Week-2 Presentation

Greedy Algorithms on Graph

Prim's Algorithm

Kruskal Algorithm

Single Source shortest Path Algorithm (Dijkstra's Algorithm)

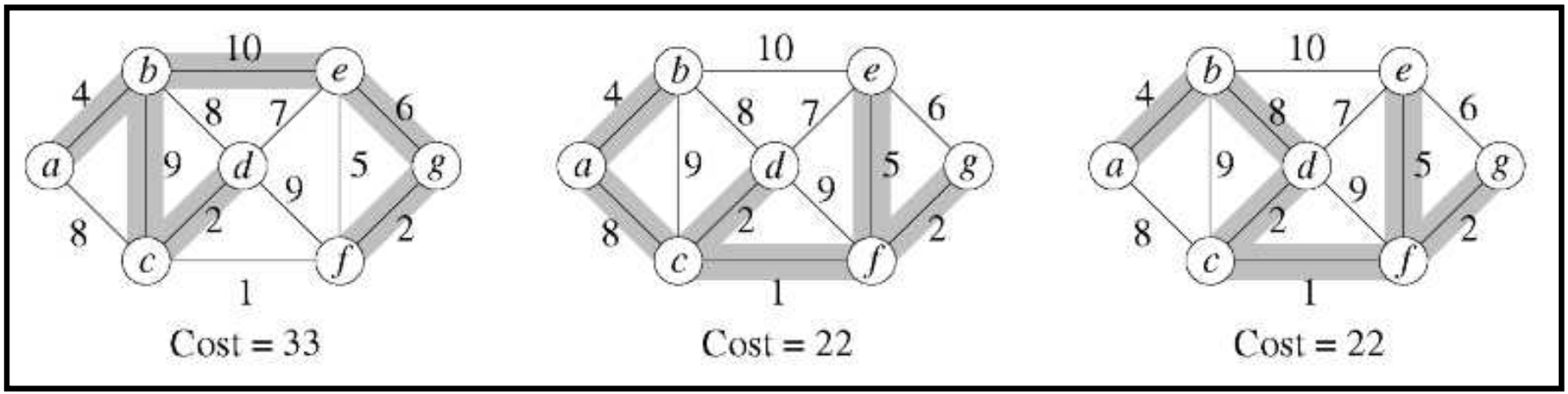
Minimum Cost Spanning Tree

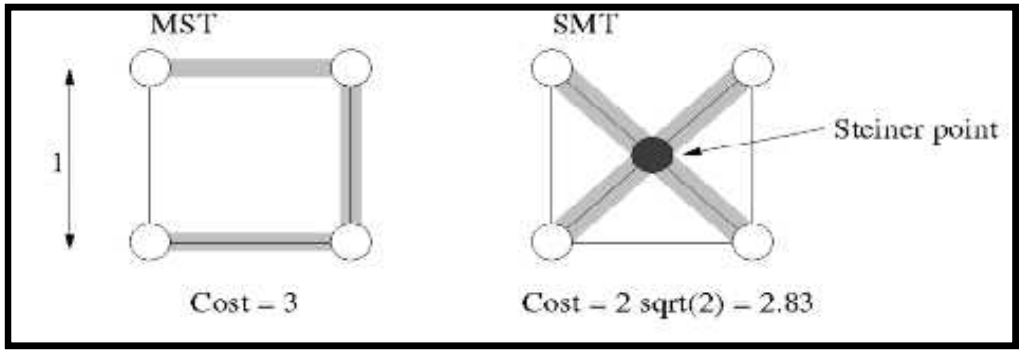
- Given a graph G =(V,E) V=Number of vertices and E=Number of edges, then Spanning tree is tree generated from graph, with characteristics
  - All the vertices present in the tree
  - There is no cycle in the tree i.e., e=v-1
- Subset of edges, that forms a tree, where total cost of edges is minimum.
- *Minimum cost spanning tree is a spanning tree with "edges of minimum cost".*
- **Applications:**
- To transmit the message without broadcasting.
- To generate travel plan in minimum cost.
- Designing network, home electric wiring etc.
- Approaches: Prims' Method and Kruskals' Method.

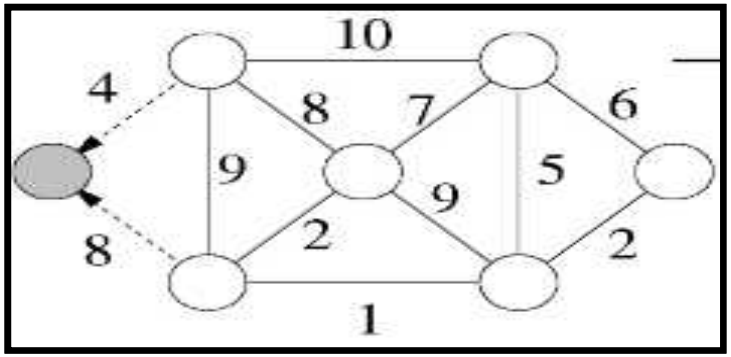
Minimum Cost Spanning Tree

- Principle:
  - Select an edge of minimum cost. The edge will derive two vertices.
  - Continue the process from one of the vertex by selecting the next edge of minimum cost.
  - Once the vertex is visited, then it is marked.
  - The process will attempt to visit unmarked vertices and terminates when all the vertices are visited.
  - Process guarantees no cycle in the tree.
- *"What if the process of tree generation starts from any arbitrary vertex".*
- **Motivation: To join points as cheaply as possible: Applications in clustering and networking**
- **Acyclic graph to connect all nodes in minimum cost.**
- **One of the term in U.S. legal code (AT&T)**

Graphs for Examples







Spanning tree: Free tree

- A free tree has following properties

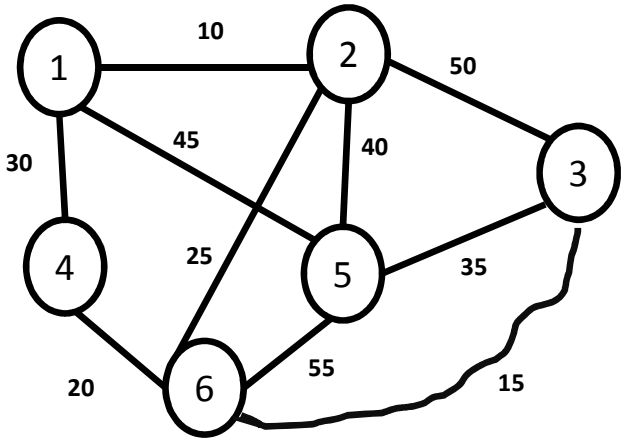
1. Exactly n-1 edges for “n” vertices
2. There exists a unique path between two vertices.
3. By adding an edge, a cycle will be created in free tree. Breaking any edge on the cycle restores the free tree.

- *Greedy: Minimization problem.*

1. Repeated selection: of minimum cost edges with certain test cases.
2. Once decision of adding edge is finalized, it cannot be revoked.
3. Basic idea: To generate subset of “E” connecting all “V”

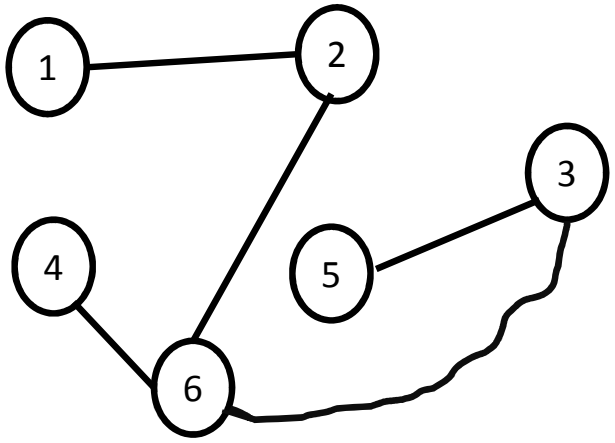
Minimum Cost Spanning Tree: Prim's method

- Example:
- Consider the following graph:



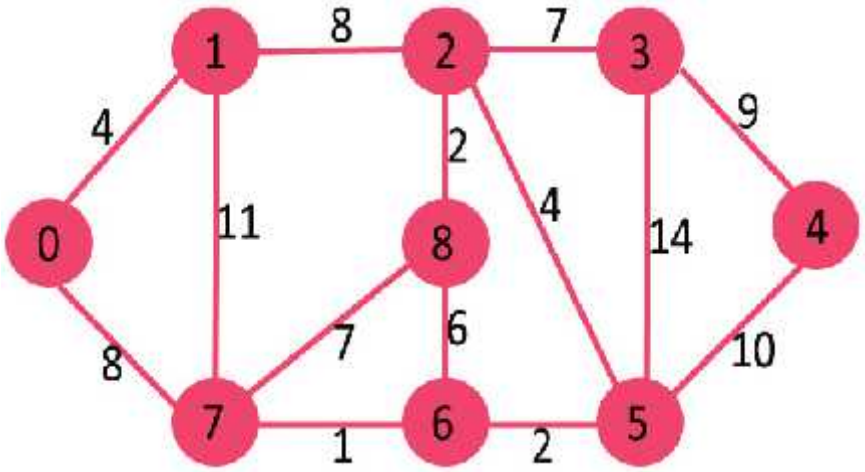
	1	2	3	4	5	6
1	99	10	99	30	45	99
2	10	99	50	99	40	25
3	99	50	99	99	35	15
4	30	99	99	99	99	20
5	45	40	35	99	99	55
6	99	25	15	20	55	99

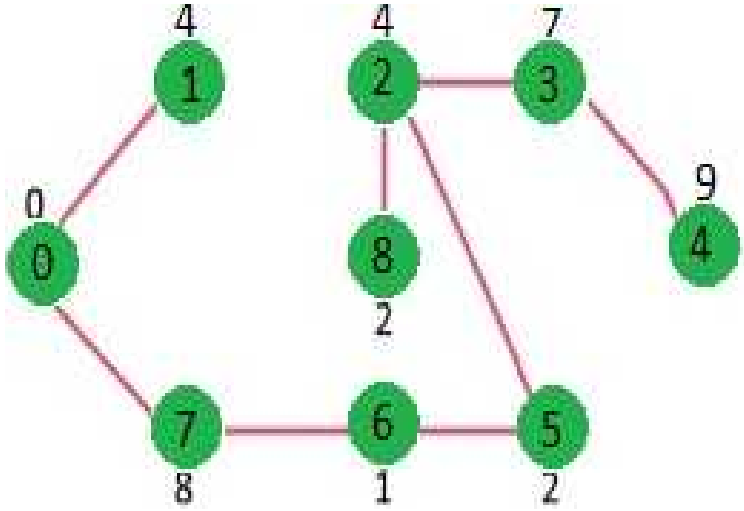
- ***Select an edge of minimum cost.***
- ***From selected vertex continue selecting edge of minimum cost***



Minimum Cost Spanning Tree: Prim's Algorithm

- Designed by Robert C. Prim in 1957 and was modified by Dijkstra so also called as DJP algorithm.





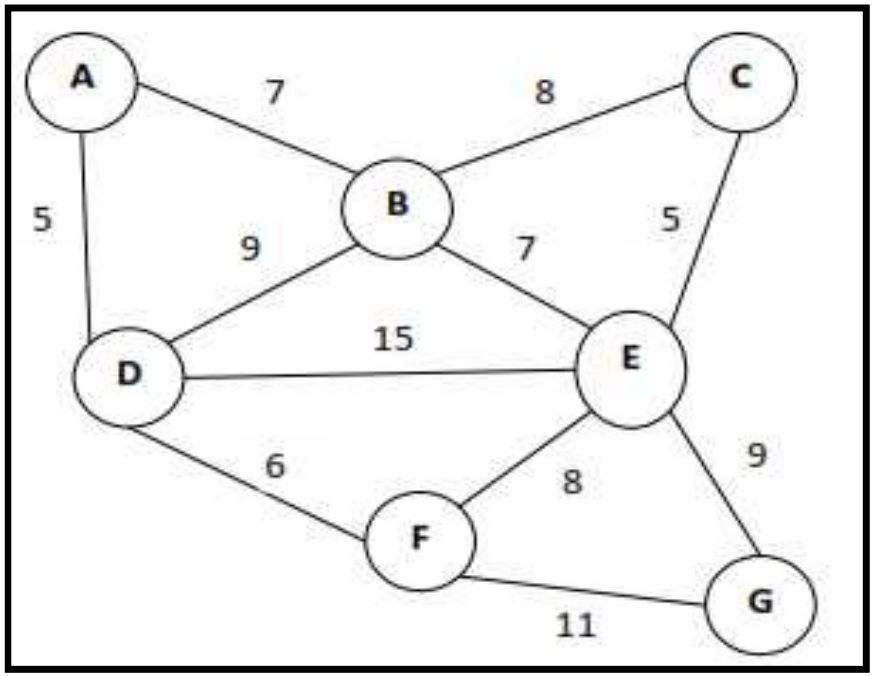
Minimum Cost Spanning Tree: Kruskals' Algorithm

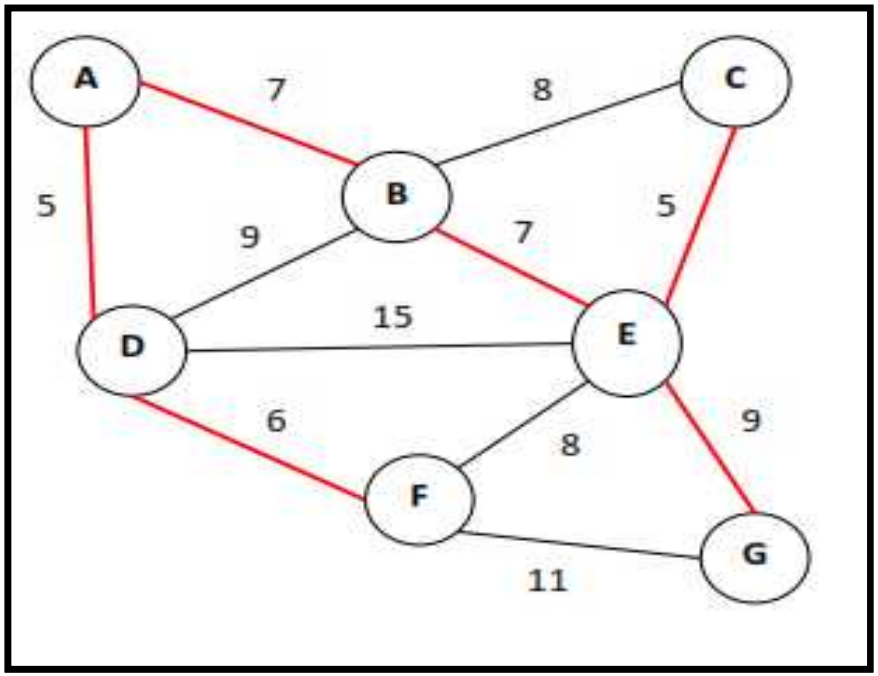
- Principle: Select an edge of minimum cost. At each step add a new edge with next minimum cost to graph, such that it does not generate cycle.
- Kruskals' algorithm, grows like forest and then generates a tree.
- Integration plays major role.
- Implementation:
- From the set of edges, select an edge of minimum cost
- Again continue the process with an edge of next minimum cost, add edge if not generating cycle.

Kruskals' Algorithm

- **Process-1: To find an edge of minimum cost**
- Sort the edges in ascending order of weight: O(e * log e)
- **Process-2: To check by adding an edge cycle is generated.**
- **Union-Find Data structure is used to check if cycle is generated.**
- The algorithm creates set for each vertex: [1] [2] [3] [4] [5] [6] etc.
- When an edge [u,v] is selected Find[u] will return the set in which vertex "u" is present and Find[v] will return the set in which vertex "v" is present.
- *If returned set are distinct*, then edge can be added and sets can merged.

Kruskals' Algorithm: Example

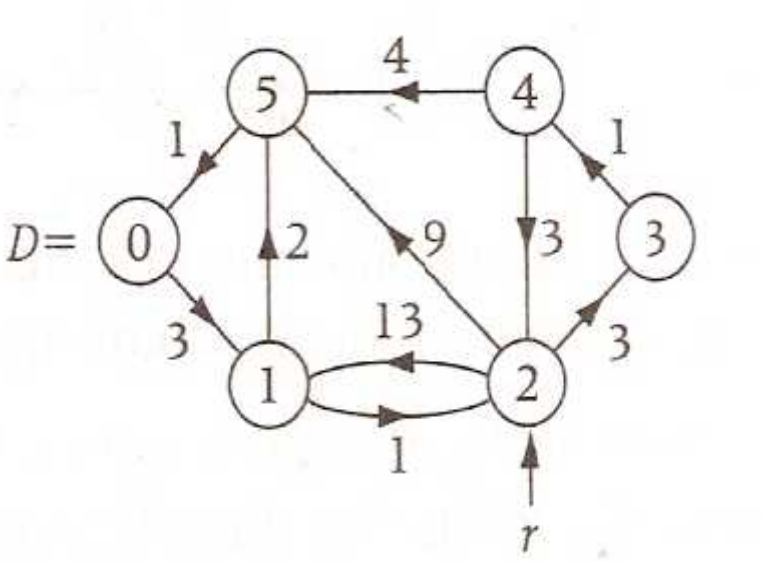




Algorithm

```
Algorithm Kruskal(E,cost,n:mincost,t)
{Step 1:
        for i = 1 to n do
                comp[1] = i;
 Step 2:
        i= 0; mincost = 0
        while (i < n-1) and (E is not empty) do
        {
                #Find a minimum cost edge from E, let this edge be (u,v)
                j = find(u);            k=find(v) //Functions
                if(j ≠ k) then
                {
                        i=i+1;
                        t[i,1] = u; t[i,2] = v; t[i,3] = cost[u,v]
                        mincost=mincost + cost[u,v]
                        union(j,k) //Function
                }
         }//end of while
        if (i≠n-1) then
         write ("No spanning tree")
        else
          return(mincost) //Optional
}
```

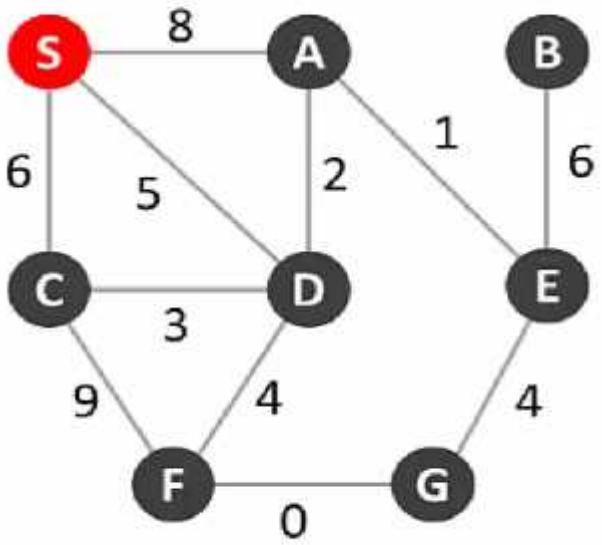
Practice examples

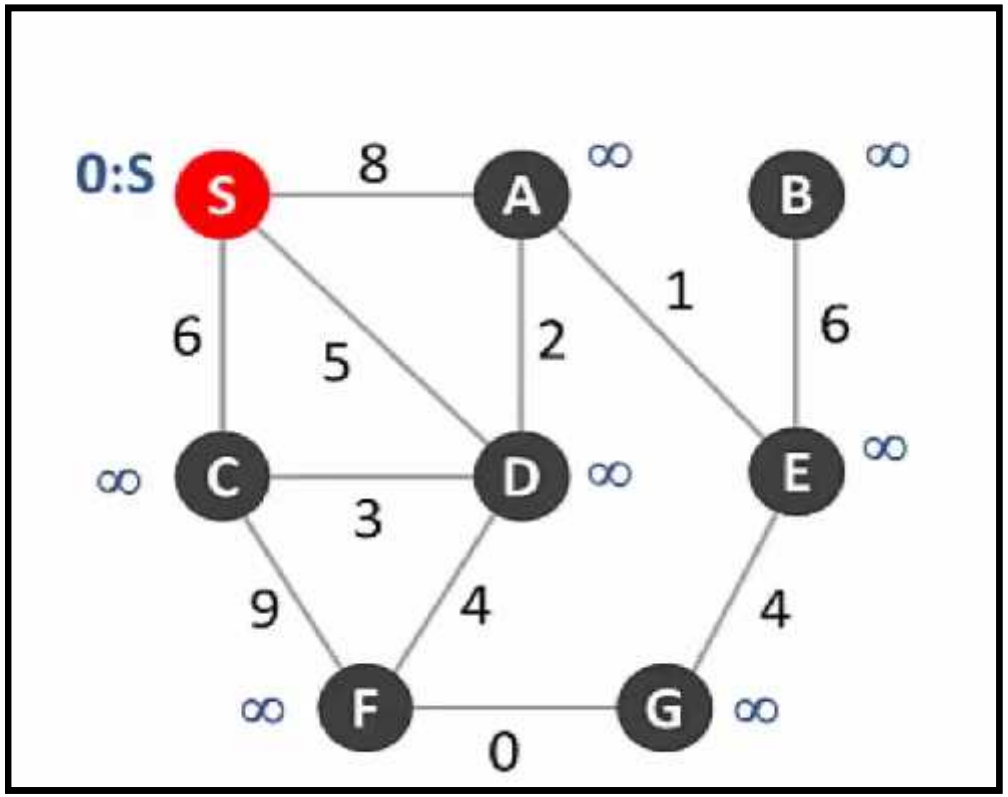


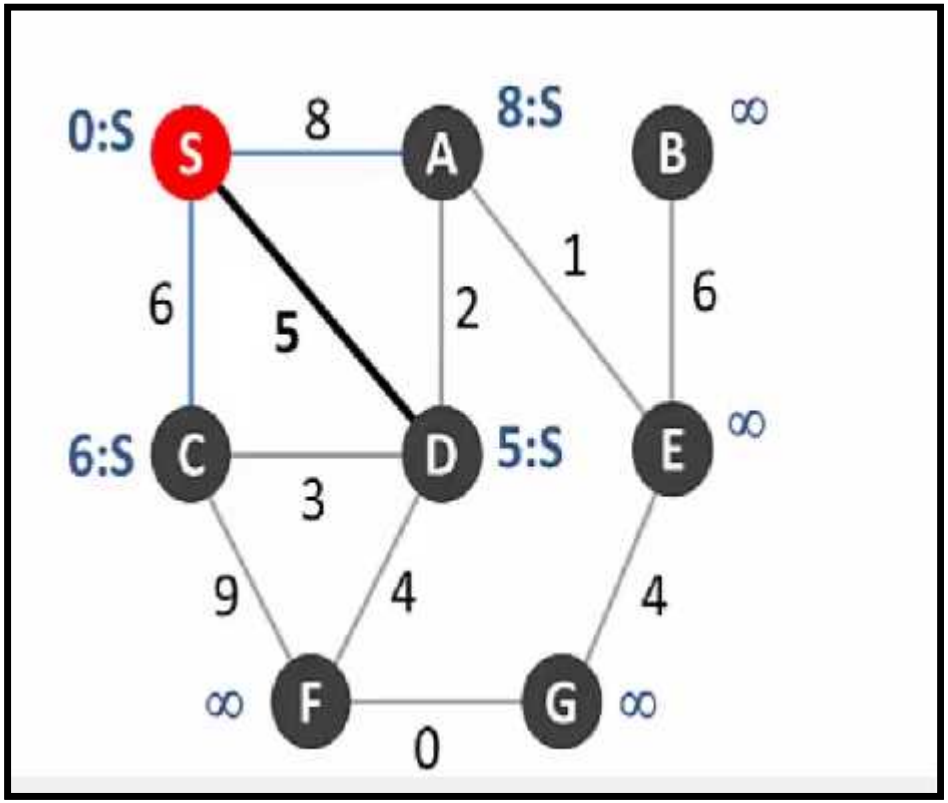
Single Source Shortest Path

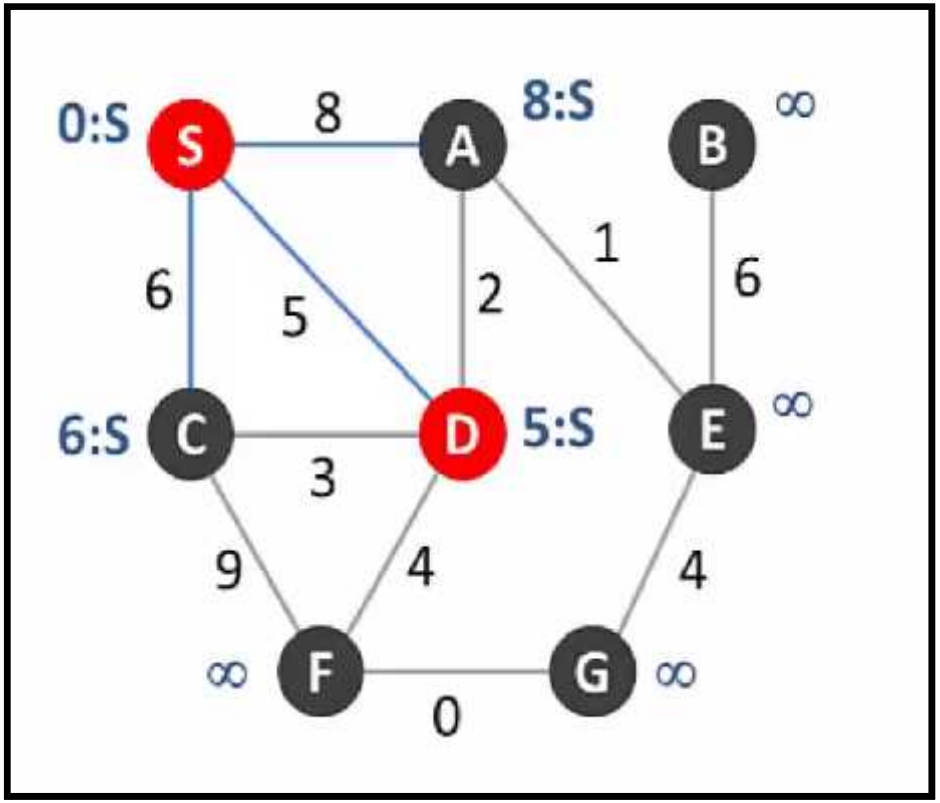
- An approach to find shortest path from given source vertex to all possible destinations.
- Minimization problem: To find the path of minimum cost
- Path may be direct or indirect
- Various algorithms:
- Dijkstra's algorithm can be used if no negative edges are present.
- Bellman Ford algorithm can be used if negative edges are present.
- Floyd's and Warshall's algorithm is extension and called as all pair shortest path algorithm, if no negative edges are given.

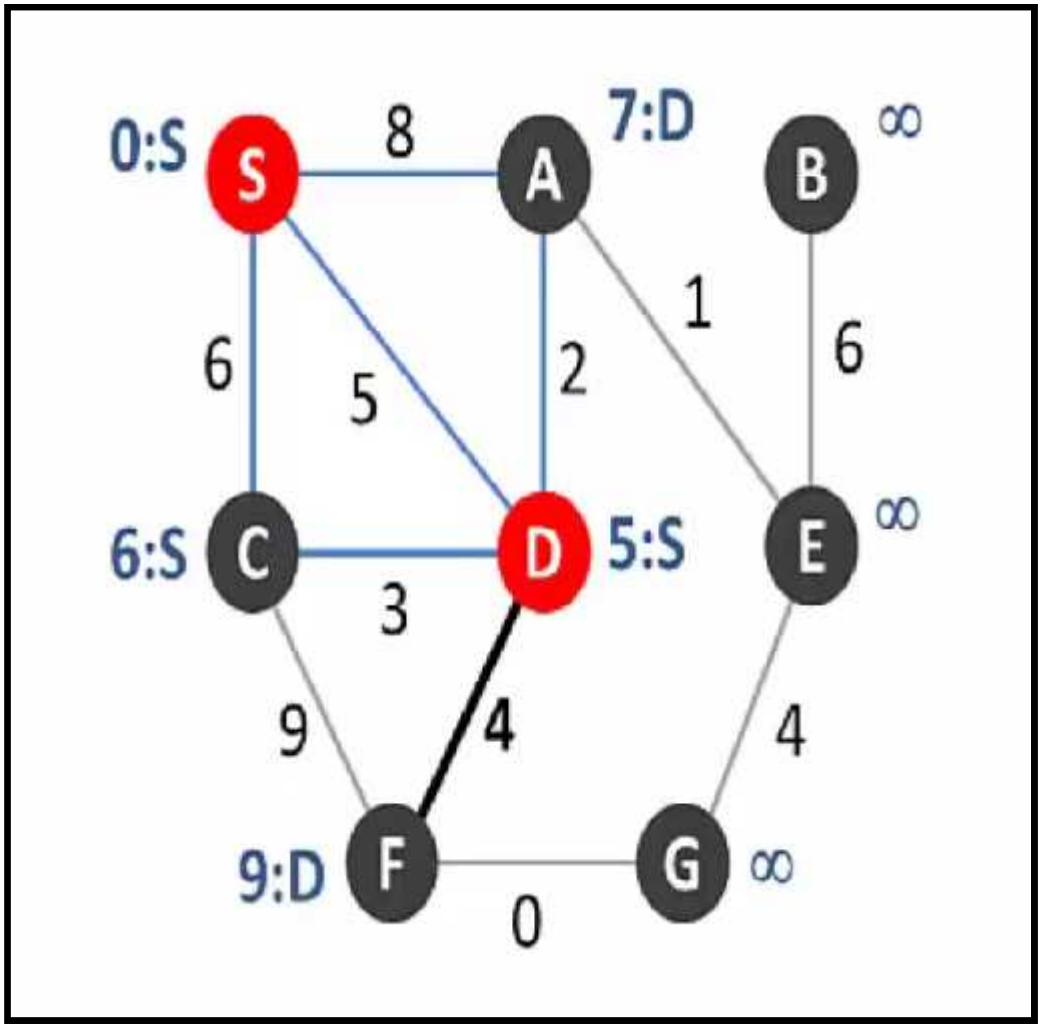
Dijkstra's on an undirected graph

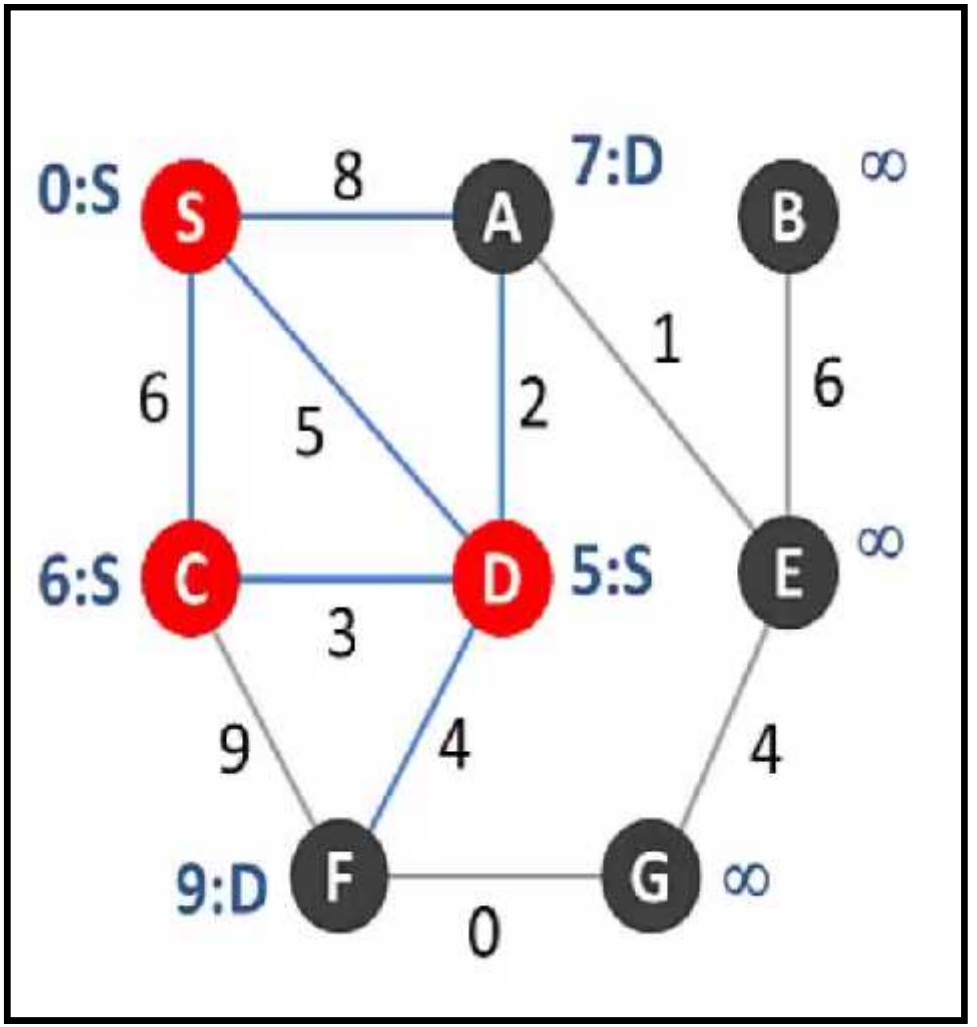


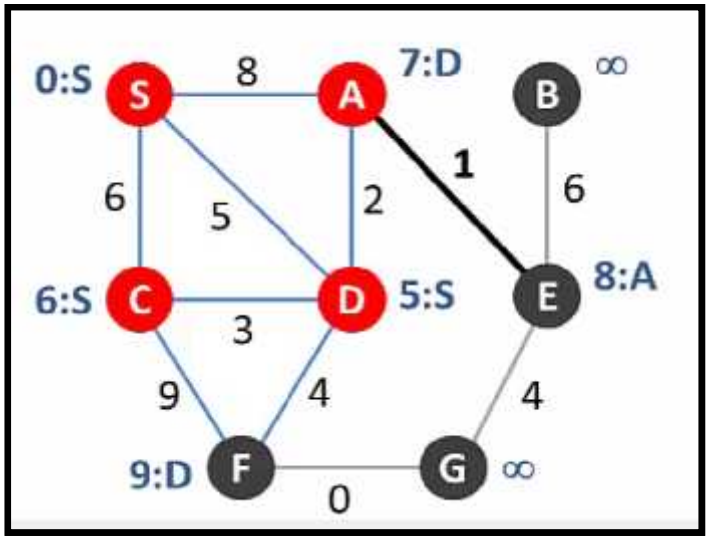


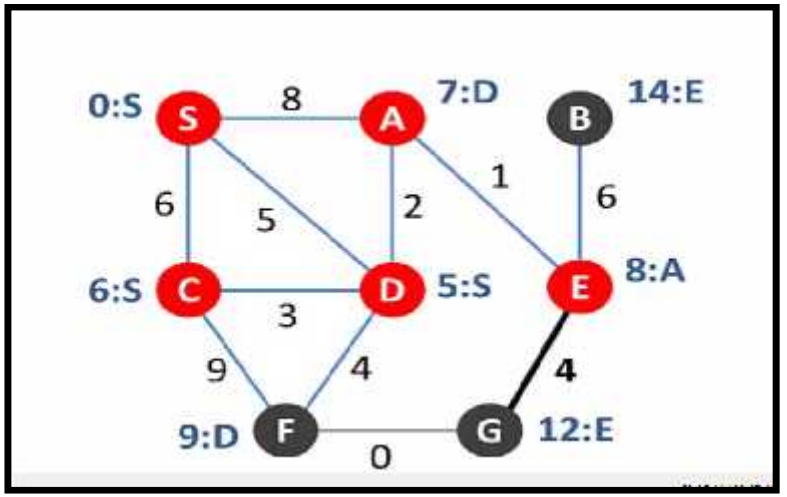

0:S
S
8
A
8:S
B
∞
6
5
2
1
6
6:S
C
D
5:S
E
∞
3
9
4
4
∞
F
G
∞
0

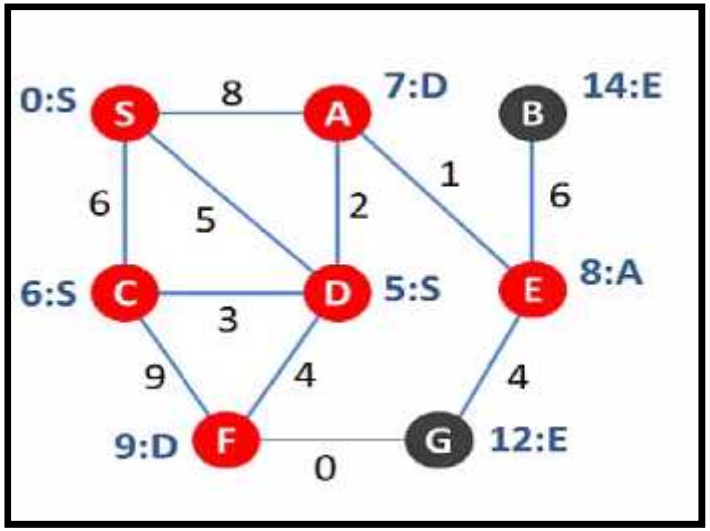

0:S
S
8
A
8:S
B
∞
6
5
2
1
6
6:S
C
D
5:S
E
∞
3
9
4
4
∞
F
G
∞
0

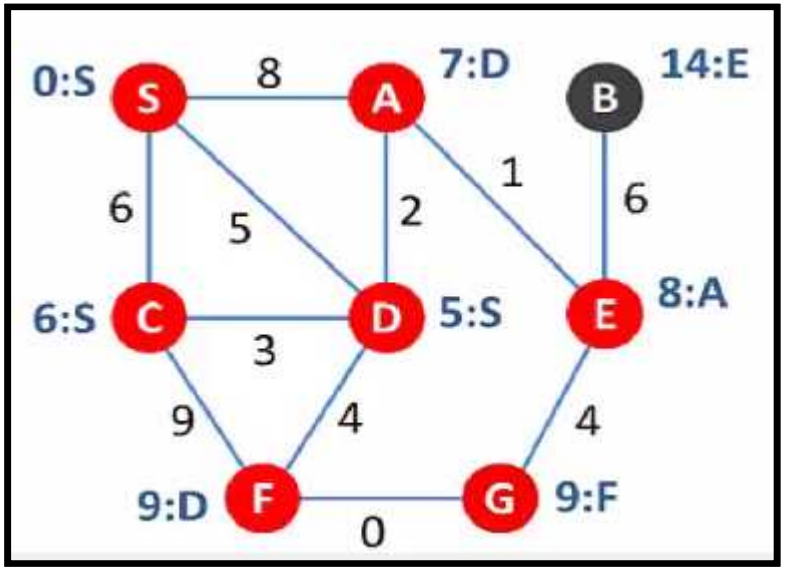

0:S
S
8
A
7:D
B
∞
6
5
2
1
6
6:S
C
D
5:S
E
∞
3
9
4
4
9:D
F
G
∞
0


0:S
S
8
A
7:D
B
∞
6
5
2
1
6
6:S
C
D
5:S
E
∞
3
9
4
4
9:D
F
G
∞
0


0:S
S
8
A
7:D
B
∞
1
6
6
5
2
6:S
C
D
5:S
E
8:A
3
9
4
4
9:D
F
0
G
∞


0:S
S
8
A
7:D
B
14:E
1
6
6
5
2
6:S
C
D
5:S
E
8:A
3
9
4
4
9:D
F
0
G
12:E


0:S
S
8
A
7:D
B
14:E
1
6
6
5
2
6:S
C
D
5:S
E
8:A
3
9
4
4
9:D
F
0
G
12:E


0:S
S
8
A
7:D
B
14:E
1
6
6
5
2
6:S
C
D
5:S
E
8:A
3
9
4
4
9:D
F
0
G
9:F

Single Source Shortest Path: Algorithm

```
Algorithm SSSP(cost, n, v, s, parent: dist)
{Step 1:
  for  i = 1 to n do
  {      s[i]=0;
         dist[i]=cost[v,i];
         if (cost[v,i] ≠ ∞) then
                  parent[i] = v
         else
                  parent[i]= #
  }
         parent [v] = -1;
         s[v] = 1
Step :2
for j = 2 to n do
{
   Find a vertex "u" such that (s[u] ≠ 1 and dist[u] = min)
         s[u] = true
    for (each adjacent vertex "w" from vertex "u") and (s[w]=0) do
         if(dist[w] > dist[u] + cost[u,w]) then
         {
                  dist[w] = dist[u] + cost[u,w]
                  parent[w]=u
         }
}//End of step 2
}//End of algorithm
```