**PUNE INSTITUTE OF COMPUTER TECHNOLOGY DHANKAWADI, PUNE**

**Data Structures And Algorithms(DSA)**

**Assignment No. 07**

**Title : Graph (Minimum Spanning Tree)**

**SE-IT-10**  **ACADEMIC YEAR :- 2020-2021**

**Name :- Diptesh Ravindra Varule Roll No :- 23277**

**Source Code :**

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

// Name : dsa\_Assignment7.cpp

// Author : Diptesh Varule

// Version : Updating…..

// Copyright : The Cartel

// Description : Hello World in C++, Ansi-style

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

**Source Code:**

**MST\_header.h :**

#ifndef MST\_HEADER\_H\_

#define MST\_HEADER\_H\_

#include <cstring>

using namespace std;

//Structure AL.

struct AL{

int vertex; //Vertex Number.

int weight; //Weight.

AL\* link; //Link To Another Vertex.

};

//Structure of Edge.

struct Edge

{

int U; //Starting Vertex.

int V; //Ending Vertex.

int wt; //Weight Of The Edge.

};

//Class Graph.

class Graph{

private:

int G[10][10] = {0}; //Adjacent Matrix.

int vertices; //Number of Vertices.

string vname[10]; //Vertices Names.

int edges; //Number of Edges.

AL \*list[10]; //Adajent List.

bool isDirected; //To check for Directed Graph.

Edge edge[50]; //Edge Array.

public:

Graph(); //Constructor.

~Graph(); //Destructor.

void createGraphMat(); //create Graph(Matrix).

void displayGraphMat(); //display Grapg(Matrix).

void primAlgo(); //Prim's Algorithm.

void kruskalAlgo(); //Kruskal's Algorithm.

void sortg(); //Sort The Edge Array.

void createGraphList(); //create Graph List.

void displayGraphList(); //display Graph List.

};

AL\* newNode(int i, int wt);

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

**MST\_Impl.cpp :**

#include <iostream>

#include "MST\_Header.h"

#include <bits/stdc++.h>

using namespace std;

//Constructor

Graph::Graph(){

vertices = 0;

edges = 0;

G[10][10] = {0};

isDirected = false;

}

//Destructor

Graph::~Graph(){}

//Create Graph Using Adjacent Matrix.

void Graph::createGraphMat(){

vertices = 0;

edges = 0;

cout<<"Is The Building Graph Directed"<<endl;

cout<<"1.True\n2.False"<<endl;

int in = -1;

while(in < 1 || in > 2){

cout<<"Directed : ";

cin>>in;

}

if(in == 1){

isDirected = true;

}

else{

isDirected = false;

}

cout<<"\nEnter The Number Of Building In the Graph : "; //Asking For Vertices.

cin>>vertices;

cout<<endl;

cin.ignore();

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

cout<<"Enter The Name Of The Building "<<i+1<<" : ";

getline(cin, vname[i]);

}

cout<<endl;

if(isDirected){

int k = 0;

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

for(int j = 0; j < vertices; j++){ //Traversing Matrix.

if(i == j){

G[i][j] = 0;

continue;

}

cout<<"Enter The Distance Between "<<vname[i]<<" --> "<<vname[j]<<" : "; //Asking For Weight.

int weight;

cin>>weight;

G[i][j] = weight; //Assigning Weight.

if(weight != 0){ //Entering The Edges in Edge Array.

edge[k].U = i;

edge[k].V = j;

edge[k].wt = weight;

k++;

edges++;

}

}

cout<<endl;

}

cout<<"\nAdjacent Matrix Created Successfully";

return;

}

int k = 0;

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

for(int j = i+1; j < vertices; j++){ //Traversing For Upper Half Matrix.

cout<<"Enter The Distance Between "<<vname[i]<<" And "<<vname[j]<<" : "; //Asking For Weight.

int weight;

cin>>weight;

G[i][j] = G[j][i] = weight; //Assigning Weight.

if(weight != 0){ //Entering The Edges in Edge Array.

edge[k].U = i;

edge[k].V = j;

edge[k].wt = weight;

k++;

edges++;

edge[k].U = j;

edge[k].V = i;

edge[k].wt = weight;

k++;

edges++;

}

}

cout<<endl;

}

cout<<"\nAdjacent Matrix Created Successfully";

}

//Display Graph Using Adjacent Matrix.

void Graph::displayGraphMat(){

if(isDirected){

cout<<endl;

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

for(int j = 0; j < vertices; j++){//Traversing Matrix.

//Displaying The Edges With it's Weight.

if(G[i][j] != 0){

cout<<"The Distance Between "<<vname[i]<<" And "<<vname[j]<<" Is : "<<G[i][j]<<endl;

}

}

cout<<endl;

}

}

else{

cout<<endl;

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

for(int j = i+1; j < vertices; j++){//Traversing For Upper Half Matrix.

//Displaying The Edges With it's Weight.

if(G[i][j] != 0){

cout<<"The Distance Between "<<vname[i]<<" And "<<vname[j]<<" Is : "<<G[i][j]<<endl;

}

}

cout<<endl;

}

}

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

cout<<"|"<<vname[edge[i].U]<<"|-->|"<<vname[edge[i].V];

cout<<"|"<<", Weight = "<<edge[i].wt<<endl;

}

cout<<endl;

cout<<"\nDisplaying Adjacency List"<<endl;

createGraphList();

displayGraphList();

}

//Prim's Algorithm.

void Graph::primAlgo()

{

int current, totalvisited, mincost, i;

int v = vertices;

int d[30], visited[30], p[30], weight[10][10];

for(int i = 0; i < 30; i++){ //Initializing distance, path and visited array.

d[i] = 7777;

p[i] = visited[i] = 0;

}

for(int m = 0; m < vertices; m++){ //Creating weight Matrix;

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

weight[m][n] = G[m][n];

}

}

current=0; d[current]=0;

totalvisited =1;

visited[current]=1;

cout<<"\nVisited, Distance and path Status : "<<endl; //Displaying Distance and Path Status.

for(i = 0; i < v; i++){

cout<<endl<<"i = "<<i<<", Visited["<<i<<"] = "<<visited[i]<<", p["<<i<<"] = "<<p[i]<<", d["<<i<<"] = "<<d[i];

}

cout<<endl<<"Entering While Loop"<<endl;

while(totalvisited != v){

//Displaying Total Visited.

cout<<endl<<"Total visited = "<<totalvisited;

for (i=0;i<v;i++) //Assigning distance and path array to it's minimum avaialible.

{

if(weight[current][i]!=0)

if(visited[i]==0)

if(d[i]>weight[current][i])

{

d[i] = weight[current][i];

p[i] = current;

}

}

//Displaying Distance and Path Status.

cout<<"\nVisited, Distance and path Status : "<<endl;

for(i = 0; i < v; i++){

cout<<endl<<"i = "<<i<<", Visited["<<i<<"] = "<<visited[i]<<", p["<<i<<"] = "<<p[i]<<", d["<<i<<"] = "<<d[i];

}

mincost= 7777;

cout<<endl<<"\nNow, Finding The Minimum Cost"<<endl;

for (i=0;i<v;i++) //Finfing The Minimum cost among distance Array.

{

if(visited[i] ==0)

if (d[i] <mincost)

{

mincost = d[i];

current = i;

}

}

//Displaying Mincost and current.

cout<<"\n\nmincost = "<<mincost;

cout<<"\ncurrent = "<<current<<endl;

visited[current] = 1;

totalvisited++;

cout<<endl<<"Current = "<<current<<endl<<"Total Visited = "<<totalvisited;

} /\*end of while loop \*/

mincost =0;

for (i=0;i<v;i++) //Calculating The minimum Cost.

mincost += d[i];

cout<<"\nMinimum cost = ";

cout<<mincost<<endl;

//Displaying Distance and Path Status.

cout<<"\nVisited, Distance and path Status : "<<endl;

for(i = 0; i < v; i++){

cout<<endl<<"i = "<<i<<", Visited["<<i<<"] = "<<visited[i]<<", p["<<i<<"] = "<<p[i]<<", d["<<i<<"] = "<<d[i];

}

//Representation of Matrix.

for(int m = 0; m < vertices; m++){ //Assigning All Weight To zero in Weight Matrix.

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

weight[m][n] = 0;

}

}

for(int m = 0; m < vertices; m++){ //Assigning Weights According To Distance Array and Path Array.

weight[m][p[m]] = d[m];

weight[p[m]][m] = d[m];

}

cout<<endl<<endl;

for(int i = 1; i < v; i++){

cout<<"|"<<vname[p[i]]<<"|-->|"<<vname[i];

cout<<"|"<<", Weight = "<<d[i]<<endl;

}

cout<<endl<<"Total Cost of MST is "<<mincost<<endl; //Displaying Cost.

}

//Method To Sort The Edges.

void Graph::sortg()

{

int i,j;

Edge temp;

//using Bubble Sort.

for(i=0;i<edges-1;i++)

{ for(j=0;j<edges-1;j++)

{ if(edge[j].wt>edge[j+1].wt) //Swapping If Greater.

{

temp=edge[j+1];

edge[j+1]=edge[j];

edge[j]=temp;

}

}

}

cout<<"\nSorted List of edges"<<endl;

for(int i=0;i<edges;i++) //Displaying The Sorted List.

{

cout<<"|"<<edge[i].U<<"->"<<edge[i].V<<"|, Weight : "<<edge[i].wt<<endl;

}

}

//Kruskal's Algorithm.

void Graph::kruskalAlgo()

{

int e = edges, i, j, conn[50], k, r, p, Val, cnt = 0, m = 0; //Initializing Variables.

cout<<"\nEdge array Entered By The User is :"<<endl; //Displaying Edge Array.

for (i=0;i<e;i++)

{

cout<<"|"<<edge[i].U<<"->"<<edge[i].V<<"|, Weight : "<<edge[i].wt<<endl;

}

sortg(); // Sort the list of edges

cout<<endl<<endl;

Edge t[50], h[50]; //creating Edge Array To Store Edges.

for (i=0;i<e;i++)

conn[i]=0;

Val=1;

cnt=0;

j=0;

while(cnt<e-1 && j<e)

{ //both vertices are not visited

if(conn[edge[j].U]==0 && conn[edge[j].V]==0)

{

//Displaying Selected Edge.

cout<<"Edge selected"<<"|"<<edge[j].U<<"->"<<edge[j].V<<"|, Weight : "<<edge[j].wt<<", Both The Vertices Are Not Visited"<<endl;

//Adding Edge to t.

t[cnt]=edge[j];

//Assigning The Visited Vertices connected value.

conn[edge[j].U]=Val;

conn[edge[j].V]=Val;

Val++;

cnt++;

//Adding Edge to h.

h[m].U = edge[j].U;

h[m].V = edge[j].V;

h[m].wt = edge[j].wt;

m++;

}

else if(conn[edge[j].U]!=conn[edge[j].V])//if conn value of both vertices is not equal

{

//Displaying Selected Edge.

cout<<"Edge selected"<<"|"<<edge[j].U<<"->"<<edge[j].V<<"|, Weight : "<<edge[j].wt<<", Both The Vertices Have Different Connected Value"<<endl;

//Adding Edge to h.

h[m].U = edge[j].U;

h[m].V = edge[j].V;

h[m].wt = edge[j].wt;

m++;

if(conn[edge[j].U]!=0 && conn[edge[j].V]!=0)//if both vertices are visited

{

t[cnt]=edge[j]; //Adding Edge to t.

if(conn[edge[j].U]<conn[edge[j].V])//if connected value of first vertex is less

{

r=conn[edge[j].U]; //Greater Connected Value.

p=conn[edge[j].V]; //Smaller Connected Value.

}

else //if connected value of second vertex is less

{ r=conn[edge[j].V]; //Greater Connected Value.

p=conn[edge[j].U]; //Smaller Connected Value.

}

for(k=0;k<e;k++)//replacing Greater connected value p with small value r

{

if(conn[k]==p)

conn[k]=r;

}

cnt++;

}

else if(conn[edge[j].U]==0 && conn[edge[j].V]!=0)//If first vertex is not visited & second is visited

{

t[cnt]=edge[j]; //Adding Edge to t.

conn[edge[j].U]=conn[edge[j].V]; //Assigning same connected Value.

cnt++;

}

else //if first vertex is visited & second is not visited

{

t[cnt]=edge[j]; //Adding Edge to t.

conn[edge[j].V]=conn[edge[j].U]; //Assigning same connected Value.

cnt++;

}

}

else //Both verticces has same connected value-reject edge--- forms cycle

//Displaying Rejected Edge.

cout<<"Edge Rejected "<<"|"<<edge[j].U<<"->"<<edge[j].V<<"|, Weight : "<<edge[j].wt<<", Both The Vertices Have Same Connected Value"<<endl;

j++;

}

cout<<"\n\nKruskal's MST contains following edges::"<<endl;

int cost=0;

for(i=0;i<cnt;i++)

{

//Displaying Selected Edges.

cout<<"|"<<h[i].U<<"->"<<h[i].V<<"|, Weight : "<<h[i].wt<<endl;

cost = cost + h[i].wt;

}

cout<<endl<<"Total Cost of MST is "<<cost<<endl; //Displaying Cost.

}

//Create Graph Using Adjacent List.

void Graph::createGraphList(){

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

AL\* nev = newNode(i, 0);

list[i] = nev;

}

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

AL\* trav = list[edge[i].U];

while(trav->link != NULL){

trav = trav->link;

}

AL\* nev = newNode(edge[i].V, edge[i].wt);

trav->link = nev;

}

}

//Display Graph Using Adjacent List.

void Graph::displayGraphList(){

cout<<endl;

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

cout<<"|"<<vname[list[i]->vertex]<<"|";

AL\* trav = list[i]; //Creating trav for Traversal.

while(trav != NULL){

trav = trav->link;

if(trav == NULL){ //If Trav is Null Then Break.

break;

}

cout<<"->|"<<vname[trav->vertex]<<", "<<trav->weight<<"|"; //Display vertex.

}

cout<<endl; //Space.

}

}

//Function to create new AL Node.

AL\* newNode(int i, int wt){

AL \*temp = new AL;

temp->vertex = i;

temp->weight = wt;

temp->link = NULL;

return temp;

}

**MST.cpp :**

#include <iostream>

#include "MST\_Header.h"

using namespace std;

int main() {

Graph g; //Creating Graph.

int ch;

do{

cout<<"\n=============================================\n";

cout<<"\n1.Input Graph \n2.Display Graph \n3.Prim's Algorithm";

cout<<"\n4.Kruskal's Algorithm \n0.Exit \n";

cout<<"\nEnter Choice : ";

cin>>ch;

cout<<"\n=============================================\n";

switch(ch){

case 1: //Creating Graph.

g.createGraphMat();

break;

case 2: //Displaying Graph.

g.displayGraphMat();

break;

case 3: //Displaying MST Using Prim's Algorithm.

g.primAlgo();

break;

case 4: //Displaying MST Using Kruskal's Algorithm.

g.kruskalAlgo();

break;

case 0: //Ending The Program.

break;

default://Handling Invalid Input.

cout<<"\n\tINVALID OPTION !!!";

break;

}

}while(ch != 0);

cout << "\nProgram End" << endl; // prints MST

return 0;

}

**Output :**

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 1

=============================================

Is The Building Graph Directed

1.True

2.False

Directed : 2

Enter The Number Of Building In the Graph : 5

Enter The Name Of The Building 1 : F-Wing

Enter The Name Of The Building 2 : H-Wing

Enter The Name Of The Building 3 : Hostels

Enter The Name Of The Building 4 : Canteen

Enter The Name Of The Building 5 : Ground

Enter The Distance Between F-Wing And H-Wing : 4

Enter The Distance Between F-Wing And Hostels : 10

Enter The Distance Between F-Wing And Canteen : 4

Enter The Distance Between F-Wing And Ground : 8

Enter The Distance Between H-Wing And Hostels : 7

Enter The Distance Between H-Wing And Canteen : 6

Enter The Distance Between H-Wing And Ground : 3

Enter The Distance Between Hostels And Canteen : 5

Enter The Distance Between Hostels And Ground : 10

Enter The Distance Between Canteen And Ground : 5

Adjacent Matrix Created Successfully

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 2

=============================================

The Distance Between F-Wing And H-Wing Is : 4

The Distance Between F-Wing And Hostels Is : 10

The Distance Between F-Wing And Canteen Is : 4

The Distance Between F-Wing And Ground Is : 8

The Distance Between H-Wing And Hostels Is : 7

The Distance Between H-Wing And Canteen Is : 6

The Distance Between H-Wing And Ground Is : 3

The Distance Between Hostels And Canteen Is : 5

The Distance Between Hostels And Ground Is : 10

The Distance Between Canteen And Ground Is : 5

|F-Wing|-->|H-Wing|, Weight = 4

|H-Wing|-->|F-Wing|, Weight = 4

|F-Wing|-->|Hostels|, Weight = 10

|Hostels|-->|F-Wing|, Weight = 10

|F-Wing|-->|Canteen|, Weight = 4

|Canteen|-->|F-Wing|, Weight = 4

|F-Wing|-->|Ground|, Weight = 8

|Ground|-->|F-Wing|, Weight = 8

|H-Wing|-->|Hostels|, Weight = 7

|Hostels|-->|H-Wing|, Weight = 7

|H-Wing|-->|Canteen|, Weight = 6

|Canteen|-->|H-Wing|, Weight = 6

|H-Wing|-->|Ground|, Weight = 3

|Ground|-->|H-Wing|, Weight = 3

|Hostels|-->|Canteen|, Weight = 5

|Canteen|-->|Hostels|, Weight = 5

|Hostels|-->|Ground|, Weight = 10

|Ground|-->|Hostels|, Weight = 10

|Canteen|-->|Ground|, Weight = 5

|Ground|-->|Canteen|, Weight = 5

Displaying Adjacency List

|F-Wing|->|H-Wing, 4|->|Hostels, 10|->|Canteen, 4|->|Ground, 8|

|H-Wing|->|F-Wing, 4|->|Hostels, 7|->|Canteen, 6|->|Ground, 3|

|Hostels|->|F-Wing, 10|->|H-Wing, 7|->|Canteen, 5|->|Ground, 10|

|Canteen|->|F-Wing, 4|->|H-Wing, 6|->|Hostels, 5|->|Ground, 5|

|Ground|->|F-Wing, 8|->|H-Wing, 3|->|Hostels, 10|->|Canteen, 5|

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 3

=============================================

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 0, p[1] = 0, d[1] = 7777

i = 2, Visited[2] = 0, p[2] = 0, d[2] = 7777

i = 3, Visited[3] = 0, p[3] = 0, d[3] = 7777

i = 4, Visited[4] = 0, p[4] = 0, d[4] = 7777

Entering While Loop

Total visited = 1

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 0, p[1] = 0, d[1] = 4

i = 2, Visited[2] = 0, p[2] = 0, d[2] = 10

i = 3, Visited[3] = 0, p[3] = 0, d[3] = 4

i = 4, Visited[4] = 0, p[4] = 0, d[4] = 8

Now, Finding The Minimum Cost

mincost = 4

current = 1

Current = 1

Total Visited = 2

Total visited = 2

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 1, p[1] = 0, d[1] = 4

i = 2, Visited[2] = 0, p[2] = 1, d[2] = 7

i = 3, Visited[3] = 0, p[3] = 0, d[3] = 4

i = 4, Visited[4] = 0, p[4] = 1, d[4] = 3

Now, Finding The Minimum Cost

mincost = 3

current = 4

Current = 4

Total Visited = 3

Total visited = 3

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 1, p[1] = 0, d[1] = 4

i = 2, Visited[2] = 0, p[2] = 1, d[2] = 7

i = 3, Visited[3] = 0, p[3] = 0, d[3] = 4

i = 4, Visited[4] = 1, p[4] = 1, d[4] = 3

Now, Finding The Minimum Cost

mincost = 4

current = 3

Current = 3

Total Visited = 4

Total visited = 4

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 1, p[1] = 0, d[1] = 4

i = 2, Visited[2] = 0, p[2] = 3, d[2] = 5

i = 3, Visited[3] = 1, p[3] = 0, d[3] = 4

i = 4, Visited[4] = 1, p[4] = 1, d[4] = 3

Now, Finding The Minimum Cost

mincost = 5

current = 2

Current = 2

Total Visited = 5

Minimum cost = 16

Visited, Distance and path Status :

i = 0, Visited[0] = 1, p[0] = 0, d[0] = 0

i = 1, Visited[1] = 1, p[1] = 0, d[1] = 4

i = 2, Visited[2] = 1, p[2] = 3, d[2] = 5

i = 3, Visited[3] = 1, p[3] = 0, d[3] = 4

i = 4, Visited[4] = 1, p[4] = 1, d[4] = 3

|F-Wing|-->|H-Wing|, Weight = 4

|Canteen|-->|Hostels|, Weight = 5

|F-Wing|-->|Canteen|, Weight = 4

|H-Wing|-->|Ground|, Weight = 3

Total Cost of MST is 16

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 4

=============================================

Edge array Entered By The User is :

|0->1|, Weight : 4

|1->0|, Weight : 4

|0->2|, Weight : 10

|2->0|, Weight : 10

|0->3|, Weight : 4

|3->0|, Weight : 4

|0->4|, Weight : 8

|4->0|, Weight : 8

|1->2|, Weight : 7

|2->1|, Weight : 7

|1->3|, Weight : 6

|3->1|, Weight : 6

|1->4|, Weight : 3

|4->1|, Weight : 3

|2->3|, Weight : 5

|3->2|, Weight : 5

|2->4|, Weight : 10

|4->2|, Weight : 10

|3->4|, Weight : 5

|4->3|, Weight : 5

Sorted List of edges

|1->4|, Weight : 3

|4->1|, Weight : 3

|0->1|, Weight : 4

|1->0|, Weight : 4

|0->3|, Weight : 4

|3->0|, Weight : 4

|2->3|, Weight : 5

|3->2|, Weight : 5

|3->4|, Weight : 5

|4->3|, Weight : 5

|1->3|, Weight : 6

|3->1|, Weight : 6

|1->2|, Weight : 7

|2->1|, Weight : 7

|0->4|, Weight : 8

|4->0|, Weight : 8

|0->2|, Weight : 10

|2->0|, Weight : 10

|2->4|, Weight : 10

|4->2|, Weight : 10

Edge selected|1->4|, Weight : 3, Both The Vertices Are Not Visited

Edge Rejected |4->1|, Weight : 3, Both The Vertices Have Same Connected Value

Edge selected|0->1|, Weight : 4, Both The Vertices Have Different Connected Value

Edge Rejected |1->0|, Weight : 4, Both The Vertices Have Same Connected Value

Edge selected|0->3|, Weight : 4, Both The Vertices Have Different Connected Value

Edge Rejected |3->0|, Weight : 4, Both The Vertices Have Same Connected Value

Edge selected|2->3|, Weight : 5, Both The Vertices Have Different Connected Value

Edge Rejected |3->2|, Weight : 5, Both The Vertices Have Same Connected Value

Edge Rejected |3->4|, Weight : 5, Both The Vertices Have Same Connected Value

Edge Rejected |4->3|, Weight : 5, Both The Vertices Have Same Connected Value

Edge Rejected |1->3|, Weight : 6, Both The Vertices Have Same Connected Value

Edge Rejected |3->1|, Weight : 6, Both The Vertices Have Same Connected Value

Edge Rejected |1->2|, Weight : 7, Both The Vertices Have Same Connected Value

Edge Rejected |2->1|, Weight : 7, Both The Vertices Have Same Connected Value

Edge Rejected |0->4|, Weight : 8, Both The Vertices Have Same Connected Value

Edge Rejected |4->0|, Weight : 8, Both The Vertices Have Same Connected Value

Edge Rejected |0->2|, Weight : 10, Both The Vertices Have Same Connected Value

Edge Rejected |2->0|, Weight : 10, Both The Vertices Have Same Connected Value

Edge Rejected |2->4|, Weight : 10, Both The Vertices Have Same Connected Value

Edge Rejected |4->2|, Weight : 10, Both The Vertices Have Same Connected Value

Kruskal's MST contains following edges::

|1->4|, Weight : 3

|0->1|, Weight : 4

|0->3|, Weight : 4

|2->3|, Weight : 5

Total Cost of MST is 16

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 5

=============================================

INVALID OPTION !!!

=============================================

1.Input Graph

2.Display Graph

3.Prim's Algorithm

4.Kruskal's Algorithm

0.Exit

Enter Choice : 0

=============================================

Program End