**#include <bits/stdc++.h>**

**using namespace std;**

**#define M 1000000007**

**struct node {**

**int data;**

**int height;**

**node\* left, \*right;**

**node(int val);**

**};**

**node::node(int val) {**

**this->data = val;**

**this->left = nullptr;**

**this->right = nullptr;**

**this->height = 1;**

**}**

**pair<int, int> giveHeight(node\* root) {**

**int leftHeight = 0, rightHeight = 0;**

**if(root->left != nullptr) {**

**leftHeight = root->left->height;**

**}**

**if(root->right != nullptr) {**

**rightHeight = root->right->height;**

**}**

**return {leftHeight, rightHeight};**

**}**

**node\* leftRotate(node\* root) {**

**node\* temp = root->right;**

**node\* child = temp->left;**

**temp->left = root;**

**root->right = child;**

**auto p = giveHeight(root);**

**root->height = max(p.first, p.second) + 1;**

**p = giveHeight(temp);**

**temp->height = max(p.first, p.second) + 1;**

**return temp;**

**}**

**node\* rightRotate(node\* root) {**

**node\* temp = root->left;**

**node\* child = temp->right;**

**temp->right = root;**

**root->left = child;**

**auto p = giveHeight(root);**

**root->height = max(p.first, p.second) + 1;**

**p = giveHeight(temp);**

**temp->height = max(p.first, p.second) + 1;**

**return temp;**

**}**

**// Balanced Binary search tree insertion**

**node\* smartInsert(node\* root, int val) {**

**if(root == nullptr) {**

**root = new node(val);**

**return root;**

**} else if(root->data < val) {**

**root->right = smartInsert(root->right, val);**

**} else {**

**root->left = smartInsert(root->left, val);**

**}**

**auto p1 = giveHeight(root);**

**// Balanced case**

**if(abs(p1.first - p1.second) <= 1) {**

**root->height = max(p1.first, p1.second) + 1;**

**return root;**

**}**

**// Disbalanced case**

**// left disbalance**

**if(p1.first - p1.second > 0) {**

**node\* child = root->left;**

**auto p2 = giveHeight(child);**

**// left left case**

**if(p2.first > p2.second) {**

**root = rightRotate(root);**

**} else {**

**// left right case**

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

**root = rightRotate(root);**

**}**

**} else {**

**node\* child = root->right;**

**auto p2 = giveHeight(child);**

**if(p2.first < p2.second) {**

**// right right case**

**root = leftRotate(root);**

**} else {**

**// right left case**

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

**root = leftRotate(root);**

**}**

**}**

**return root;**

**}**

**node\* insert(node\* root, int val) {**

**// O(height)**

**if(root == nullptr) {**

**root = new node(val);**

**} else if(root->data > val) {**

**root->left = insert(root->left, val);**

**} else {**

**root->right = insert(root->right, val);**

**}**

**return root;**

**}**

**bool search(node\* root, int val) {**

**// O(height)**

**if(root == nullptr) {**

**return false;**

**} else if(root->data > val) {**

**return search(root->left, val);**

**} else if(root->data < val) {**

**return search(root->right, val);**

**} else {**

**return true;**

**}**

**}**

**node\* del(node\* root, int val) {**

**// O(height)**

**if(root == nullptr) {**

**return root;**

**} else if(root->data > val) {**

**root->left = del(root->left, val);**

**return root;**

**} else if(root->data < val) {**

**root->right = del(root->right, val);**

**return root;**

**} else {**

**// First case Is root a leaf?**

**if(root->left == nullptr and root->right == nullptr) {**

**delete root;**

**return nullptr;**

**}**

**// Second Case when root has only one child**

**if(root->left == nullptr) {**

**node\* temp = root->right;**

**delete root;**

**return temp;**

**}**

**if(root->right == nullptr) {**

**node\* temp = root->left;**

**delete root;**

**return temp;**

**}**

**// 3rd case when root has both the childs;**

**node\* rightmost = root->left;**

**while(rightmost->right != nullptr) {**

**rightmost = rightmost->right;**

**}**

**root->data = rightmost->data;**

**del(root->left, rightmost->data);**

**return root;**

**}**

**}**

**int main() {**

**node\* root = nullptr;**

**root = smartInsert(root, 5);**

**root = smartInsert(root, 10);**

**root = smartInsert(root, 4);**

**root = smartInsert(root, 7);**

**cout<< search(root, 10) << endl;**

**cout<< search(root, 4) << endl;**

**cout<< search(root, 7) << endl;**

**root = del(root, 5);**

**cout<< search(root, 5) << endl;**

**cout<< search(root, 4) << endl;**

**cout<< search(root, 7) << endl;**

**root = del(root, 7);**

**cout<< search(root, 4) << endl;**

**cout<< search(root, 7) << endl;**

**return 0;**

**}**