**Practical GroupB\_06**

**Aim:**

Beginning with an empty binary search tree, Construct binary search tree by inserting the values in the order given. After constructing a binary tree - i. Insert new node, ii. Find number of nodes in longest path from root, iii. Minimum data value found in the tree, iv. Change a tree so that the roles of the left and right pointers are swapped at every node, v. Search a value

**Code:**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* left;

Node\* right;

Node(int value) : data(value), left(nullptr), right(nullptr) {}

};

class BinarySearchTree {

public:

BinarySearchTree() : root(nullptr) {}

void insert(int value);

int longestPath();

int findMinimum();

void swapPointers();

bool search(int value);

private:

Node\* root;

Node\* insert(Node\* node, int value);

int longestPath(Node\* node);

int findMinimum(Node\* node);

void swapPointers(Node\* node);

bool search(Node\* node, int value);

};

void BinarySearchTree::insert(int value) {

root = insert(root, value);

}

Node\* BinarySearchTree::insert(Node\* node, int value) {

if (node == nullptr) {

return new Node(value);

}

if (value < node->data) {

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

} else {

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

}

return node;

}

int BinarySearchTree::longestPath() {

return longestPath(root);

}

int BinarySearchTree::longestPath(Node\* node) {

if (node == nullptr) {

return 0;

}

int leftPath = longestPath(node->left);

int rightPath = longestPath(node->right);

return 1 + max(leftPath, rightPath);

}

int BinarySearchTree::findMinimum() {

if (root == nullptr) {

cout << "Tree is empty." << endl;

return -1; // Assuming -1 represents an invalid minimum value

}

Node\* current = root;

while (current->left != nullptr) {

current = current->left;

}

return current->data;

}

void BinarySearchTree::swapPointers() {

swapPointers(root);

}

void BinarySearchTree::swapPointers(Node\* node) {

if (node == nullptr) {

return;

}

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

swapPointers(node->left);

swapPointers(node->right);

}

bool BinarySearchTree::search(int value) {

return search(root, value);

}

bool BinarySearchTree::search(Node\* node, int value) {

if (node == nullptr) {

return false;

}

if (node->data == value) {

return true;

} else if (value < node->data) {

return search(node->left, value);

} else {

return search(node->right, value);

}

}

int main() {

BinarySearchTree bst;

int choice;

while (true) {

cout << "-------------------------" << endl;

cout << "Binary Search Tree Menu" << endl;

cout << "-------------------------" << endl;

cout << "1. Insert a new node" << endl;

cout << "2. Find the number of nodes in the longest path from the root" << endl;

cout << "3. Find the minimum value in the tree" << endl;

cout << "4. Swap left and right pointers at every node" << endl;

cout << "5. Search for a value" << endl;

cout << "6. Quit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

int value;

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

cin >> value;

bst.insert(value);

break;

}

case 2: {

int longest = bst.longestPath();

cout << "Number of nodes in the longest path from the root: " << longest << endl;

break;

}

case 3: {

int minimum = bst.findMinimum();

if (minimum != -1) {

cout << "Minimum value in the tree: " << minimum << endl;

}

break;

}

case 4: {

bst.swapPointers();

cout << "Pointers swapped successfully." << endl;

break;

}

case 5: {

int value;

cout << "Enter the value to search for: ";

cin >> value;

bool found = bst.search(value);

if (found) {

cout << "Value found in the tree." << endl;

} else {

cout << "Value not found in the tree." << endl;

}

break;

}

case 6: {

cout << "Thanks for using this program!" << endl;

return 0;

}

default:

cout << "Invalid choice. Please try again." << endl;

}

cout << endl;

}

}