**Name-Ayan Sadhukhan**

**Regd No-19BEC1448**

***Question 1:***

Write a program to compute the following operations on binary tree

1. Height of a binary tree
2. List of leaf nodes
3. Number of nodes
4. Degree of each node
5. Ancestors of a given node

***Code:***

1. ***Height of a binary tree:***

// C++ program to print inorder traversal

// using stack.

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree Node has data, pointer to left child

and a pointer to right child \*/

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

/\* Iterative function for inorder tree

traversal \*/

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

}

int height(Node\* root)

{

// Base case: empty tree has height 0

if (root == NULL)

return 0;

// recur for left and right subtree and consider maximum depth

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

}

/\* Driver program to test above functions\*/

int main()

{

struct 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);

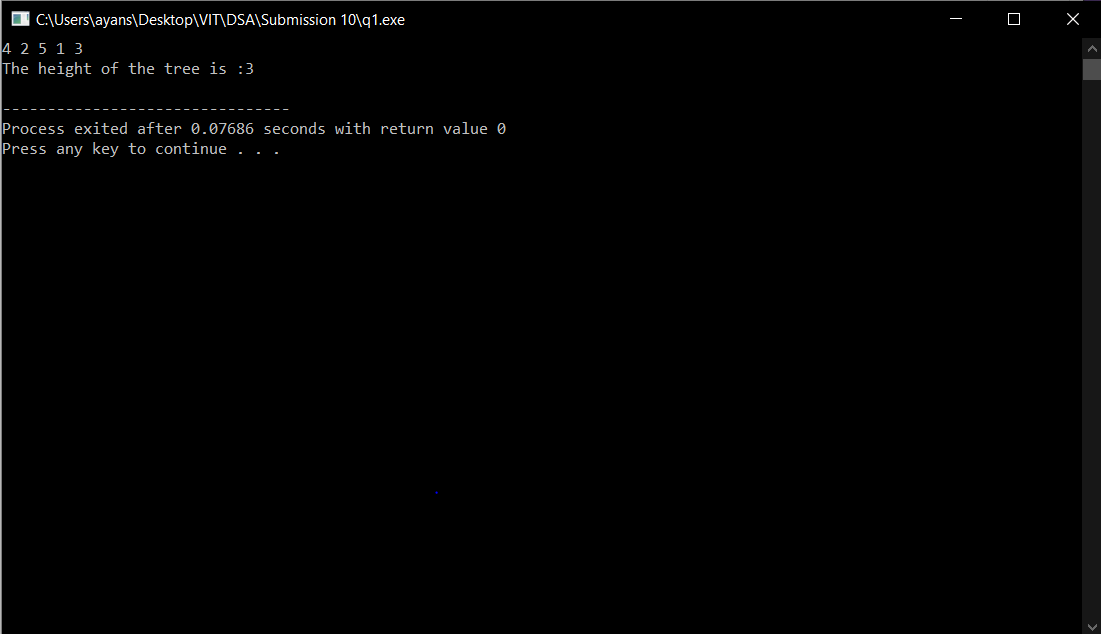
inOrder(root);

cout<<"\nThe height of the tree is :"<<height(root)<<endl;

return 0;

}

***Output:***



1. List of leaf nodes:

***Code:***

// C++ program to print inorder traversal

// using stack.

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree Node has data, pointer to left child

and a pointer to right child \*/

vector<int> a;

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

/\* Iterative function for inorder tree

traversal \*/

void degree(struct Node \*node){

int f=0;

if(node==NULL)

cout<<"degree is 0";

if(node->left!=NULL)

{

f+=1;

}

if(node->right!=NULL)

{

f+=1;

}

//cout<<"\nThe degree is :"<<f<<endl;

if(f==0)

{

a.push\_back(node->data);

}

f=0;

//degree(node->left);

//degree(node->right);

//node=node->

}

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

int f=0;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

f=f+1;

//degree(curr);

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

degree(curr);

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

cout<<"\nThe number of nodes in the tree is :";

cout<<f<<endl;

}

struct Node \*end(struct Node \*root)

{

}

/\* Driver program to test above functions\*/

int main()

{

struct 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);

inOrder(root);

cout<<"The list of leaf nodes are :";

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

{

cout<<a[i]<<" ";

}

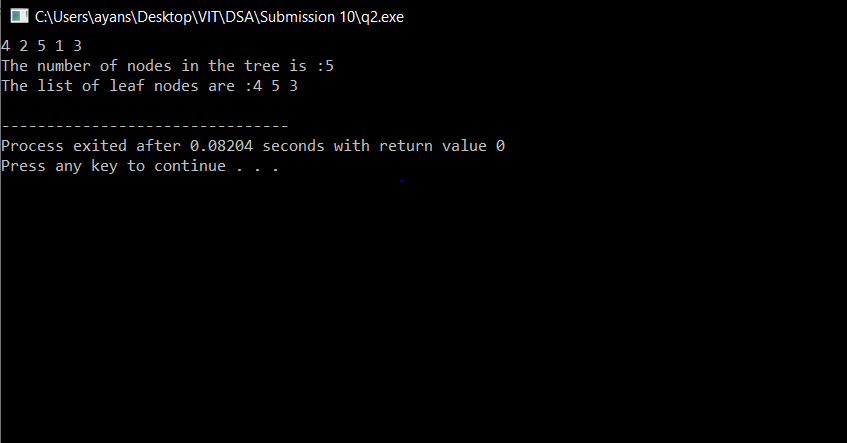
cout<<endl;

//degree(curr);

return 0;

}

***Output:***



1. Number of nodes

***Code:***

// C++ program to print inorder traversal

// using stack.

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree Node has data, pointer to left child

and a pointer to right child \*/

vector<int> a;

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

/\* Iterative function for inorder tree

traversal \*/

void degree(struct Node \*node){

int f=0;

if(node==NULL)

cout<<"degree is 0";

if(node->left!=NULL)

{

f+=1;

}

if(node->right!=NULL)

{

f+=1;

}

//cout<<"\nThe degree is :"<<f<<endl;

if(f==0)

{

a.push\_back(node->data);

}

f=0;

//degree(node->left);

//degree(node->right);

//node=node->

}

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

int f=0;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

f=f+1;

//degree(curr);

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

degree(curr);

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

cout<<"\nThe number of nodes in the tree is :";

cout<<f<<endl;

}

struct Node \*end(struct Node \*root)

{

}

/\* Driver program to test above functions\*/

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

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

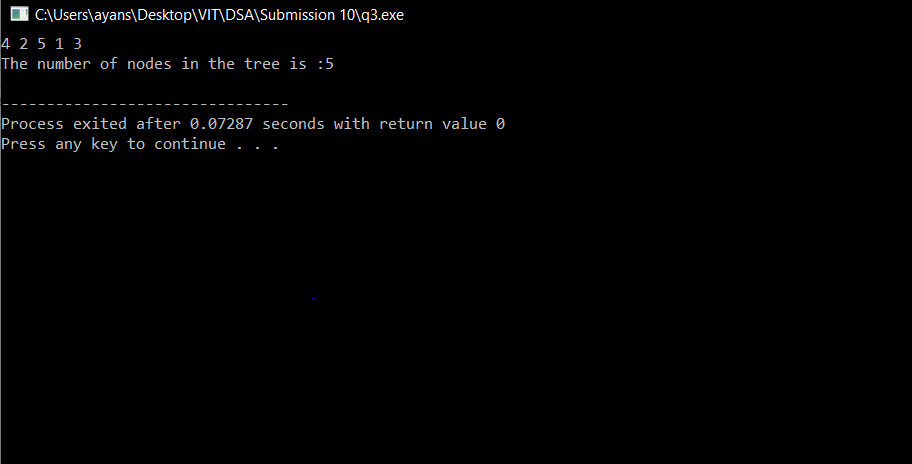
root->left->right = new Node(5);

inOrder(root);

return 0;

}

***Output:***



1. Degree of each node

***Code:***

// C++ program to print inorder traversal

// using stack.

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree Node has data, pointer to left child

and a pointer to right child \*/

vector<int> a;

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

/\* Iterative function for inorder tree

traversal \*/

void degree(struct Node \*node){

int f=0;

if(node==NULL)

cout<<"degree is 0";

if(node->left!=NULL)

{

f+=1;

}

if(node->right!=NULL)

{

f+=1;

}

cout<<"\nThe degree is :"<<f<<endl;

if(f==0)

{

a.push\_back(node->data);

}

f=0;

//degree(node->left);

//degree(node->right);

//node=node->

}

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

int f=0;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

f=f+1;

//degree(curr);

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

degree(curr);

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

//cout<<"\nThe number of nodes in the tree is :";

//cout<<f<<endl;

}

struct Node \*end(struct Node \*root)

{

}

/\* Driver program to test above functions\*/

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

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

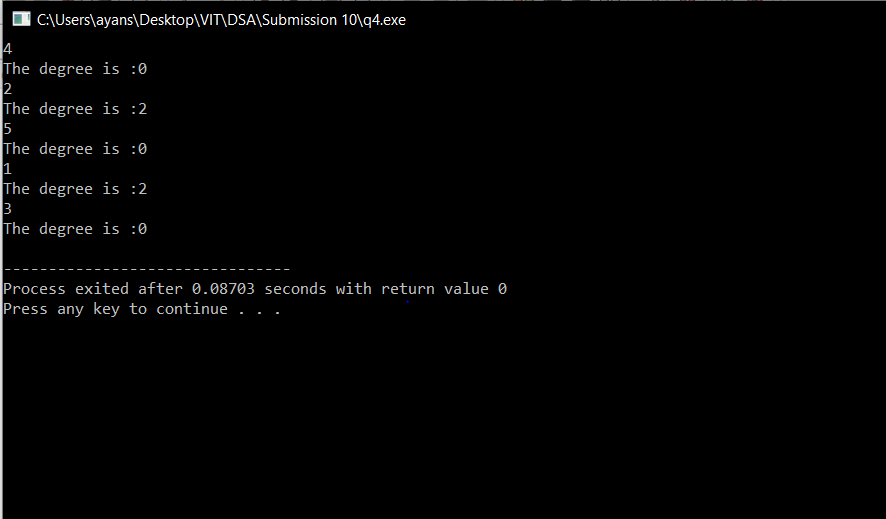
root->left->right = new Node(5);

inOrder(root);

return 0;

}

***Output:***



1. Ancestors of a given node

***Code:***

#include<bits/stdc++.h>

using namespace std;

/\* A binary tree node has data, pointer to left child

and a pointer to right child \*/

struct node

{

int data;

struct node\* left;

struct node\* right;

};

/\* If target is present in tree, then prints the ancestors

and returns true, otherwise returns false. \*/

bool printAncestors(struct node \*root, int target)

{

/\* base cases \*/

if (root == NULL)

return false;

if (root->data == target)

return true;

/\* If target is present in either left or right subtree of this node,

then print this node \*/

if ( printAncestors(root->left, target) ||

printAncestors(root->right, target) )

{

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

return true;

}

/\* Else return false \*/

return false;

}

/\* Helper function that allocates a new node with the

given data and NULL left and right pointers. \*/

struct node\* newnode(int data)

{

struct node\* node = (struct node\*)

malloc(sizeof(struct node));

node->data = data;

node->left = NULL;

node->right = NULL;

return(node);

}

void inOrder(struct node \*root)

{

stack<node \*> s;

node \*curr = root;

int f=0;

while (curr != NULL || s.empty() == false)

{

/\* Reach the left most Node of the

curr Node \*/

while (curr != NULL)

{

/\* place pointer to a tree node on

the stack before traversing

the node's left subtree \*/

s.push(curr);

curr = curr->left;

f=f+1;

//degree(curr);

}

/\* Current must be NULL at this point \*/

curr = s.top();

s.pop();

cout << curr->data << " ";

//degree(curr);

/\* we have visited the node and its

left subtree. Now, it's right

subtree's turn \*/

curr = curr->right;

} /\* end of while \*/

//cout<<"\nThe number of nodes in the tree is :";

//cout<<f<<endl;

}

/\* Driver program to test above functions\*/

int main()

{

struct node \*root = newnode(1);

root->left = newnode(2);

root->right = newnode(3);

root->left->left = newnode(4);

root->left->right = newnode(5);

root->left->left->left = newnode(7);

cout<<"The tree is :"<<endl;

inOrder(root);

cout<<"\n The ancestor nodes of node 7 is :";

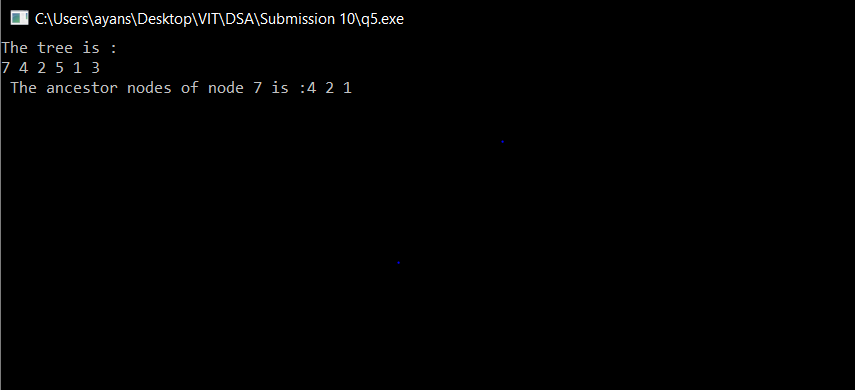
printAncestors(root, 7);

getchar();

return 0;

}

***Output:***



***Question 2:***

A full node is a node with two children. Write a program to compute all full nodes in the given binary tree.

***Code:***

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node\* left, \*right;

};

unsigned int getfullCount(struct Node\* root)

{

if (root == NULL)

return 0;

int res = 0;

if (root->left && root->right)

res++;

res += (getfullCount(root->left) +

getfullCount(root->right));

return res;

}

struct Node\* newNode(int data)

{

struct Node\* node = new Node;

node->data = data;

node->left = node->right = NULL;

return (node);

}

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

int f=0;

while (curr != NULL || s.empty() == false)

{

while (curr != NULL)

{

s.push(curr);

curr = curr->left;

f=f+1;

}

curr = s.top();

s.pop();

cout << curr->data << " ";

curr = curr->right;

}

cout<<"\nThe number of nodes in the tree is :";

cout<<f<<endl;

}

// Driver program

int main(void)

{

struct Node \*root = newNode(2);

root->left = newNode(7);

root->right = newNode(5);

root->left->right = newNode(6);

root->left->right->left = newNode(1);

root->left->right->right = newNode(11);

root->right->right = newNode(9);

root->right->right->left = newNode(4);

inOrder(root);

cout <<"The total number of full nodes in a binary tree is :"<< getfullCount(root);

return 0;

}

***Output:***

