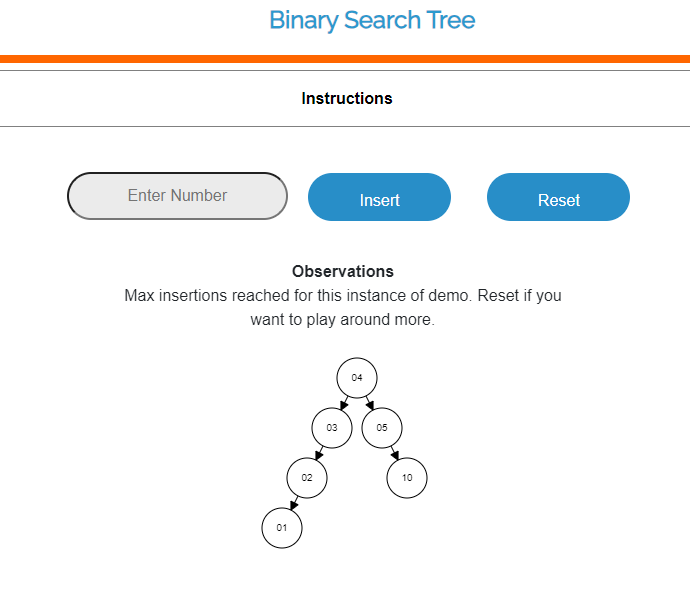
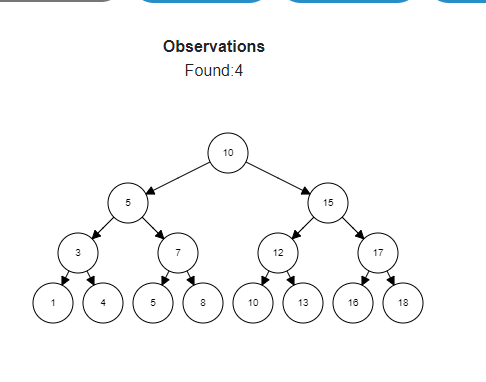
**15B17CI371 – Data Structures Lab**

**ODD 2024**

**Week 7-LAB B**

**Practice Lab**





1.

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node \*left, \*right;

Node(int value) {

data = value;

left = NULL;

right = NULL;

}

};

void printCurrentLevel(Node\* root, int level);

int height(Node\* node);

void printLevelOrder(Node\* root) {

int h = height(root);

for (int i = 1; i <= h; i++)

printCurrentLevel(root, i);

}

void printCurrentLevel(Node\* root, int level) {

if (root == NULL)

return;

if (level == 1)

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

else if (level > 1) {

printCurrentLevel(root->left, level - 1);

printCurrentLevel(root->right, level - 1);

}

}

int height(Node\* node) {

if (node == NULL)

return 0;

else {

int lheight = height(node->left);

int rheight = height(node->right);

return (lheight > rheight) ? (lheight + 1) :

(rheight + 1);

}

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

printLevelOrder(root);

return 0;

}



#include <bits/stdc++.h>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int v)

{

this->data = v;

this->left = this->right = NULL;

}

};

void printInorder(Node\* node)

{

if (node == NULL)

return;

printInorder(node->left);

cout << node->data << " ";

printInorder(node->right);

}

int main()

{

Node\* root = new Node(100);

root->left = new Node(20);

root->right = new Node(200);

root->left->left = new Node(10);

root->left->right = new Node(30);

root->right->left = new Node(150);

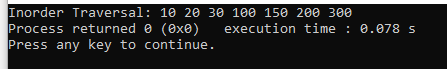
root->right->right = new Node(300);

cout << "Inorder Traversal: ";

printInorder(root);

return 0;

}



#include <bits/stdc++.h>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int v)

{

this->data = v;

this->left = this->right = NULL;

}

};

void printPreOrder(Node\* node)

{

if (node == NULL)

return;

cout << node->data << " ";

printPreOrder(node->left);

printPreOrder(node->right);

}

int main()

{

Node\* root = new Node(100);

root->left = new Node(20);

root->right = new Node(200);

root->left->left = new Node(10);

root->left->right = new Node(30);

root->right->left = new Node(150);

root->right->right = new Node(300);

cout << "Preorder Traversal: ";

printPreOrder(root);

return 0;

}



#include <bits/stdc++.h>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int v)

{

this->data = v;

this->left = this->right = NULL;

}

};

void printPostOrder(Node\* node)

{

if (node == NULL)

return;

printPostOrder(node->left);

printPostOrder(node->right);

cout << node->data << " ";

}

int main()

{

Node\* root = new Node(100);

root->left = new Node(20);

root->right = new Node(200);

root->left->left = new Node(10);

root->left->right = new Node(30);

root->right->left = new Node(150);

root->right->right = new Node(300);

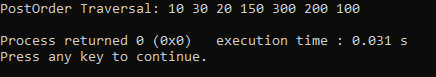
cout << "PostOrder Traversal: ";

printPostOrder(root);

cout << "\n";

return 0;

}



2.

#include <bits/stdc++.h>

using namespace std;

struct node

{

int key;

struct node \*left, \*right;

};

struct node \*newNode(int item)

{

struct node \*temp = new node;

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void preorder(struct node \*root)

{

if (root != NULL)

{

cout << root->key << " ";

preorder(root->left);

preorder(root->right);

}

}

vector<struct node \*> constructTrees(int start, int end)

{

vector<struct node \*> list;

if (start > end)

{

list.push\_back(NULL);

return list;

}

for (int i = start; i <= end; i++)

{

vector<struct node \*> leftSubtree = constructTrees(start, i - 1);

vector<struct node \*> rightSubtree = constructTrees(i + 1, end);

for (int j = 0; j < leftSubtree.size(); j++)

{

struct node\* left = leftSubtree[j];

for (int k = 0; k < rightSubtree.size(); k++)

{

struct node \* right = rightSubtree[k];

struct node \* node = newNode(i);

node->left = left;

node->right = right;

list.push\_back(node);

}

}

}

return list;

}

int main()

{

vector<struct node \*> totalTreesFrom1toN = constructTrees(1, 3);

cout << "Preorder traversals of all constructed BSTs are \n";

for (int i = 0; i < totalTreesFrom1toN.size(); i++)

{

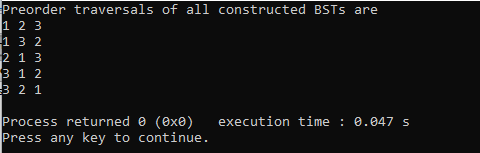
preorder(totalTreesFrom1toN[i]);

cout << endl;

}

return 0;

}



3.

#include <bits/stdc++.h>

using namespace std;

struct Node {

int key;

Node \*left, \*right;

Node(int k) {

key = k;

left = NULL;

right = NULL;

}

};

bool findPath(Node\* root, vector<int>& path, int k) {

if (!root)

return false;

path.push\_back(root->key);

if (root->key == k)

return true;

if ((root->left && findPath(root->left, path, k)) ||

(root->right && findPath(root->right, path, k)))

return true;

path.pop\_back();

return false;

}

int findLCA(Node\* root, int n1, int n2) {

vector<int> path1, path2;

if (!findPath(root, path1, n1) ||

!findPath(root, path2, n2))

return -1;

int i;

for (i = 0; i < path1.size() && i < path2.size(); i++) {

if (path1[i] != path2[i])

break;

}

return path1[i - 1];

}

int main() {

Node\* root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

root->right->left = new Node(6);

root->right->right = new Node(7);

cout << "LCA(4, 5) = " << findLCA(root, 4, 5) << endl;

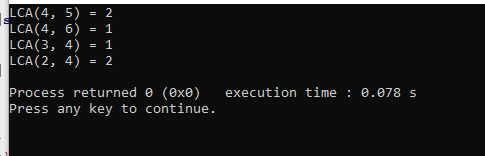
cout << "LCA(4, 6) = " << findLCA(root, 4, 6) << endl;

cout << "LCA(3, 4) = " << findLCA(root, 3, 4) << endl;

cout << "LCA(2, 4) = " << findLCA(root, 2, 4) << endl;

return 0;

}



4.

#include <bits/stdc++.h>

using namespace std;

class node

{

public:

int data;

node\* left;

node\* right;

};

int max(int inorder[], int strt, int end);

node\* newNode(int data);

node\* buildTree (int inorder[], int start, int end)

{

if (start > end)

return NULL;

int i = max (inorder, start, end);

node \*root = newNode(inorder[i]);

if (start == end)

return root;

root->left = buildTree (inorder, start, i - 1);

root->right = buildTree (inorder, i + 1, end);

return root;

}

int max (int arr[], int strt, int end)

{

int i, max = arr[strt], maxind = strt;

for(i = strt + 1; i <= end; i++)

{

if(arr[i] > max)

{

max = arr[i];

maxind = i;

}

}

return maxind;

}

node\* newNode (int data)

{

node\* Node = new node();

Node->data = data;

Node->left = NULL;

Node->right = NULL;

return Node;

}

void printInorder (node\* node)

{

if (node == NULL)

return;

printInorder (node->left);

cout<<node->data<<" ";

printInorder (node->right);

}

int main()

{

int inorder[] = {5, 10, 40, 30, 28};

int len = sizeof(inorder)/sizeof(inorder[0]);

node \*root = buildTree(inorder, 0, len - 1);

cout << "Inorder traversal of the constructed tree is \n";

printInorder(root);

return 0;

}

