**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**

**Program :**

#include<iostream>

#include<queue>

using namespace std;

struct Node{

int data;

Node\* left;

Node\* right;

};

Node\* createNode(int data){

Node\* newNode = new Node();

newNode->data = data;

newNode->left = newNode->right = NULL;

return newNode;

}

Node\* insertNode(Node\* root,int data){

if(root == NULL){

root = createNode(data);

}

else if(data <= root->data){

root->left = insertNode(root->left,data);

}

else{

root->right = insertNode(root->right,data);

}

return root;

}

int findLongestPath(Node\* root){

if(root == NULL){

return 0;

}

int leftHeight = findLongestPath(root->left);

int rightHeight = findLongestPath(root->right);

return max(leftHeight,rightHeight)+1;

}

Node\* swapLeftRightPointers(Node\* root){

if(root == NULL)

return NULL;

Node\* temp = root->left;

root->left = root->right;

root->right = temp;

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

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

return root;

}

int findMinimumValue(Node \*root){

//Check whether tree is empty

if(root == NULL) {

printf("Tree is empty\n");

return 0;

}

else{

int leftMin, rightMin;

//Min will store temp's data

int min = root->data;

//It will find smallest element in left subtree

if(root->left != NULL){

leftMin = findMinimumValue(root->left);

//If min is greater than leftMin then store the value of leftMin into min

min = (min > leftMin) ? leftMin : min;

}

//It will find smallest element in right subtree

if(root->right != NULL){

rightMin = findMinimumValue(root->right);

//If min is greater than rightMin then store the value of rightMin into min

min = (min > rightMin) ? rightMin : min;

}

return min;

}

}

bool searchValue(Node\* root, int val)

{

if (root == NULL)

return false;

if (root->data == val)

return true;

/\* then recur on left subtree \*/

bool res1 = searchValue(root->left, val);

// node found, no need to look further

if(res1) return true;

/\* node is not found in left,

so recur on right subtree \*/

bool res2 = searchValue(root->right, val);

return res2;

}

void displayTree(Node\* root){

if(root == NULL)

return;

queue<Node\*>q;

q.push(root);

while(!q.empty()){

int levelSize = q.size();

for(int i=0;i<levelSize;i++){

Node\* current = q.front();

q.pop();

cout<<current->data <<" ";

if(current->left)

q.push(current->left);

if(current->right)

q.push(current->right);

}

cout<<endl;

}

}

void displayMenu(){

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

cout<<"1. Insert a Node"<<endl;

cout<<"2. Find Number of nodes in Longest Path of Root"<<endl;

cout<<"3. Find Minimum Value"<<endl;

cout<<"4. Swap Left and Right Pointers at Every Node"<<endl;

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

cout<<"6. Display the Tree"<<endl;

cout<<"7. Exit"<<endl;

cout<<"Enter your choice: ";

}

int main(){

Node\* root = NULL;

int choice;

int value;

do{

displayMenu();

cin>>choice;

switch(choice){

case 1:

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

cin>>value;

root = insertNode(root,value);

cout<<"Node inserted successfully!"<<endl;

break;

case 2:

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

break;

case 3:

cout<<"Minimum value in the tree: "<<findMinimumValue(root)<<endl;

break;

case 4:

root = swapLeftRightPointers(root);

cout<<"Left and right pointers swapped at every node."<<endl;

break;

case 5:

cout<<"Enter a value to search: ";

cin>>value;

if(searchValue(root,value))

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

else

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

break;

case 6:

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

displayTree(root);

break;

case 7:

cout<<"Exiting the program."<<endl;

break;

default:

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

}

cout<<endl;

}while(choice != 7);

return 0;

}