nullptr) {}

// Thêm khóa vào cây

void insert(int key) {

root = insert(root, key);

}

// Xóa khóa khỏi cây

void deleteNode(int key) {

root = deleteNode(root, key);

}

// Tìm kiếm khóa trong cây

bool search(int key) {

return search(root, key) != nullptr;

}

// In cây theo thứ tự in-order

void inOrder() {

inOrder(root);

cout << endl;

}

// In cây theo thứ tự pre-order

void preOrder() {

preOrder(root);

cout << endl;

}

// In cây theo thứ tự post-order

void postOrder() {

postOrder(root);

cout << endl;

}

// Lấy chiều cao của cây

int getHeight() {

return height(root);

}

};

// Hàm main để kiểm tra

int main() {

AVLTree tree;

// Thêm các nút vào cây

tree.insert(10);

tree.insert(20);

tree.insert(30);

tree.insert(40);

tree.insert(50);

tree.insert(25);

cout << "Duyet theo thu tu in-order: ";

tree.inOrder();

cout << "Duyet theo thu tu pre-order: ";

tree.preOrder();

cout << "Duyet theo thu tu post-order: ";

tree.postOrder();

cout << "Chieu cao cua cay: " << tree.getHeight() << endl;

// Tìm kiếm nút

cout << "Tim kiem 30: " << (tree.search(30) ? "Tim thay" : "Khong tim thay") << endl;

cout << "Tim kiem 35: " << (tree.search(35) ? "Tim thay" : "Khong tim thay") << endl;

// Xóa nút

cout << "Xoa nut 30" << endl;

tree.deleteNode(30);

cout << "Duyet theo thu tu in-order sau khi xoa: ";

tree.inOrder();

return 0;

}