Q. 1. Implement minimum cost spanning tree algorithm.

#include <iostream>

#include <vector>

#include <utility>

#include <algorithm>

using namespace std;

const int MAX = 1000;

int id[MAX], nodes, edges; //array id is use for check the parent of vertex;

pair <long long, pair<int, int> > p[MAX];

//initialise the parent array id[]

void init()

{

for(int i = 0;i < MAX;++i)

id[i] = i;

}

int root(int x)

{

while(id[x] != x) //if x is not itself parent then update its parent

{

id[x] = id[id[x]];

x = id[x];

}

return x; //return the parent

}

//function for union

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

//function to find out the edges in minimum spanning tree and its cost

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

for(int i = 0;i < edges;++i)

{

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

if(root(x) != root(y))

{

minimumCost += cost;

cout<<x<<" ----> "<<y<<" :"<<p[i].first<<endl;//print the edges contain in

spanning tree

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

init();

cout <<"Enter Nodes and edges"<<endl;

cin >> nodes >> edges;

//enter the vertex and cost of edges

for(int i = 0;i < edges;++i)

{

cout<<"Enter the value of X, Y and edges"<<endl;

cin >> x >> y >> weight;

p[i] = make\_pair(weight, make\_pair(x, y));

}

//sort the edges according to their cost

sort(p, p + edges);

minimumCost = kruskal(p);

cout <<"Minimum cost is "<< minimumCost << endl;

return 0;

}

Q. 2. Create binary tree and perform recursive traversals.

/\*Create binary tree and perform recursive traversals\*/

#include <iostream>

using namespace std;

// Structure for a node of a binary tree

struct Node {

int data;

Node\* left;

Node\* right;

};

// Function to create a new node and return its address

Node\* getNewNode(int data) {

Node\* newNode = new Node();

newNode->data = data;

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

return newNode;

}

// Recursive function to do pre-order traversal of the binary tree

void preOrder(Node\* root) {

if (root == NULL) return;

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

preOrder(root->left);

preOrder(root->right);

}

// Recursive function to do in-order traversal of the binary tree

void inOrder(Node\* root) {

if (root == NULL) return;

inOrder(root->left);

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

inOrder(root->right);

}

// Recursive function to do post-order traversal of the binary tree

void postOrder(Node\* root) {

if (root == NULL) return;

postOrder(root->left);

postOrder(root->right);

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

}

int main() {

Node\* root = getNewNode(1);

root->left = getNewNode(2);

root->right = getNewNode(3);

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

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

cout << "Pre-order traversal: ";

preOrder(root);

cout << endl;

cout << "In-order traversal: ";

inOrder(root);

cout << endl;

cout << "Post-order traversal: ";

postOrder(root);

cout << endl;

return 0;

}