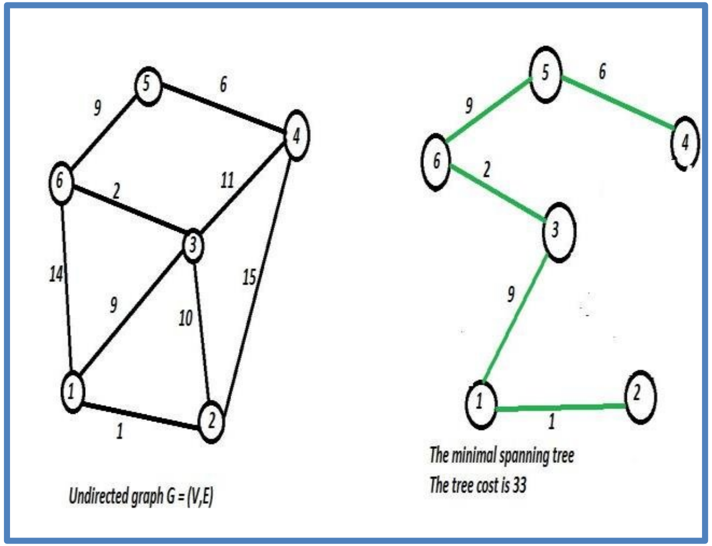
MINIMUM SPANNING TREES



Aritra Hazra & Partha P Chakrabarti

Indian Institute of Technology Kharagpur

Spanning Tree of an Undirected Graph

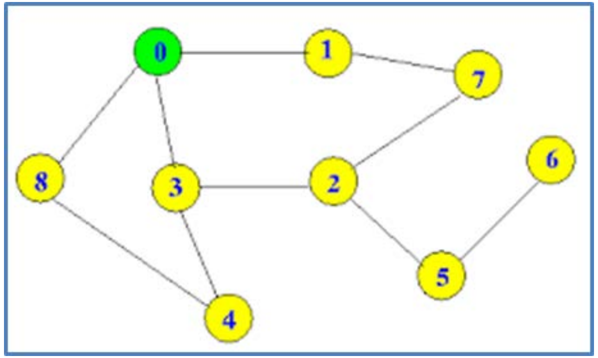
Given a connected Undirected Graph G = (V, E), a Spanning Tree is a connected sub-Tree of G covering every node of G.

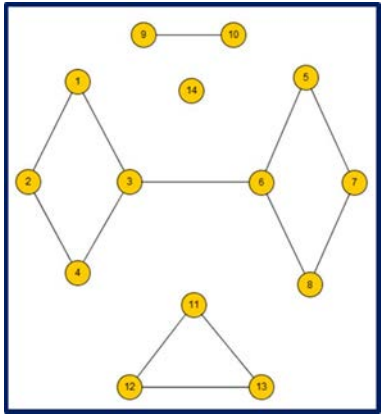
Each Spanning Tree T=(V, E') consists of all the V vertices of G and E' is a subset of E such that G remains connected by the edges of T. Thus E' will have |V| - 1 edges.

Tree Edges of a basic DFS or BFS Traversal will generate a Spanning Tree.

A Graph G may have many Spanning Trees.

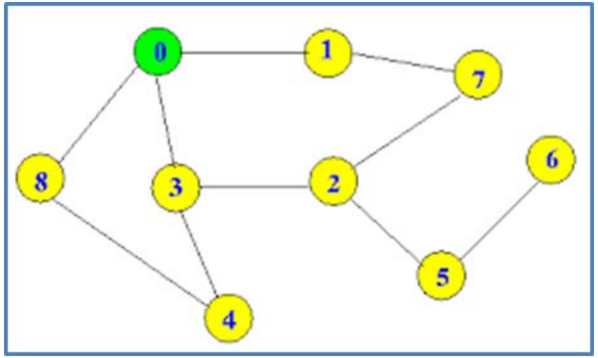
If G = (V,E) is a graph of multiple separate connected components, then we have a Spanning Tree for every connected component and together it is called a Spanning Forest.





DFS based Spanning Tree Algorithm

```
Global Data: G = (V,E)
visited [i] indicates if node i is visited. / initially 0 /
Parent[i] = parent of a node in the Search / initially
NULL /
Tree[i,j] indicates if the edge is a tree edge or not
/initially all 0/
succ(i) = {set of nodes to which node i is
connected}
Dfs(node) {
    visited[node] = 1;
    for each j in succ(node) do {
        if (visited [j] ==0) { Parent[j] = node;
                               Tree[node,j] = 1;
                                Dfs(j) }
     }
  }
```



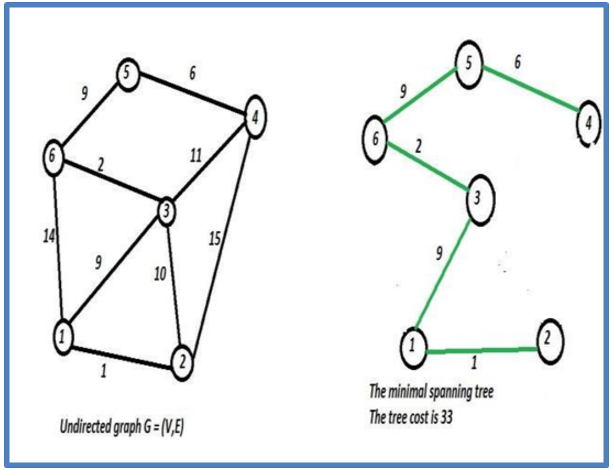
Minimum Spanning Tree of a Weighted Undirected Graph

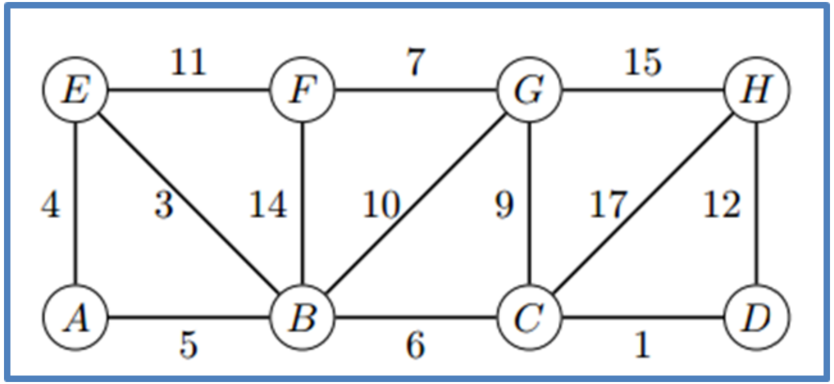
Given a connected Weighted Undirected Graph G = (V, E), a Minimum Spanning Tree (MST) is a Spanning Tree of G of Minimum Cost.

There may be multiple MSTs in a graph. However, if each edge of G has a distinct weight, then the MST is UNIQUE

Applications:

- Design of various kinds of Networks (Circuits, Telecom, Transport, Water Supply, Power Grids, etc)
- Geometric / Vision / Image Processing Problems and Analysis
- Approximations of Complex Problems





Algorithm for MST: Step 1 (Initial Recursive Definition)

MST(G1, G2,T) {

If (G2 =={}) return <T,0>

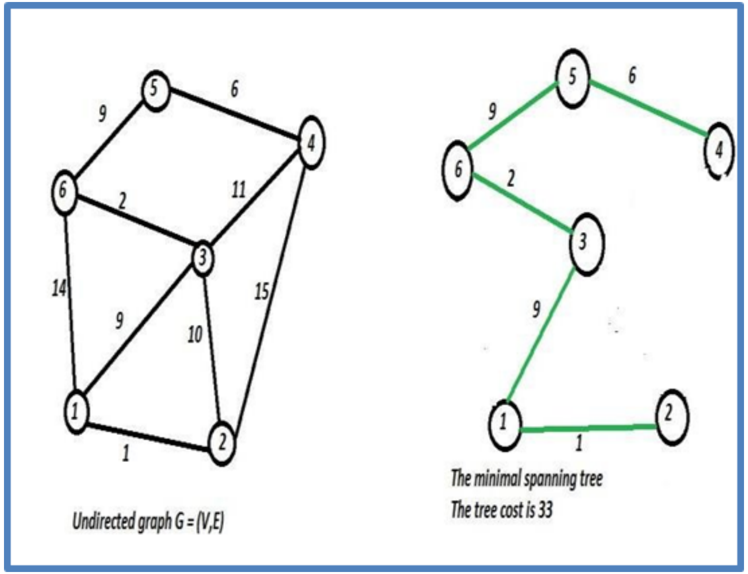
For each edge e = (n,m) from G1 to G2, recursively find the minimum cost Tree (cost_a) using the remaining part of G2 and the corresponding T’ on selection of the edge e as a part of the MST

{ G1a = G1 + {m};
G2a = G2 – {m}
Ta = T + {e}
<T’, cost’> = MST(G1a, G2a,Ta)
cost_a = C[n,m] + cost’
}

Let <T_min, Cost_min> be the minimum cost_a and corresponding T’ found across all edges e

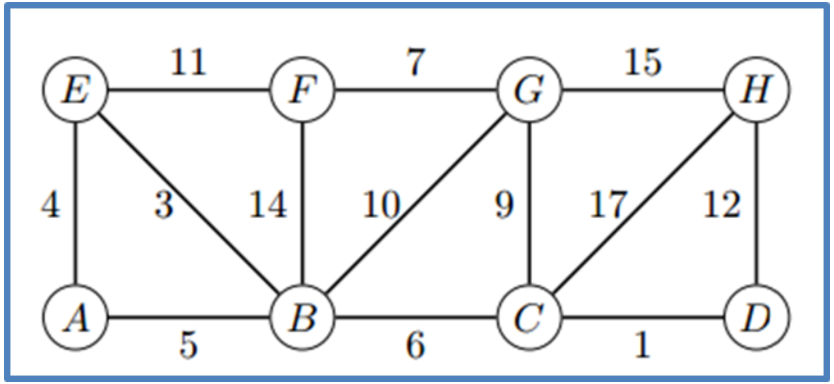
Return <T_min, Cost_Min>

}



Algorithm for MST: Step 1A (Initial Recursive Definition)

```
MST(G1, G2,T) {
If (G2 =={}) return <T,0>
For each edge e = (n,m) from G1 to G2, recursively
find the minimum cost Tree (cost_a) using the
remaining part of G2 and the corresponding T' on
selection of the edge e as a part of the MST
  {   G1a = G1 + {m};
      G2a = G2 – {m}
      Ta = T + {e}
      <T', cost'> = MST(G1a, G2a,Ta)
       cost_a =  C[n,m] + cost'
  }
Let <T_min, Cost_min> be the minimum cost_a and
corresponding T' found across all edges e
Return <T_min, Cost_Min>
}
```



Algorithm for MST: Step 1B (Initial Recursive Definition)

MST(G1, G2,T) {

If (G2 =={}) return <T,0>

For each edge e = (n,m) from G1 to G2, recursively find the minimum cost Tree (cost_a) using the remaining part of G2 and the corresponding T' on selection of the edge e as a part of the MST

{ G1a = G1 + {m};

G2a = G2 – {m}

Ta = T + {e}

<T', cost'> = MST(G1a, G2a,Ta)

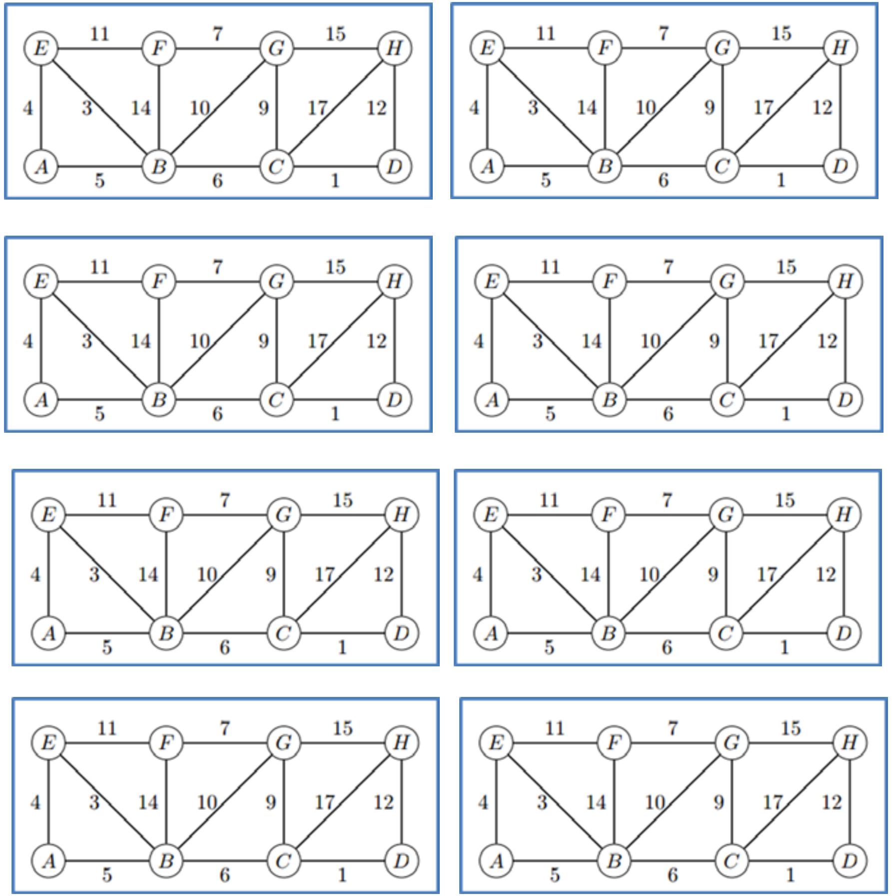
cost_a = C[n,m] + cost'

}

Let <T_min, Cost_min> be the minimum cost_a and corresponding T' found across all edges e

Return <T_min, Cost_Min>

}



Algorithm for MST: Step 2 (Analyzing the Properties)

MST(G1, G2,T) {

If (G2 =={}) return <T,0>

For each edge e = (n,m) from G1 to G2, recursively find the minimum cost Tree (cost_a) using the remaining part of G2 and the corresponding T’ on selection of the edge e as a part of the MST

{ G1a = G1 + {m};

G2a = G2 – {m}

Ta = T + {e}

<T’, cost’> = MST(G1a, G2a,Ta)

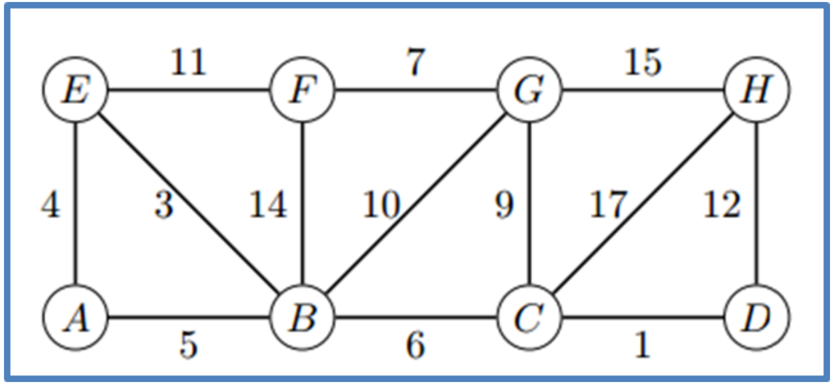
cost_a = C[n,m] + cost’

}

Let <T_min, Cost_min> be the minimum cost_a and corresponding T’ found across all edges e

Return <T_min, Cost_Min>

}



Algorithm for MST: Step 3 (Making a Greedy Choice)

MST_Greedy (G1, G2,T) {

If (G2 =={}) return <T,0>

From all edges e = (n,m) from G1 to G2 do and find the minimum cost edge e' = (n',m') from G1 to G2 and do the following:

{ G1a = G1 + {m'};

G2a = G2 – {m'}

Ta = T + {e'}

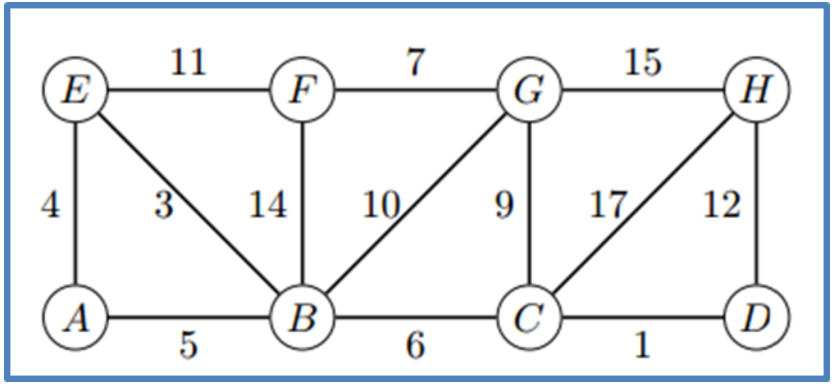
<T',cost'> = MST_Greedy (G1a, G2a,Ta)

cost_a = C[n,m] + cost'

}

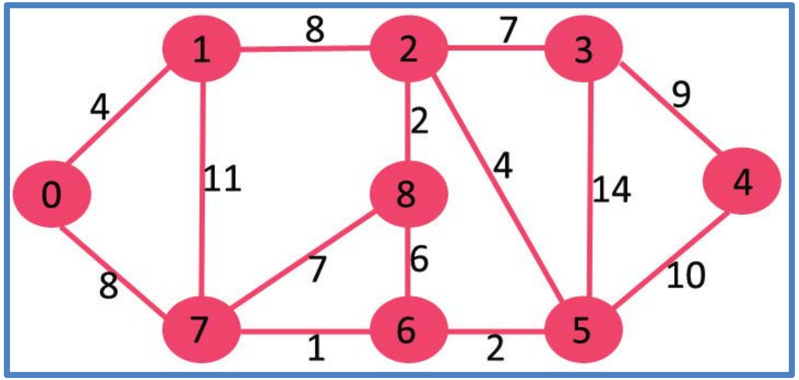
Return <T', cost_a>

}



Algorithm for MST: Step 3 (Example)

```
MST_Greedy (G1, G2,T) {
If (G2 =={}) return <T,0>
From all edges e = (n,m) from G1 to G2 do and
find the minimum cost edge e' = (n',m') from
G1 to G2 and do the following:
  {   G1a = G1 + {m'};
      G2a = G2 – {m'}
      Ta = T + {e'}
      <T',cost'> = MST_Greedy (G1a, G2a,Ta)
       cost_a =  C[n,m] + cost'
   }
Return <T', cost_a>
}
```



Algorithm for MST: Step 3 (Analysis)

MST_Greedy (G1, G2,T) {

If (G2 =={}) return <T,0>

From all edges e = (n,m) from G1 to G2 do and find the minimum cost edge e' = (n',m') from G1 to G2 and do the following:

{ G1a = G1 + {m'};

G2a = G2 – {m'}

Ta = T + {e'}

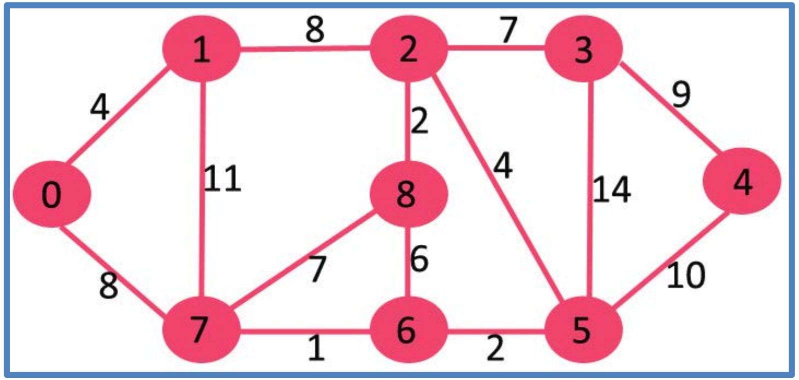
<T',cost'> = MST_Greedy (G1a, G2a,Ta)

cost_a = C[n,m] + cost'

}

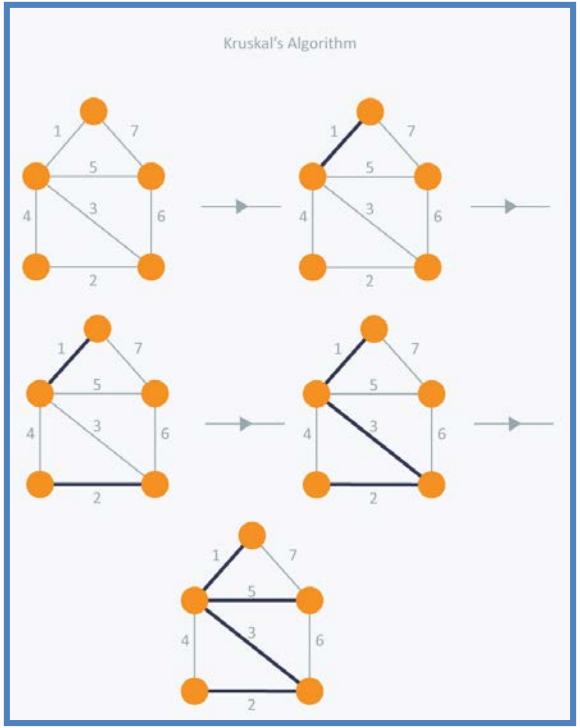
Return <T', cost_a>

}



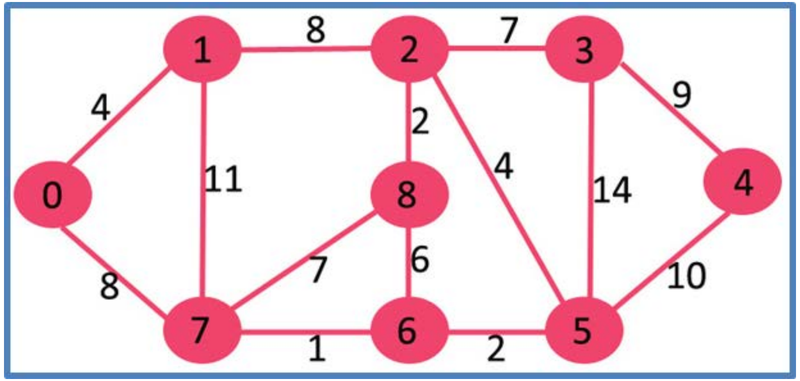
Kruskal’s Algorithm for MST

```
Kruskal (V,E) {
Sort the edges in E in increasing cost;
VT = {}; ET = {}; cost = 0;
While E is not empty or VT = V do {
Choose the next minimum cost edge e =
(n,m) in E;
E = E – {e}
If adding n, m in VT and e in ET makes a
cycle,  discard e, else {
        VT = VT + {n} + {m}
        ET = ET + { e}
        cost = cost + C[n,m] }
   }
Return <GT = (VT, ET), cost>
}
```



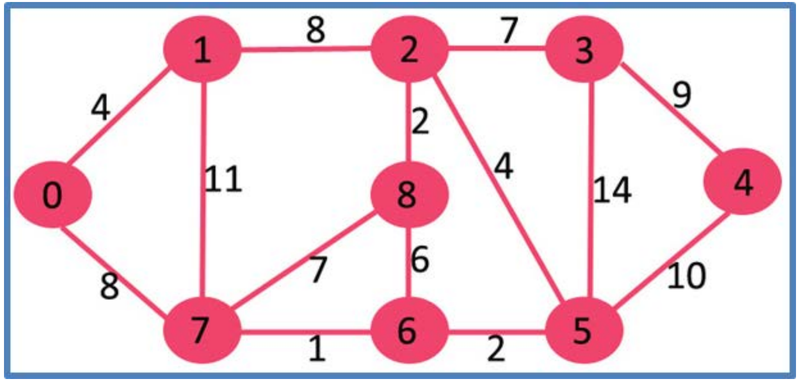
Kruskal’s Algorithm for MST (Example)

```
Kruskal (V,E) {
Sort the edges in E in increasing cost;
VT = {}; ET = {}; cost = 0;
While E is not empty or VT = V do {
Choose the next minimum cost edge e =
(n,m) in E;
E = E – {e}
If adding n, m in VT and e in ET makes a
cycle,  discard e, else {
        VT = VT + {n} + {m}
        ET = ET + { e}
        cost = cost + C[n,m] }
   }
Return <GT = (VT, ET), cost>
}
```



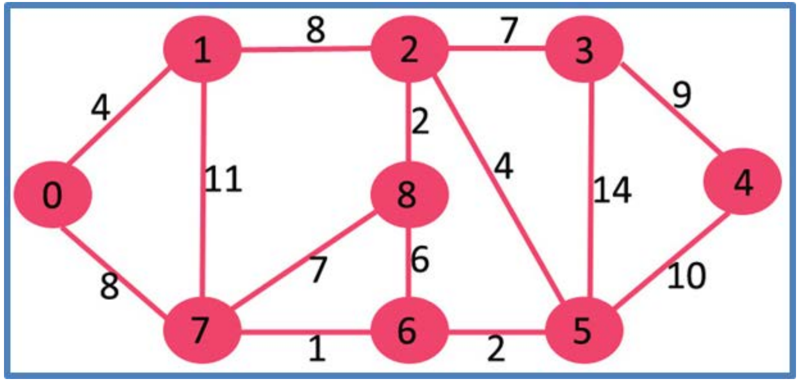
Kruskal's Algorithm for MST (Analysis)

Kruskal (V,E) {
Sort the edges in E in increasing cost;
VT = {}; ET = {}; cost = 0;
While E is not empty or VT = V do {
Choose the next minimum cost edge e = (n,m) in E;
E = E – {e}
If adding n, m in VT and e in ET makes a cycle, discard e, else {
    VT = VT + {n} + {m}
    ET = ET + { e}
    cost = cost + C[n,m] }
  }
Return <GT = (VT, ET), cost>
}



Kruskal's Algorithm (using Disjoint UNION-FIND)

```
algorithm Kruskal_UF (G = (V,E)) {
A = {};
for each v in V do MAKE-SET(v);
for each edge e = (u, v) in E ordered by
increasing weight(u, v) do
   {
     if FIND-SET(u) ≠ FIND-SET(v) then
     {
       A = A + {(u, v)}
       UNION(FIND-SET(u), FIND-SET(v))
      }
    }
return A
}
```



Thank you