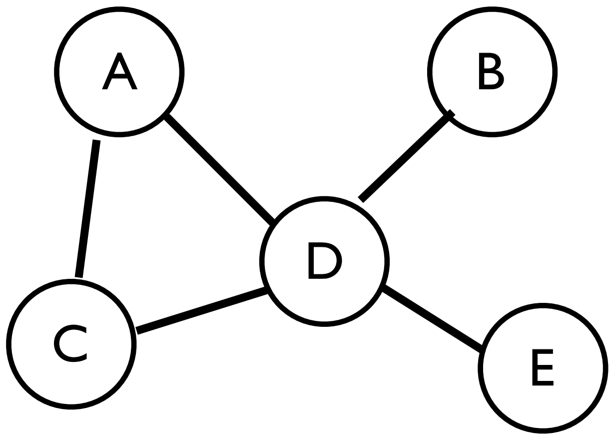
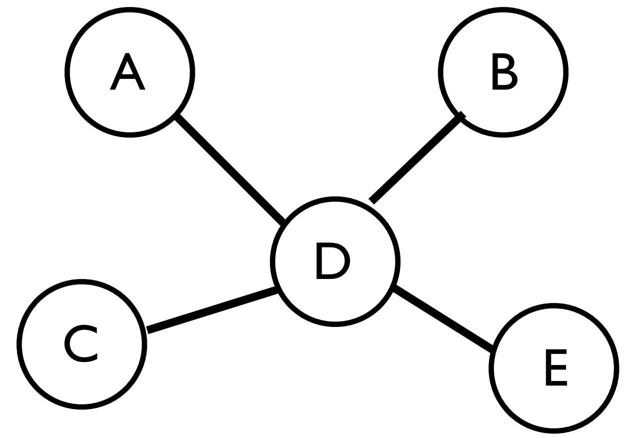
Data Structure:
Graph

Spanning tree

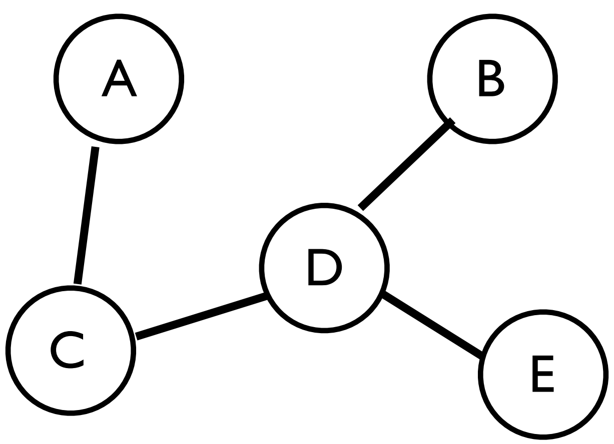
A spanning tree of G is a subgraph of G that is a tree containing every vertex of G



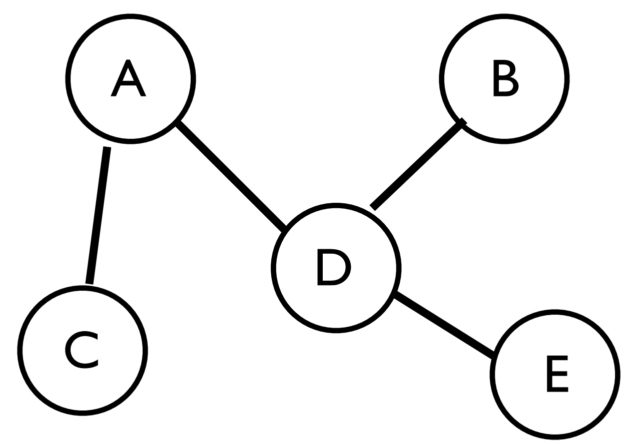
G



spanning tree of G



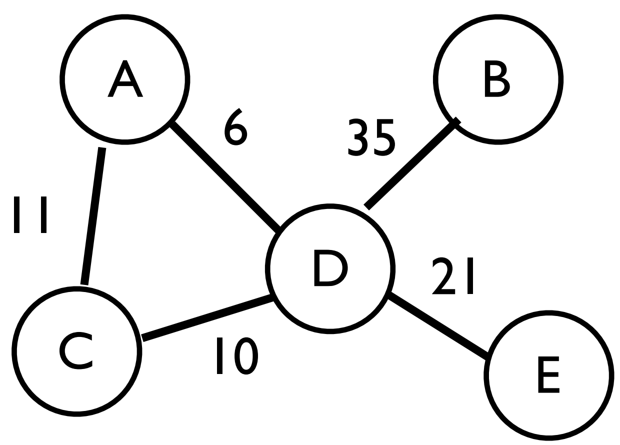
spanning tree of G



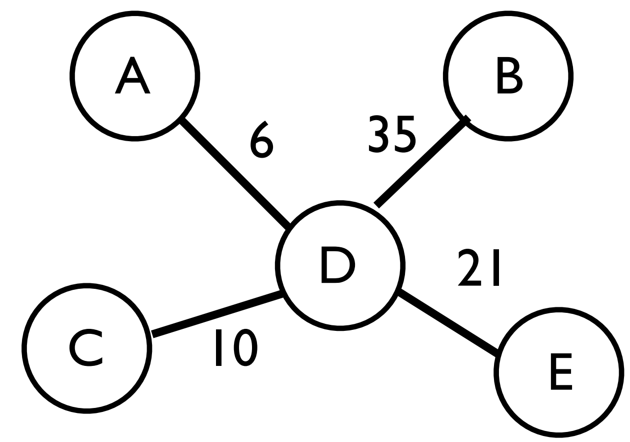
spanning tree of G

Minimum spanning tree (MST)

A minimum spanning tree in a connected weighted graph is a spanning tree that has the smallest possible sum of weights of its edges

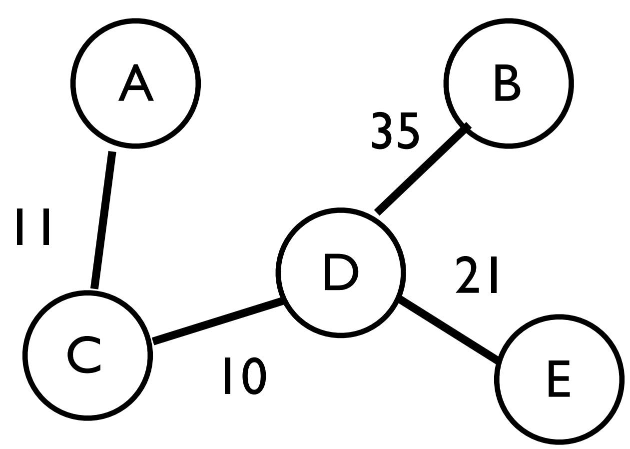


G



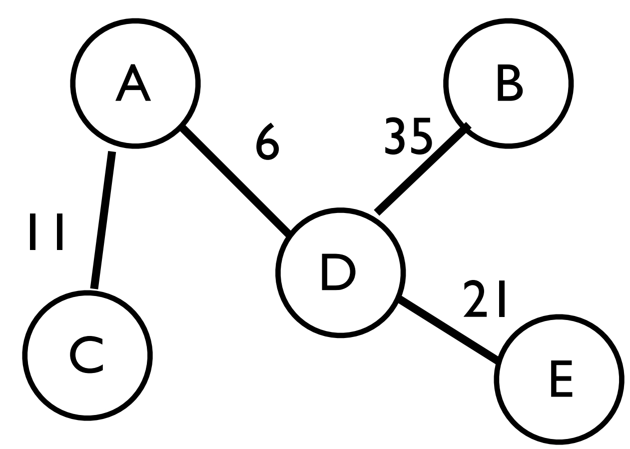
minimum spanning tree of G

weight:72



spanning tree of G

weight:77



spanning tree of G

weight:73

Minimum spanning tree (MST)

- given a connected, undirected graph $G = (V, E)$, a spanning tree is an acyclic subset of edges $T \subseteq E$ that connects all vertices together.
- a common problem in communication networks and circuit design
- the cost of a spanning tree T is $w(T) = \sum_{(u,v)\in T} w(u,v)$
- a minimum spanning tree is the one with minimum cost

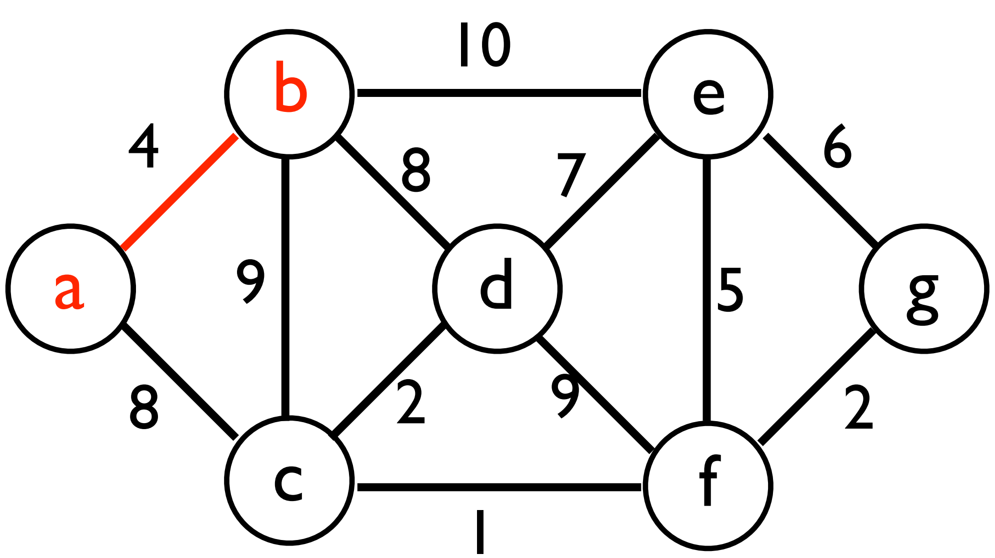
- the idea of finding MST (greedy approach)
  - start with an empty graph
  - add edges (with the smallest cost at each step) one at a time

- several algorithms depending on how to choose edges to add
  - Prim's algorithm
  - Kruskal's algorithm

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u]   /* in Dijkstra's algorithm

  d[v] = min(d[v], w(u, v))   /* d[v] = min(d[v], d[u]+ w(u, v)) */

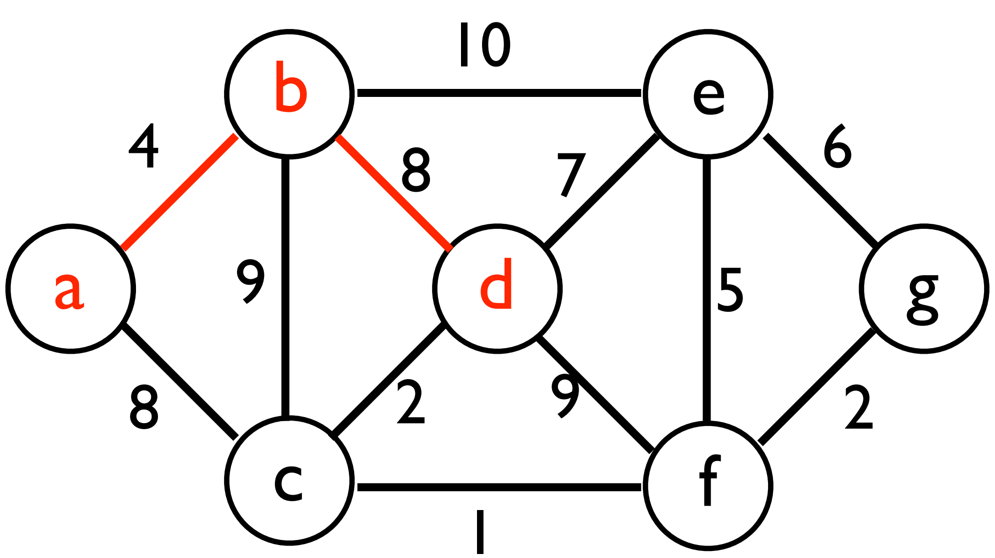


a	0						
b	∞	4 (a)					
c	∞	8 (a)					
d	∞	∞					
e	∞	∞					
f	∞	∞					
g	∞	∞					

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u]   /* in Dijkstra's algorithm

    d[v] = min(d[v], w(u, v))   /* d[v] = min(d[v], d[u]+ w(u, v)) */

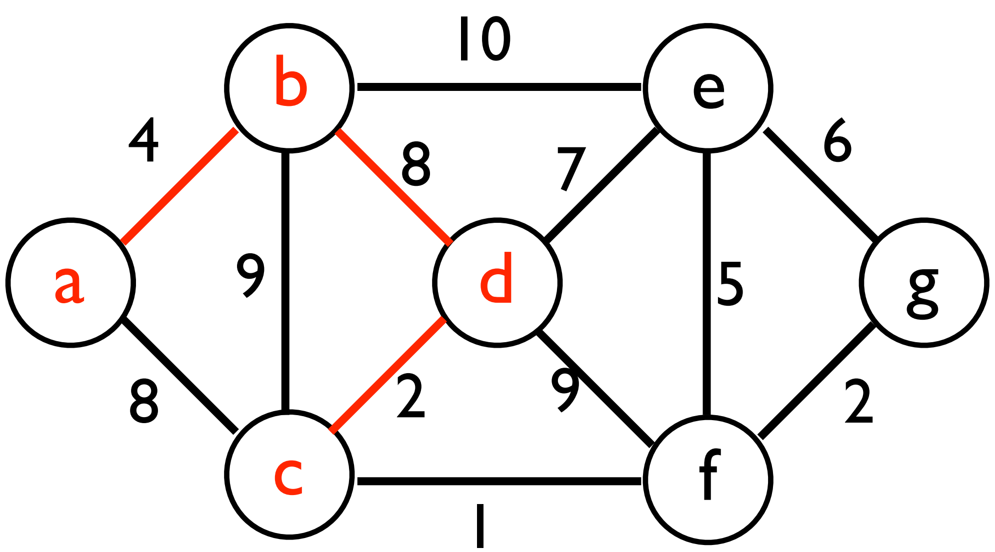


a	0						
b	∞	4 (a)					
c	∞	8 (a)	8 (a)				
d	∞	∞	8(b)				
e	∞	∞	10 (b)				
f	∞	∞	∞				
g	∞	∞	∞				

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u] /* in Dijkstra's algorithm

  d[v] = min(d[v], w(u, v)) /* d[v] = min(d[v], d[u]+ w(u, v)) */

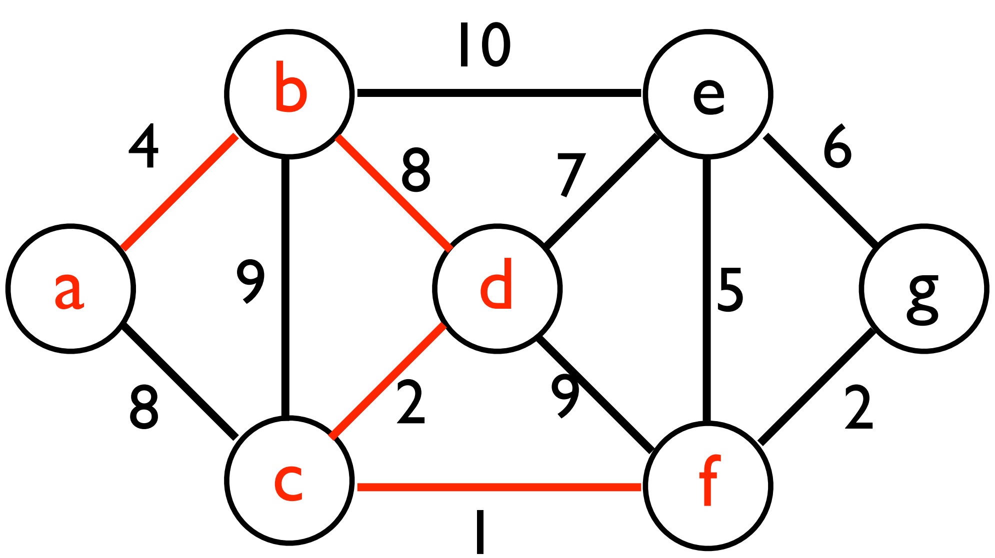


a	0						
b	∞	4 (a)					
c	∞	8 (a)	8 (a)	2(d)			
d	∞	∞	8(b)				
e	∞	∞	10 (b)	7(d)			
f	∞	∞	∞	9(d)			
g	∞	∞	∞	∞			

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u] /* in Dijkstra's algorithm

  d[v] = min(d[v], w(u, v)) /* d[v] = min(d[v], d[u]+ w(u, v)) */

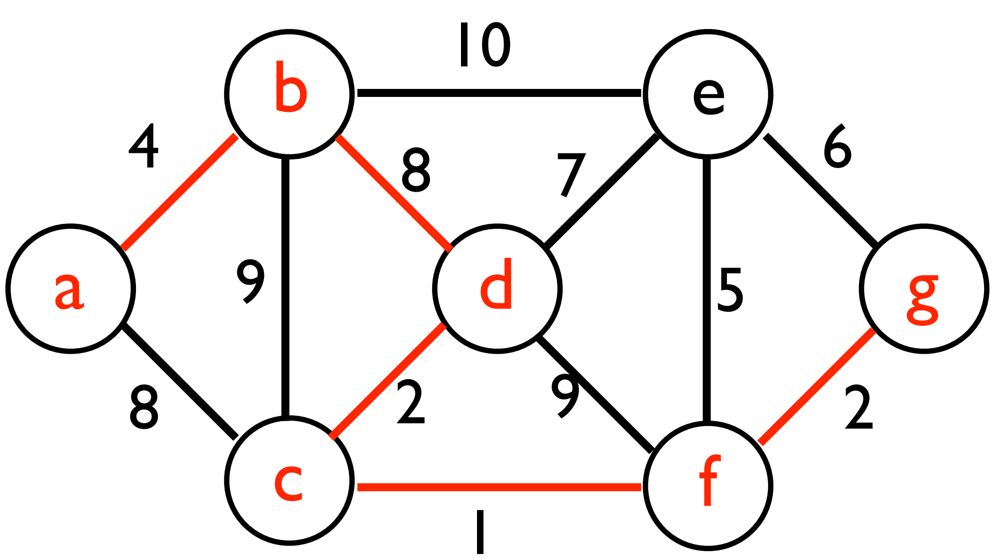


a	0						
b	∞	4 (a)					
c	∞	8 (a)	8 (a)	2(d)			
d	∞	∞	8(b)				
e	∞	∞	10 (b)	7(d)	7(d)		
f	∞	∞	∞	9(d)	1(c)		
g	∞	∞	∞	∞	∞		

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u]   /* in Dijkstra's algorithm

    d[v] = min(d[v], w(u, v))   /* d[v] = min(d[v], d[u]+ w(u, v)) */

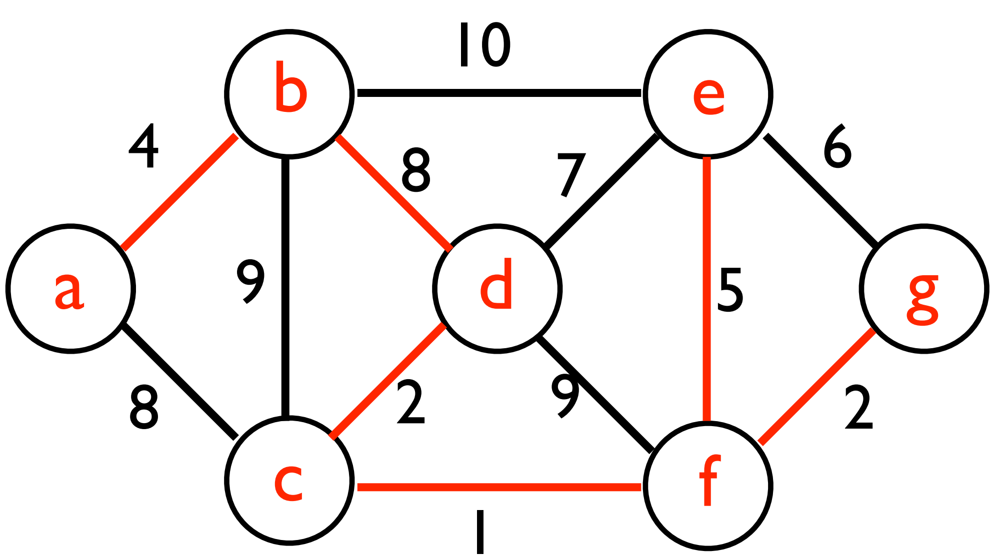


a	0						
b	∞	4 (a)					
c	∞	8 (a)	8 (a)	2(d)			
d	∞	∞	8(b)				
e	∞	∞	10 (b)	7(d)	7(d)	5(f)	
f	∞	∞	∞	9(d)	1(c)		
g	∞	∞	∞	∞	∞	2(f)	

Minimum spanning tree: Prim's algorithm

- similar to Dijkstra's algorithm (finding the shortest path)
- for each v in Adj[u] /* in Dijkstra's algorithm

  d[v] = min(d[v], w(u, v)) /* d[v] = min(d[v], d[u]+ w(u, v)) */

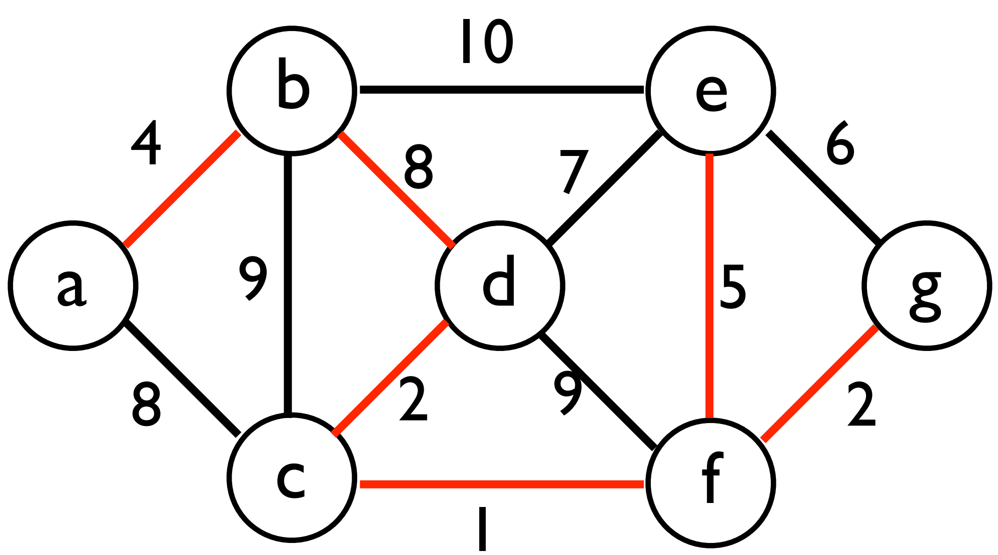


a	0						
b	∞	4 (a)					
c	∞	8 (a)	8 (a)	2(d)			
d	∞	∞	8(b)				
e	∞	∞	10 (b)	7(d)	7(d)	5(f)	5(f)
f	∞	∞	∞	9(d)	1(c)		
g	∞	∞	∞	∞	∞	2(f)	

Minimum spanning tree: Prim's algorithm

```
Prim (G, w, r)  {
    for each u in V    {
        key[u] = infinite;    color[u] = W;
    }
    key[r] = 0;
    pred[r] = NIL;
    Q = MakePriorityQueue(V);
    While( Q  is nonempty)  {
        u = deleteMin(Q);
        for each (v is adjacent u){
            if ( (color[v] == W  && w[u,v] < key[v]){
                key[v] = w[u, v];
                pred[v] = u;
                Decrease_Priority(Q, v);
            }
        }
        color[u] = B;
    }
}
```

Minimum spanning tree: Kruskal's algorithm



cf	1	o
cd	2	o
fg	2	o
ab	4	o
ef	5	o
eg	6	
de	7	
bd	8	o
ac	8	
df	9	
bc	9	
be	10	

Minimum spanning tree: Kruskal's algorithm

```
Kruskal (G = (V, E))
{
  MST = {};
  for each v in V
     Create_Set({v});                                          O(n)

  Sort the edges of E in increasing order of weights;          O(e log e)


  for each edge (u, v) in E in weight order do                 O(e log n)
  {
    if ( Find(u) != Find(v) ) THEN
    {
      MST = MST + {(u, v)};
      Union(Find(u), Find(v));
    }
  }
}
```

- since G is connected, we have n-1 $\leq e \leq n^2$, thus (log e) = Θ(log n)
- the total running time is
  T(n, e) = O(n) + O(e log e) + O(e log n) = O(e log n)