***DSA ASSIGNMENT 15***

***PART 1***

***AIM:***

Store the graph using adjacency matrix representation and implement Prim’s algorithm to find minimum spanning tree.

***OBJECTIVES*:**

Implementing Prim’s algorithm and running it on weighted undirected graph. The algorithm is used to find the minimum spanning tree in a graph.

***THEORY:***

A minimum spanning tree (MST) or minimum weight spanning tree is a subset of the edges of a connected, edge-weighted undirected graph that connects all the vertices together, without any cycles and with the minimum possible total edge weight. That is, it is a spanning tree whose sum of edge weights is as small as possible.

 Prim’s algorithm is a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.A group of edges that connects two set of vertices in a graph is called cut in graph theory. *So, at* every step of Prim’s algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).

Example:



Minimum spanning tree:



***ALGORITHM*:**

1. Create a set mstSet that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn’t include all vertices
4. Pick a vertex *u* which is not there in mstSetand has minimum key value.
5. Include *u*to mstSet
6. Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge u-v is less than the previous key value of *v*, update the key value as weight of u-v

***CODE:***

#include<iostream>

#define INFINITY 999

#define MAX 6

using namespace std;

class tree

{

private:

int G[MAX][MAX], ST[MAX][MAX];

int v,e;

public:

tree();

void display();

void display\_ST();

void MST();

};

tree::tree() //default constructor

{

int i,j;

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

{

for(j=0;j<MAX;j++)

{

if(i==j)

G[i][j]=0;

else

G[i][j]=INFINITY;

ST[i][j]=0;

}

}

}

void tree::display()

{

int i,j;

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

{

for(j=0;j<MAX;j++)

{

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

}

cout<<"\n";

}

}

void tree::display\_ST() //display adjacency matrix of spanning tree

{

int i,j;

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

{

for(j=0;j<MAX;j++)

{

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

}

cout<<"\n";

}

}

void tree::MST()

{

int i,j,v0,v1,wt,count;

int process[MAX]={0},via[MAX]={0},distance[MAX]={0};

int sv,mindist,u,v2,Totalcost\_Graph=0, Totalcost\_MST=0;

//accepting the values

cout<<"\nEnter the no. Of vertices : ";

cin>>v;

cout<<"\nEnter the no. Of edges : ";

cin>>e;

cout<<"\nEnter the "<<e<<" edges : ";

cout<<"\nSource\tDest\tWeight\n";

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

{

cin>>v0>>v2>>wt;

Totalcost\_Graph+=wt; //adding total cost of graph

G[v0][v2]=wt;

G[v2][v0]=wt;

}

cout<<"\n";

cout<<"\nAdjacency Matrix of graph:\n\n";

display(); //display graph

cout<<"\nPlease Enter the Starting vertex : ";

cin>>sv;

process[sv]=1;

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

distance[i]=G[sv][i]; //copying contents of original graph in dist array

via[i]=sv; //

}

count=0;

while(count<v-1) //traverse until vertex-1

{

mindist=999;

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

{

if((distance[i]<mindist)&&(process[i]!=1))

{

mindist=distance[i];

v1=i;

}

}

u=via[v1];

if(u!=v1)

{

process[v1]=1;

ST[u][v1]=mindist;

ST[v1][u]=mindist;

Totalcost\_MST+=mindist; //adding total cost of minimum spanning tree

for(j=0;j<v;j++)

{

if((G[v1][j]<distance[j]) && (process[j]!=1))

{

distance[j]=G[v1][j];

via[j]=v1;

}

}

count++; //incrementing count

}//while

}

cout<<"\nSpanning tree matrix is\n\n";

display\_ST(); //displaying spanning tree

cout<<"\n\nThe Edges added in the minimum spanning tree";

cout<<"\n\nSource\tDest\tWeight\n";

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

{

for(j=0;j<MAX;j++)

{

if((j>=i)&&ST[i][j]!=0)

cout<<i<<"\t"<<j<<"\t"<<ST[i][j]<<"\n";

}

// cout<<"\n";

}

cout<<"\nTotalcost\_Graph is = "<<Totalcost\_Graph; //displaying

cout<<"\nTotalcost\_MST is = "<<Totalcost\_MST;

}

int main()

{

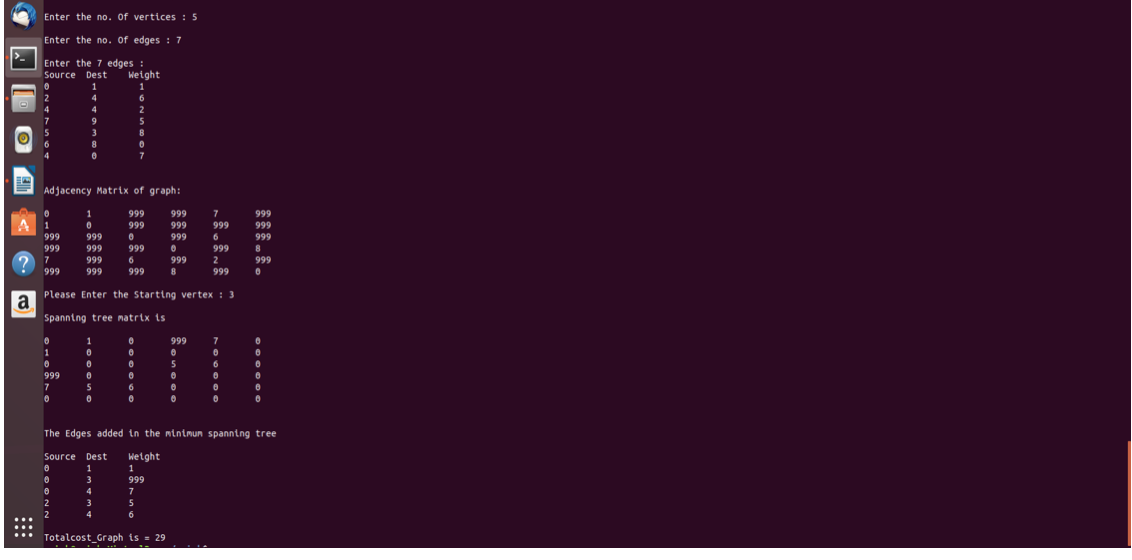
tree t;

t.MST();

return 0;

}

***OUTPUT:***



***CONCLUSION:***

|  |  |
| --- | --- |
| **Minimum edge weight data structure** | **Time complexity (total)** |
| Adjacency matrix, searching | O(|V|2) |
| Binary heap and adjacency list | O((|V|+|E|)log|V|)=O(|E|log|V|) |
| Fibonacci heap and adjacency list | O(|E|+|V|log|V|) |