Name-Harshita Totala Roll No-SEITA14

Div-A

Assignment-7
Minimum Spanning tree

AIM : Implementation of Minimum Spanning tree using Prims and Kruskals Algorithm

DETAILED PROBLEM STATEMENT:

Represent a graph of your college campus using adjacency list /adjacency matrix. Nodes should represent the various departments/institutes and links should represent the distance between them. Find minimum spanning tree

a) Using Kruskal's algorithm.
b) Using Prim's algorithm.

OBJECTIVE

1. To study Graph theory.
2. To study different graph traversal methods.
3. To understand the real time applications of graph theory.
4. To Implement a MST using Prims and Kruskals algorithm

OUTCOME :

1. Understand Non-linear data structure - graph.
2. Represent a graph using adjacency list / adjacency matrix.
3. Implement Prim's and Kruskal's Algorithm 4. Identify applications of Minimum Spanning Trees.
5. Analyze the Time and Space complexity.

THEORY:

Introduction Graph :

- **Definition :** A graph is a triple G = (V,E, φ)where • V is a finite set, called the vertices of G, E is a finite set, called the edges of G, and,φ is a function with domain E and codomain P2(V).
- **Loops:** A loop is an edge that connects a vertex to itself.
- **Degrees of vertices**: Let G = (V,E, φ) be a graph and v ∈ V a vertex. Define the degree of v, d(v) to be the number of e ∈ E such that v ∈ φ(e); i.e., e is Incident on v.
- **Directed graph:** A directed graph (or digraph) is a triple D = (V,E, φ) where V and E are finite sets and φ is a function with domain E and codomain V × V . We call E the set of edges of the digraph D and call V the set of vertices of D.
- **Path:** Let G = (V,E, φ) be a graph.

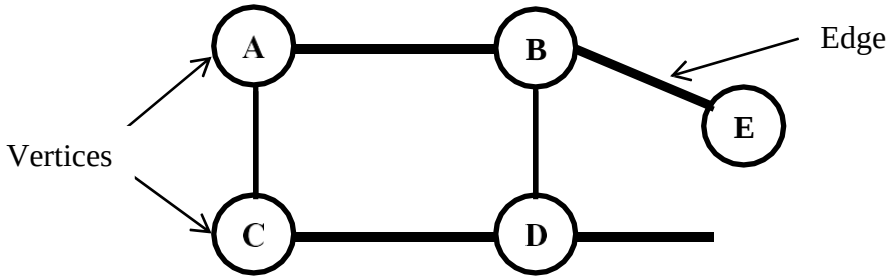
  Let e1, e2, . . . , en−1 be a sequence of elements of E (edges of G) for which there is a sequence a1, a2, . . . , an of distinct elements of V (vertices of G) such that φ(ei) = {ai, ai+1} for i = 1, 2, . . . , n − 1. The sequence of edges e1, e2, . . . , en−1 is called a path in G. The sequence of vertices a1, a2, . . . , an is called the vertex sequence of the path.

Circuit and Cycle: Let G = (V, E, φ) be a graph and let e1, . . . , en be a trail with vertex sequence a1, . . . , an, a1. (It returns to its starting point.) The subgraph G′of Ginduced by the set of edges{e1,...,en}is called a circuit of

G. The length of the circuit is n.

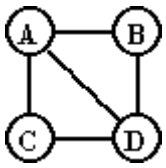
e.g.:

This graph G can be defined as G = (V, E) Where V= {A,B,C,D,E} and E={(A,B),(A,C)(A,D),(B,D),(C,D),(B,E),(E,D)}.



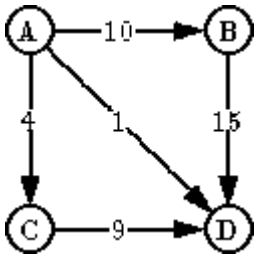
Different representations of graph:

1. **Adjacency matrix:** Graphs *G* = (*V, E*) can be represented by adjacency matrices *G* [$v_1..v_{|V|}$, $v_1..v_{|V|}$], where the rows and columns are indexed by the nodes, and the entries *G* [v_i, v_j] represent the edges. In the case of unlabeled graphs, the entries are just Boolean values.



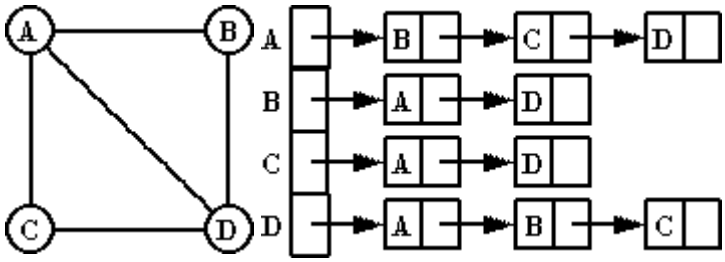
	A	**B**	**C**	**D**
A	0	1	1	1
B	1	0	0	1
C	1	0	0	1
D	1	1	1	0

In case of labeled graphs, the labels themselves may be introduced into the entries.



	A	**B**	**C**	**D**
A	∞	10	4	1
B	∞	∞	∞	15
C	∞	∞	∞	9
D	∞	∞	∞	∞

2. **Adjacency List:** A representation of the graph consisting of a list of nodes, with each node containing a list of its neighboring nodes.



- **Spanning Tree:**

A Spanning Tree of a graph G = (V, E) is a sub graph of G having all vertices of G and no cycles in it.

Minimal Spanning Tree: The cost of a graph is the sum of the costs of the edges in the weighted graph. A spanning tree of a graph G= (V, E) is called minimal cost spanning tree or simply the minimal spanning tree of G if its cost is minimum.

- When a graph G is connected, depth first or breadth first search starting at any vertex visits all the vertices in G.
- The edges of G are partitioned into two sets i.e. T for the tree edges & B for back edges. T is the set of tree edges and B for back edges. T is the set of edges used or traversed during the search & B is the set of remaining edges.
- The edges of G in T form a tree which includes all the vertices of graph G and this tree is called as spanning tree.

Definition: Any tree, which consists solely of edges in graph G and includes all the vertices in G, is called as spanning tree. Thus for a given connected graph there are multiple spanning trees possible. For maximal connected graph having ‗n' vertices the number of different possible spanning trees is equal to n.

Cycle: If any edge from set B of graph G is introduced into the corresponding spanning tree T of graph G then cycle is formed. This cycle consists of edge (v, w) from the set B and all edges on the path from w to v in T.

There are many approaches to computing a minimum spanning tree. We could try to detect cycles and remove edges, but the two algorithms we will study build them from the bottom-up in a greedy fashion.

Prim's Algorithm – starts with a single vertex and then adds the minimum edge to extend the spanning tree.

Kruskal's Algorithm – starts with a forest of single node trees and then adds the edge with the minimum weight to connect two components.

• **Prim's algorithm:** Prim‘s algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. The algorithm was discovered in 1930 by mathematician Vojtech Jarník and later independently by computer scientist Robert C. Prim in 1957 and rediscovered by Edsger Dijkstra in 1959. Therefore it is sometimes called the DJP algorithm, the Jarník algorithm, or the Prim-Jarník algorithm.

Applications of spanning Trees:

- To find independent set of circuit equations for an electrical network. By adding an edge from set B to spanning tree we get a cycle and then Kirch off‘s second law is used on the resulting cycle to obtain a circuit equation.
- Using the property of spanning trees we can select the spanning tree with (n-1) edges such that total cost is minimum if each edge in a graph represents cost.
- Analysis of project planning
- Identification of chemical compounds
- Statistical mechanics, genetics, cybernetics, linguistics, social sciences

ALGORITHMS/ PESUDOCODE :

- **Prims algorithm :**

Data structure used:

Array: Two dimensional array (adjacency matrix) to store the adjacent vertices & the weights associated edges

One dimensional array to store an indicator for each vertex whether visited or not. #define max 20 int adj_ver[max][max], int edge_wt[max][max], int ind[max]

- **Algorithm to generate spanning tree by Prim's**

Prims(Weight ,Vertex)

//Weight is a two dimensional array having V no of rows and columns. KOWN ,cost , PRIEV are the l vectors.

Step 1: Repeat for I = 1 to V

i. KNOWN[I] = 0 ii.
PREV[I] = 0 iii.
cost[I] = 32767

Step 2: current= 1 //starting vertex of prims

Step 3: Total_V =0 //total vertex considered till the time

Step 4: KNOWN[current] =1 // Start is known now

Step 5: Repeat thru step 6 while Total_v != Vertex

i. mincost= 32767 ii.
Repeat for I =1 to V
iii. If (weight[current][I] != 0 AND KNOWN[I] =0)
i. If (cost[I] >= weight[current][I])

a. cost[I] = weight[current][I]
ii. end if
iv. Repeat for I =1 to V //finding min cost edge from current vertices
i. If (KNOWN[I] = 0 AND cost[I] <= mincost)
a. mincost =cost[I] //if min is Cost[i]
b. current = I //next node visited is I
ii. end if
v. end if
Step 6: KNOWN[current] = 1
Step 7: Toatal_v= Total_v + 1
Step 8: mincost = 0
Step 9: Repeat for I = 1 to V
i. WRITE(I , PREV[I]) //display mst edges
ii. If cost[I] != 32767
a. mincost =mincost + cost[I]
Step 10 : display final mincost
Step 11: end of prims

Trace of Prim's algorithm for graph G1 starting from vertex 1

Step No.	**Set A**	**Set (V-A)**	**Min cost Edge (u, v)**	**Cost**	**Set B**
Initial	{1}	{2,3,4,5,6,7}	--	--	{}
1	{1,2}	{3,4,5,6,7}	(1, 2)	1	{(1, 2)}
2	{1, 2, 3}	{4, 5, 6, 7}	(2, 3)	2	{(1,2},(2,3)}
3	{1,2,3,5}	{4, 6, 7}	(2, 5)	4	{(1,2),(2,3),(2,5)}
4	{1,2,3,5,4}	{6, 7}	(1,4)	4	{(1,2),(2,3),(2,5),(1,4)}
5	{1,2,3,5,4,7}	{6}	(4,7)	4	{(1,2),(2,3),(2,5),(1,4),(4,7)}
6	{1,2,3,5,4,7.6}	{ }	(7,6)	3	{(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}
		Total Cost		17	

Thus the minimum spanning tree for graph G1 is : A = { 1,2,3,4,5,7,6} B
={(1,2),(2,3),(2,5),(1,4),(4,7),(7,6)}, total Weight: 1+2+4+3+4+3=17

Kruskal's Algorithm:

- **Prerequisite for Kruskals** struct edge

```
{
    Number v1,v2,wt
} edge
```

- **Create edge Matrix**

AdjToEdges(int weight[][MAX], int n, edge E[])

Step 1: for i=0 to n do
Step 2: for j=i+1 to n do
i.i f(weight[i][j])
a. edge_matrix[k]. start=i
b. edge_matrix[k].end =j
c. edge_matrix[k++].Value=G[i][j] ii. end if
Step 3:end for
Step 4: end for return k;
Step 5: end AdjToEdges

- **Function to sort the edges according to weights Algorithm**

SortEdges(edge edge_matrix[], no_edge)

Step 1:for i=0 to no_edge
Step 2: for j=i+1 to no_edge
i. if(edge_matrix[i].value> edge_matrix[j].value)
a. t=edge_matrix[i]
b. edge_matrix[i]= edge_matrix[j]
c. edge_matrix[j]=t
ii. end if
Step 3 : end for j
Step 4: end for i
Step 5: end sort edge

Kruskals (G, N)

//G is a pointer to head of the adjacency List. N is max. number of vertices.

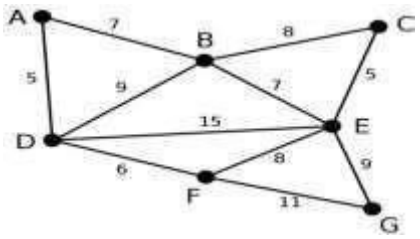
// L and K =0
Step 1: mincost = 0

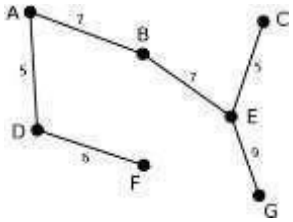
Step 2: EARRAY(G)

Step 3:Repeat for I=1 to N
i. set[I] = I

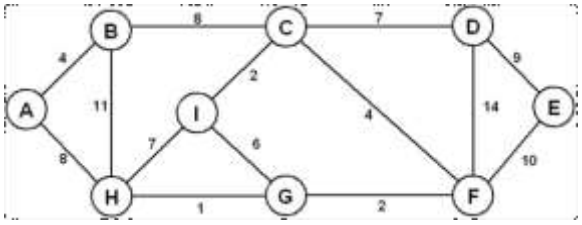
Step 4: Repeat Thru step 6 while L < Vertex-1 //select the min cost edge tillno of edges =Vertex-1
i. T = Edge_mat[K] // select min cost edge
ii. K = K+1 // to select next edge iii. Repeat for I = 1 to N
iv. PV1=FIND(set,T.V1) //check the set Membership of V1
v.PV2=FIND(set,T.V2) // check the Set membership of V2
vi. if (PV1 != PV2) //if both v1 and v2 are belongs to different set
//they are not forming cycle and can be added to fina
a.WRITE(V1(T),V2(T) , D(T))
b. mincost = micost + Dist[T]
c. L=L+1
d. for J=1 to N
1. If (C[J] = PV2)
i. C[J] □ PV1
2. end if
e. end for
vii. end if

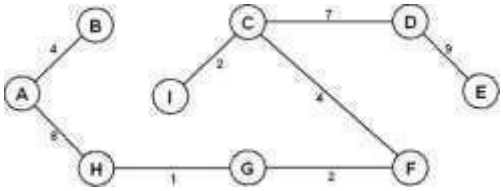
Step 5: display mincost
Step 6: end Kruskals





Cost of MST - 39





Cost of MST -37

- **Sample Input/Output:**

Input: Enter vertices of a graph: 1 2 3 4 5 6 7
Enter vertex wise adjacent vertices & cost of edges.

Vertex	Adjacent Vertex	Cost
Pune (1)	2	1
	4	4
	0	
Mumbai (2)	**1**	**1**
	4	**6**
	5	**4**
	3	**2**
	0	
Bangalore (3)	**5 2 5 2**	
	6	**8**
	0	
Hyderabad (4)	**1**	**4**
	2	**6**
	5	3
	5	3
	7	4
	0	
Chennai (5)	**2**	**4**
	3	**5**
	6	**8**
	7	**7**
	4	**3**
	0	
Delhi (6)	**3**	**8**
	5	**8**

	7 0	**3**
Ahmadabad (7)	**4**	**4**
	5	**7**
	6 0	**3**

OUTPUT : Display of adjacent matrix_adj vertices & cost of associated edges.

Pune (1): (0, 0) (2, 1) (0,0) (4, 4) (0, 0) (0, 0) (0, 0)
Mumbai (2): (1, 1) (0, 0) (3, 2) (4, 6) (5, 4) (0, 0) (0, 0)
Bangalore (3): (0, 0) (2, 2) (0, 0) (0, 0) (5, 5) (6, 8) (0, 0)
Hyderabad (4): (1, 4) (2, 6) (0, 0) (0, 0) (5, 3) (0, 0) (7, 4)
Chennai (5): (0, 0) (2, 4) (3, 5) (4, 3) (0, 0) (6, 8) (7, 7)
Delhi (6): (0, 0) (0, 0) (3, 8) (0, 0) (5, 8) (0, 0) (7, 3)
Ahmadabad (7): (0, 0) (0, 0) (0, 0) (4, 4) (5, 7) (6, 3) (0, 0)

Total cost: 17

Testcases :

a) Display the total number of comparisons required to construct the graph in computer memory.
b) Display the results as given in the sample o/p above.
c) Finally conclude on time & time space complexity for the construction of the graph and for generation of minimum spanning tree using Prim‘s algorithm.

Time Complexity:

Fortheconstructionofanundirectedgraphwith_n‘verticesand_e‘edgesusing adjacency list is O(n+e),since fore very vertex _v‘in Gweneed to store all adjacent edges to vertex v.

- ✓ In Prim‘s algorithm to get minimum spanning tree from an undirected graph with_n‘ vertices using adjacency matrix is $O(n^2)$.
- ✓ Using Kruskal‘salgorithm using adjacency matrix = $O(n^2)$.
  using adjacency list=O(eloge)

CODE FOR PRIM'S ALGORITHM:

```
#include<iostream>
using namespace std;

class graph
{
 int G[20][20],n;

 public:

 void accept()
 {
    int i,j,e;
    int src,dest,cost;
      cout<<"\nEnter the no. of vertices: ";
      cin>>n;

      for(i=0;i<n;i++)
      {
      for(j=0;j<n;j++)
      {
      G[i][j]=0;
      }
      }

      cout<<"\nEnter the no. of  Edges: ";
      cin>>e;



      for(i=0;i<e;i++)
      {
      cout<<"\nEnter Source: ";
       cin>>src;
      cout<<"\nDestination: ";
       cin>>dest;
      cout<<"\nCost: ";
      cin>>cost;

      G[src][dest]=cost;
      G[dest][src]=cost;
     }
```

```
}




void display()
{
  int i,j;
 for(i=0;i<n;i++)
 {
  cout<<"\n";
  for(j=0;j<n;j++)
  {
      cout<<"\t"<<G[i][j];
  }
 }
}



void prims()
{
  int i,j,R[20][20];
  int src,dest,cost,count,min;
  int total=0;
  int visited[20];

  for(i=0;i<n;i++)
  {
      for(j=0;j<n;j++)
      {
          if(G[i][j]==0)
          {
              R[i][j]=999;
          }
      else
      R[i][j]=G[i][j];
      }
  }



  for(i=0;i<n;i++)
  {
```

```
        visited[i]=0;
    }

    cout<<"\nEnter start vertex: ";
    cin>>src;
    visited[src]=1;

    count=0;
    while(count<n-1)
    {
    min=999;

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

    if(visited[i]==1)
    for(j=0;j<n;j++)
    {
    if(visited[j]!=1)
    {
    if(min>R[i][j])
    {
        min=R[i][j];
        src=i;
        dest=j;
    }
    }
    }
    }


    cout<<"\nEdge from "<<src<<" to "<<dest<<" \twith cost:  "<<min;
    total=total+min;
    visited[dest]=1;
    count++;
    }

        cout<<"\nTotal Cost: "<<total<<"\n";
    }

};


  int main()
  {
```

```
graph g;
g.accept();
g.display();
g.prims();
}
```

OUTPUT FOR PRIM'S ALGORITHM:

Enter the no. of vertices: 4

Enter the no. of Edges: 5

Enter Source: 0

Destination: 1

Cost: 4

Enter Source: 1

Destination: 2

Cost: 6

Enter Source: 0

Destination: 2

Cost: 7

Enter Source: 3

Destination: 1

Cost: 1

Enter Source: 3

Destination: 2

Cost: 5

```
0    4    7    0
4    0    6    1
7    6    0    5
0    1    5    0
Enter start vertex: 0

Edge from 0 to 1      with cost:  4
Edge from 1 to 3      with cost:  1
Edge from 3 to 2      with cost:  5
Total Cost: 10
```

CODE FOR KRUSKAL'S ALGORITHM:

```
#include<iostream>
#define INFINITY 999
using namespace std;

class kruskal{
 typedef struct graph{
     int v1, v2, cost;
        }GR;
        GR G[20];

        public:
        int tot_edges, tot_nodes;
          void create();
          void spanning_tree();
          void get_input();
          int minimum(int);
};

int find(int v2, int parent[]){

    while(parent[v2!=v2]){
            v2 = parent[v2];
        }
        return v2;
}

void un(int i,int j,int parent[]){
    if(i<j)
     parent[j]=i;
     else
        parent[i]=j;
}


void kruskal :: get_input(){
        cout<<"\nEnter number of nodes : "<<endl;
            cin>>tot_nodes;
            cout<<"Enter number of edges : "<<endl;
            cin>>tot_edges;
}

void kruskal :: create(){
    for(int k=0; k<tot_edges; k++){
```

```
        cout<<"Enter V1 and V2 : ";
          cin>>G[k].v1;
          cin>>G[k].v2;
        cout<<"Enter cost : "<<endl;
          cin>>G[k].cost;
        }
}

int kruskal::minimum(int n){
    int i,small,pos;
    small=INFINITY;
        pos=-1;
        for(i=0;i<n;i++){
            if(G[i].cost<small){
                small=G[i].cost;
                pos=i;
            }
        }
        return pos;
}

void kruskal::spanning_tree(){

    int count,k,v1,v2,i,j,tree[10][10],pos,parent[10];
    int sum;
    count=0;
    k=0;
    sum=0;
        for(i=0;i<tot_nodes;i++)
            parent[i]=i;
    while(count!=tot_nodes-1){
                pos=minimum(tot_edges);
            if(pos==-1)
            break;
               v1=G[pos].v1;
               v2=G[pos].v2;
               i=find(v1,parent);
               j=find(v2,parent);
               if(i!=j){
                   tree[k][0]=v1;
                   tree[k][1]=v2;
                k++;
                count++;
                sum+=G[pos].cost;
```

```
                un(i,j,parent);
            }
            G[pos].cost=INFINITY;
        }
    if(count==tot_nodes-1){
         cout<<"\nSpanning tree is: "<<endl;


            for(i=0;i<tot_nodes-1;i++){
                cout<<"|"<<tree[i][0];
                cout<<" ";
                cout<<tree[i][1]<<"|"<<endl;
            }
        cout<<"\n--------------------------------- \n";
        cout<<"Cost of spanning tree is: "<<sum<<endl;
        }
        else{
            cout<<"There is no spanning tree "<<endl;
        }
}


int main(){

    kruskal obj;
    obj.get_input();
    obj.create();
    obj.spanning_tree();
}
```

OUTPUT FOR KRUSKAL'S ALGORITHM

Enter number of nodes :
4
Enter number of edges :
5
Enter V1 and V2 : 1 2
Enter cost :
4
Enter V1 and V2 : 2 3
Enter cost :

```
7
Enter V1 and V2 : 3 4
Enter cost :
7
Enter V1 and V2 : 4 1
Enter cost :
3
Enter V1 and V2 : 3 5
Enter cost :
5

Spanning tree is:
|4 1|
|1 2|
|3 5|

---------------------------------
Cost of spanning tree is: 12
```

CONCLUSION:

Hence we have studied the concept of prims algorithm using adjacency matrix and Kruskal's algorithm by using adjacency list.