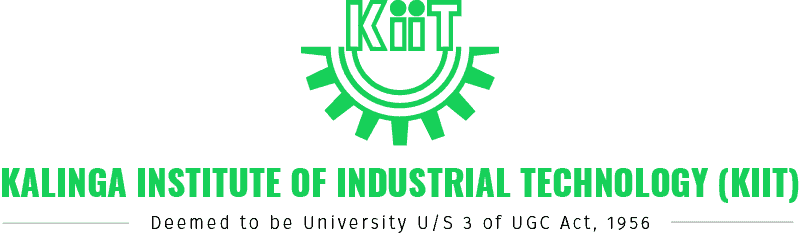
****

**NAME : SOHAM SAMANTA**

**ROLL NUMBER : 20051107**

**SECTION : CSE-14**

# Lab-06:- Evaluation

1. Create a Tree and Perform the In-order Traversal in the Tree.

**Solution:**

// SOHAM SAMANTA CODES

#include <bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss \

ios\_base::sync\_with\_stdio(false); \

cin.tie(NULL);

struct Node{

int data;

struct Node \*left, \*right;

Node(int data){

this->data = data;

left = right = NULL;

}

};

void inorder(struct Node \*node){

if (node == NULL)

return;

inorder(node->left);

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

inorder(node->right);

}

int32\_t main(){

ss;

// 1

// 2 3

// 4 5 6 7

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);

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

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

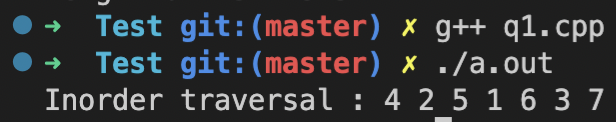
cout << "Inorder traversal : ";

inorder(root);

return 0;

}

**Output:**

  
2. Create a Graph and find the minimum spanning tree.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

typedef pair<int, int> iPair;

struct Graph{

int V, E;

vector< pair<int, iPair> > edges;

Graph(int V, int E){

this->V = V;

this->E = E;

}

void addEdge(int u, int v, int w){

edges.push\_back({w, {u, v}});

}

int kruskalMST();

};

struct DisjointSets{

int \*parent, \*rnk;

int n;

DisjointSets(int n){

this->n = n;

parent = new int[n+1];

rnk = new int[n+1];

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

rnk[i] = 0;

parent[i] = i;

}

}

int find(int u){

if (u != parent[u])

parent[u] = find(parent[u]);

return parent[u];

}

void merge(int x, int y){

x = find(x), y = find(y);

if (rnk[x] > rnk[y])

parent[y] = x;

else

parent[x] = y;

if (rnk[x] == rnk[y])

rnk[y]++;

}

};

int Graph::kruskalMST(){

int mst\_wt = 0;

sort(edges.begin(), edges.end());

DisjointSets ds(V);

vector< pair<int, iPair> >::iterator it;

for (it=edges.begin(); it!=edges.end(); it++)

{

int u = it->second.first;

int v = it->second.second;

int set\_u = ds.find(u);

int set\_v = ds.find(v);

if (set\_u != set\_v){

cout << u << " - " << v << endl;

mst\_wt += it->first;

ds.merge(set\_u, set\_v);

}

}

return mst\_wt;

}

int32\_t main(){

ss;

int V = 9, E = 14;

Graph g(V, E);

g.addEdge(0, 1, 4);

g.addEdge(0, 7, 8);

g.addEdge(1, 2, 8);

g.addEdge(1, 7, 11);

g.addEdge(2, 3, 7);

g.addEdge(2, 8, 2);

g.addEdge(2, 5, 4);

g.addEdge(3, 4, 9);

g.addEdge(3, 5, 14);

g.addEdge(4, 5, 10);

g.addEdge(5, 6, 2);

g.addEdge(6, 7, 1);

g.addEdge(6, 8, 6);

g.addEdge(7, 8, 7);

cout << "Edges of MST are \n";

int mst\_wt = g.kruskalMST();

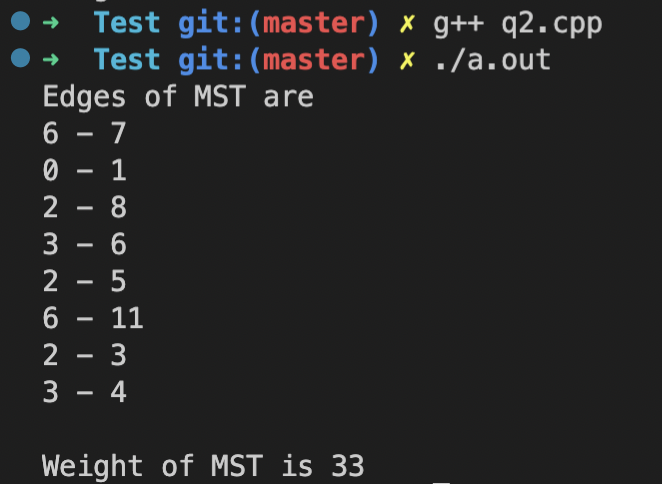
cout << "\nWeight of MST is " << mst\_wt;

cout<<endl;

return 0;

}

**Output:**

****

