Problem statement:

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

#include <iostream>

using namespace std;

struct Bstnode {

int data;

Bstnode\* left = NULL;

Bstnode\* right = NULL;

};

class Btree {

public:

Bstnode\* root;

Btree() {

root = NULL;

}

Bstnode\* GetNewNode(int in\_data) {

Bstnode\* ptr = new Bstnode();

ptr->data = in\_data;

return ptr;

}

Bstnode\* insert(Bstnode\* temp, int in\_data) {

if (temp == NULL) {

return GetNewNode(in\_data);

}

if (in\_data < temp->data) {

temp->left = insert(temp->left, in\_data);

} else {

temp->right = insert(temp->right, in\_data);

}

return temp;

}

void addNode() {

int value;

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

cin >> value;

root = insert(root, value);

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

}

int findDepth(Bstnode\* temp) {

if (temp == NULL)

return 0;

return max(findDepth(temp->left), findDepth(temp->right)) + 1;

}

void findMinValue() {

if (root == NULL) {

cout << "The tree is empty!" << endl;

return;

}

Bstnode\* temp = root;

while (temp->left != NULL) {

temp = temp->left;

}

cout << "Minimum value in the tree: " << temp->data << endl;

}

void mirrorTree(Bstnode\* temp) {

if (temp == NULL)

return;

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

mirrorTree(temp->left);

mirrorTree(temp->right);

}

void mirror() {

if (root == NULL) {

cout << "The tree is empty!" << endl;

return;

}

mirrorTree(root);

cout << "Tree mirrored successfully!" << endl;

}

bool search(Bstnode\* temp, int in\_data) {

if (temp == NULL)

return false;

if (temp->data == in\_data)

return true;

if (in\_data < temp->data)

return search(temp->left, in\_data);

return search(temp->right, in\_data);

}

void searchValue() {

int value;

cout << "Enter value to search: ";

cin >> value;

if (search(root, value)) {

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

} else {

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

}

}

void inorder(Bstnode\* temp) {

if (temp == NULL)

return;

inorder(temp->left);

cout << temp->data << " ";

inorder(temp->right);

}

void display() {

if (root == NULL) {

cout << "The tree is empty!" << endl;

return;

}

cout << "Inorder traversal of the tree: ";

inorder(root);

cout << endl;

}

};

int main() {

Btree tree;

int choice;

while (true) {

cout << "\nMenu:\n"

<< "1. Insert new node\n"

<< "2. Find number of nodes in the longest path (depth)\n"

<< "3. Find minimum data value in the tree\n"

<< "4. Mirror the tree\n"

<< "5. Search for a value\n"

<< "6. Display tree\n"

<< "7. Exit\n"

<< "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

tree.addNode();

break;

case 2:

cout << "Number of nodes in the longest path (depth): " << tree.findDepth(tree.root) << endl;

break;

case 3:

tree.findMinValue();

break;

case 4:

tree.mirror();

break;

case 5:

tree.searchValue();

break;

case 6:

tree.display();

break;

case 7:

cout << "Exiting program!" << endl;

return 0;

default:

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

}

}

return 0;

}

