#include <iostream>

#include <queue>

using namespace std;

// Node structure for the binary search tree

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

TreeNode(int val) {

data = val;

left = nullptr;

right = nullptr;

}

};

// Binary Search Tree class

class BST {

private:

TreeNode\* root;

// Helper function to insert a value into the BST

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

if (root == nullptr) {

return new TreeNode(val);

}

if (val < root->data) {

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

} else {

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

}

return root;

}

// Helper function to find the number of nodes in the longest path from root

int longestPath(TreeNode\* root) {

if (root == nullptr) return 0;

return 1 + max(longestPath(root->left), longestPath(root->right));

}

// Helper function to find the minimum data value in the tree

int findMin(TreeNode\* root) {

if (root->left == nullptr) return root->data;

return findMin(root->left);

}

// Helper function to swap left and right pointers at every node

void swapPointers(TreeNode\* root) {

if (root == nullptr) return;

swap(root->left, root->right);

swapPointers(root->left);

swapPointers(root->right);

}

// Helper function to search for a value in the BST

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

if (root == nullptr) return false;

if (root->data == val) return true;

if (val < root->data) {

return search(root->left, val);

} else {

return search(root->right, val);

}

}

// Helper function to delete a node with given value from the BST

TreeNode\* deleteNode(TreeNode\* root, int val) {

if (root == nullptr) return nullptr;

if (val < root->data) {

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

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

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

} else {

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

delete root;

return nullptr;

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

TreeNode\* temp = root->right;

delete root;

return temp;

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

TreeNode\* temp = root->left;

delete root;

return temp;

} else {

TreeNode\* temp = root->right;

while (temp->left != nullptr) {

temp = temp->left;

}

root->data = temp->data;

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

}

}

return root;

}

// Helper function to perform inorder traversal of the tree

void inorderTraversal(TreeNode\* root) {

if (root == nullptr) return;

inorderTraversal(root->left);

cout << root->data << " ";

inorderTraversal(root->right);

}

public:

// Constructor

BST() {

root = nullptr;

}

// Function to insert a value into the BST

void insert(int val) {

root = insert(root, val);

}

// Function to find the number of nodes in the longest path from root

int longestPathFromRoot() {

return longestPath(root);

}

// Function to find the minimum data value in the tree

int findMinimum() {

if (root == nullptr) {

cout << "Tree is empty\n";

return -1;

}

return findMin(root);

}

// Function to change the tree so that the roles of the left and right pointers are swapped

void swapPointersAtNodes() {

swapPointers(root);

}

// Function to search for a value in the BST

bool search(int val) {

return search(root, val);

}

// Function to delete a node with given value from the BST

void deleteNode(int val) {

root = deleteNode(root, val);

}

// Function to display the binary search tree (inorder traversal)

void display() {

cout << "Binary Search Tree (Inorder Traversal): ";

inorderTraversal(root);

cout << endl;

}

};

int main() {

BST tree;

char choice;

do {

cout << "1. Insert a node\n";

cout << "2. Delete a node\n";

cout << "3. Find a node\n";

cout << "4. Find the number of nodes in the longest path from root\n";

cout << "5. Find the minimum data value in the tree\n";

cout << "6. Change the tree so that the roles of the left and right pointers are swapped at every node\n";

cout << "7. Search for a value\n";

cout << "8. Display the tree\n";

cout << "9. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case '1': {

int value;

cout << "Enter value to insert into the tree: ";

cin >> value;

tree.insert(value);

break;

}

case '2': {

int value;

cout << "Enter value to delete from the tree: ";

cin >> value;

tree.deleteNode(value);

break;

}

case '3': {

int value;

cout << "Enter value to find in the tree: ";

cin >> value;

cout << (tree.search(value) ? "Found" : "Not Found") << endl;

break;

}

case '4':

cout << "Number of nodes in longest path from root: " << tree.longestPathFromRoot() << endl;

break;

case '5':

cout << "Minimum data value in the tree: " << tree.findMinimum() << endl;

break;

case '6':

tree.swapPointersAtNodes();

cout << "Tree nodes swapped successfully!\n";

break;

case '7': {

int value;

cout << "Enter value to search in the tree: ";

cin >> value;

cout << (tree.search(value) ? "Found" : "Not Found") << endl;

break;

}

case '8':

tree.display();

break;

case '9':

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice. Please try again.\n";

}

} while (choice != '9');

return 0;

}