*main() function:*

//============================================================================

// Name : 21118\_DSA\_A07.cpp

// Author : Shubham (Roll No: 21118)

//============================================================================

**#include** <iostream>

**#include** "mst.h"

**using** **namespace** std;

**int** **main**() {

**while** (1) {

**int** n, m, u, v, wt, opt;

cout << "\n\nEnter number of vertices and edges.\n";

cin >> n >> m;

mst mymst(n, m);

cout << "Enter edges and weights:\n";

**for** (**int** i = 0; i < m; i++) {

cin >> u >> v >> wt;

mymst.addEdge(u, v, wt);

}

cout << "How do you want to calculate mst:\n";

cout << "\t1 for Prim's Algorithm\n";

cout << "\t2 for Kruskal's Algorithm\n";

cin >> opt;

**if** (opt == 1)

cout << mymst.prims\_mst\_wt() << **endl**;

**else** **if** (opt == 2)

cout << mymst.kruskals\_mst\_wt() << **endl**;

**else**

cout << "INVALID CHOICE.\n";

cout << "Do you want to calculate again?(0 or 1)\n";

cin >> opt;

**if** (opt == 0)

**break**;

}

**return** 0;

}

*mst.h header file(contain declaration of mst class):*

/\*

\* kruskals\_mst.h

\*

\* Created on: 25-May-2021

\* Author: Shubham

\*/

**#ifndef** MST\_H\_

**#define** MST\_H\_

**class** Edge {

**private**:

**int** u, v, wt;

**public**:

**Edge**() {u=0, v=0, wt=0;}

**Edge**(**int** x, **int** y, **int** w);

**friend** **class** mst;

};

**class** mst {

**private**:

**int** vertices, edges;

**int**\*\* adj\_list;

**int**\* par; // to keep track of root node of mst in disjoint mst's (kruskal's algo)

**int** **find\_mst**(**int** u);

**void** **union\_mst**(**int** u1, **int** u2);

**public**:

**mst**(**int** n, **int** m);

**void** **printGraph**();

**void** **addEdge**(**int** u, **int** v, **int** w);

**int** **prims\_mst\_wt**();

**int** **kruskals\_mst\_wt**();

**~mst**();

};

**#endif** /\* MST\_H\_ \*/

*mst.cpp file(contains implementation of mst class):*

/\*

\* mst.cpp

\*

\* Created on: 25-May-2021

\* Author: Shubham

\*/

**#include** <iostream>

**#include** <algorithm> // for sort function

**#include** "mst.h"

**using** **namespace** std;

**Edge::Edge**(**int** x, **int** y, **int** w) {

u = x, v = y, wt = w;

}

// find mst of vertex u

**int** **mst::find\_mst**(**int** u) {

**while** (par[u] != u)

u = par[u];

**return** u;

}

// take union of mst's represented by u1 and u2

**void** **mst::union\_mst**(**int** u1, **int** u2) {

**int** x = find\_mst(u1), y = find\_mst(u2);

**if** (x != y)

par[x] = y;

}

**mst::mst**(**int** n, **int** m) {

vertices = n, edges = m;

par = **new** **int**[n];

adj\_list = **new** **int**\*[n];

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

adj\_list[i] = **new** **int**[n];

par[i] = i;

}

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

**for** (**int** j = 0; j < n; j++)

adj\_list[i][j] = 0;

}

**void** **mst::addEdge**(**int** u, **int** v, **int** w) {

adj\_list[u][v] = w;

adj\_list[v][u] = w;

}

**int** **mst::prims\_mst\_wt**() {

**bool** mst\_vert[vertices] = { 0 };

**int** vert\_wt[vertices], mst\_wt = 0, parent\_vert[vertices];

**for** (**int** i = 0; i < vertices; i++)

vert\_wt[i] = 10000, parent\_vert[i] = -1;

vert\_wt[0] = 0;

**while** (1) {

**int** v = -1;

**for** (**int** i = 0; i < vertices; i++)

**if** (!mst\_vert[i] && (v == -1 || vert\_wt[i] < vert\_wt[v]))

v = i;

**if** (v == -1)

**break**;

mst\_vert[v] = 1;

**if** (vert\_wt[v] == 10000)

vert\_wt[v] = 0;

mst\_wt += vert\_wt[v];

**for** (**int** j = 0; j < vertices; j++)

**if** (adj\_list[v][j] && !mst\_vert[j]

&& (adj\_list[v][j] < vert\_wt[j])) {

vert\_wt[j] = adj\_list[v][j];

parent\_vert[j] = v;

}

}

cout << "mst edges:\n";

**for** (**int** i = 0; i < vertices; i++)

**if** (parent\_vert[i] != -1)

cout << i << " <-> " << parent\_vert[i] << '\n';

**return** mst\_wt;

}

**int** **mst::kruskals\_mst\_wt**() {

**int** mst\_wt = 0;

Edge edge\_list[edges];

**int** cnt = 0;

**for** (**int** i = 0; i < vertices; i++)

**for** (**int** j = i + 1; j < vertices; j++)

**if** (adj\_list[i][j])

edge\_list[cnt++] = Edge(i, j, adj\_list[i][j]);

// sort function from c++ standard template library with custom comparator (using lambda fn)

sort(edge\_list, edge\_list + edges, [](Edge e1, Edge e2) {

**return** e1.wt < e2.wt;

});

**int** edge\_cnt = 0, i = 0, mst\_edges[vertices - 1][2];

**while** (edge\_cnt < vertices - 1 && i < edges) {

Edge cur = edge\_list[i++];

**int** u\_par = find\_mst(cur.u), v\_par = find\_mst(cur.v);

**if** (u\_par != v\_par) {

mst\_edges[edge\_cnt][0] = cur.u, mst\_edges[edge\_cnt][1] = cur.v;

mst\_wt += cur.wt, edge\_cnt++;

union\_mst(u\_par, v\_par);

}

}

cout << "mst edges:\n";

**for** (**int** i = 0; i < edge\_cnt; i++)

cout << mst\_edges[i][0] << " <-> " << mst\_edges[i][1] << **endl**;

**return** mst\_wt;

}

**void** **mst::printGraph**() {

cout << "Adjacency matrix representation of graph: \n";

**for** (**int** i = 0; i < vertices; i++, cout << '\n')

**for** (**int** j = 0; j < vertices; j++)

cout << adj\_list[i][j] << " ";

}

**mst::~mst**() {

**for** (**int** i = 0; i < vertices; i++)

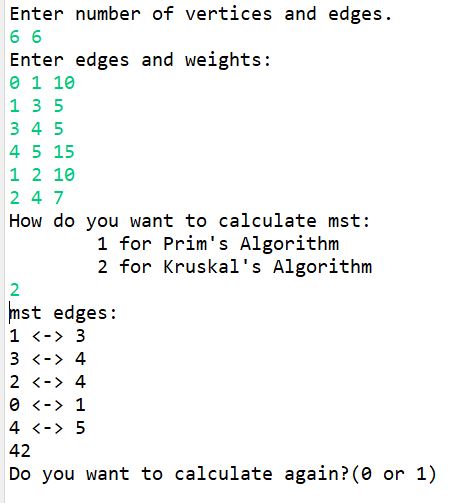
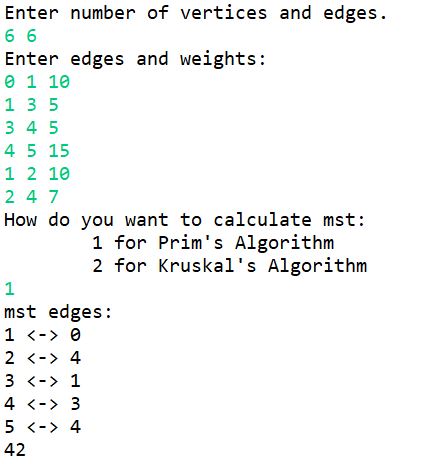
**delete**[] adj\_list[i];

**delete**[] adj\_list;

**delete**[] par;

}

**TESTCASE 1:**



**TESTCASE 2:**

