# PRIM’S ALGORITHM:

#include<iostream>

using namespace std;

class graph

{

public:

int graph[20][20], n, e;

void node();

void display();

void prims();

};

void graph::node()

{

int src, dest, cost;

cout << "Enter total vertices: "; //Taking input

cin >> n;

cout << "Enter total edges: ";

cin >> e;

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

graph[i][j] = 0; //Initialzing with zero

}

}

for (int i = 1; i <= e; i++)

{

cout << "Enter source, Destination and Cost: "; //Setting cost to their edges

cin >> src >> dest >> cost;

graph[src][dest] = cost;

graph[dest][src] = cost;

}

}

void graph::display() // Output Function

{

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

cout << " " << graph[i][j];

}

cout << endl;

}

}

void graph::prims()

{

int visited[20], src, min = 9999, dest = 0, total = 0, m = 0;

for (int i = 1; i <= n; i++)

{

for (int j = 1; j <= n; j++)

{

if (graph[i][j] == 0) //Initializing

{

graph[i][j] = 9999;

}

}

}

for (int i = 1; i <= n; i++)

{

visited[i] = 0;

}

cout << "Source for finding Minimum Spanning Tree: "; //Stating point for algo

cin >> src;

visited[src] = 1;

e = 0;

while (e < n - 1)

{

for (int i = 1; i <= n; i++)

{

if (visited[i] == 1)

{

for (int j = 1; j <= n; j++)

{

if (visited[j] != 1)

{

if (min > graph[i][j])

{

graph[i][j];

src = i;

dest = j;

}

}

}

}

}

visited[dest] = 1;

total = total + graph[src][dest];

cout << "Edge= " << src << "..." << dest << "cost=" << graph[src][dest] << endl;

e++;

}

cout << "Total cost of minimum spanning tree" << total;

}

int main()

{

graph g;

g.node();

g.display();

g.prims();

return 0;

}

# OUTPUT:

# KRUSKAL’S ALGORITHM:

#include<iostream>

#include<string.h>

using namespace std;

class Graph

{

char vertices[10][10];

int cost[10][10], no, edges;

public:

Graph();

void creat\_graph();

void display();

int Position(char[]);

void kruskal\_algo();

};

/\* Initialzing adj matrix with 999 \*/

/\* 999 denotes infinite distance \*/

Graph::Graph()

{

no=0;

for(int i=0;i<10;i++)

for(int j=0;j<10;j++)

{

cost[i][j]=999;

}

}

/\* Taking inputs for creating graph \*/

void Graph::creat\_graph()

{

char ans,Start[10],End[10];

int wt,i,j;

cout<<"Enter the number of vertices: ";

cin>>no;

cout << "Enter the number of edges: ";

cin >> edges;

cout<<"\nEnter the vertices: ";

for(i=0;i<no;i++)

cin>>vertices[i];

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

{

cout << "\nEnter Start and End vertex of the edge: ";

cin >> Start >> End;

cout << "Enter weight: ";

cin >> wt;

i = Position(Start);

j = Position(End);

cost[i][j] = cost[j][i] = wt;

}

}

/\* Displaying Cost matrix \*/

void Graph::display()

{

int i,j;

cout<<"\n\nCost matrix: ";

for(i=0;i<no;i++)

{

cout<<"\n";

for(j=0;j<no;j++)

cout<<"\t"<<cost[i][j];

}

}

/\* Retrieving position of vertices in 'vertices' array \*/

int Graph::Position(char key[10])

{

int i;

for(i=0;i<10;i++)

if(strcmp(vertices[i],key)==0)

return i;

return -1;

}

void Graph::kruskal\_algo()

{

int i,j,v[10]={0},x,y,Total\_cost=0,min,gr=1,flag=0,temp,d;

while(flag==0)

{

min=999;

for(i=0;i<no;i++)

{

for(j=0;j<no;j++)

{

if(cost[i][j]<min)

{

min=cost[i][j];

x=i;

y=j;

}

}

}

if(v[x]==0 && v[y]==0)

{

v[x]=v[y]=gr;

gr++;

}

else if(v[x]!=0 && v[y]==0)

v[y]=v[x];

else if(v[x]==0 && v[y]!=0)

v[x]=v[y];

else

{

if(v[x]!=v[y])

{

d=v[x];

for(i=0;i<no;i++)

{

if(v[i]==d)

v[i]=v[y];

}//end for

}

}

cost[x][y]=cost[y][x]=999;

Total\_cost=Total\_cost+min; /\* calculating cost of minimum spanning tree \*/

cout<<"\n\t"<<vertices[x]<<"\t\t"<<vertices[y]<<"\t\t"<<min;

temp=v[0]; flag=1;

for(i=0;i<no;i++)

{

if(temp!=v[i])

{

flag=0;

break;

}

}

}

cout<<"\nTotal cost of the tree= "<<Total\_cost;

}

int main()

{

Graph g;

g.creat\_graph();

g.display();

cout<<"\n\n\nMinimum Spanning tree using kruskal algo=>";

cout<<"\nSource vertex\tDestination vertex\tWeight\n";

g.kruskal\_algo();

return 0;

# } OUTPUT:

# 