Kruskal Algorithm

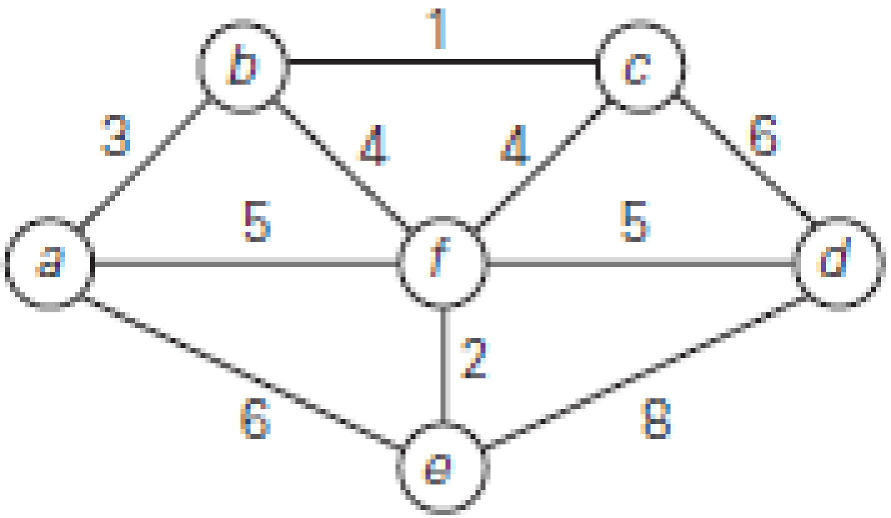
Algorithm

- Joseph Kruskal
- second-year graduate student
- The algorithm begins by sorting the graph’s edges in nondecreasing order of their weights. Then, starting with the empty subgraph, it scans this sorted list, adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

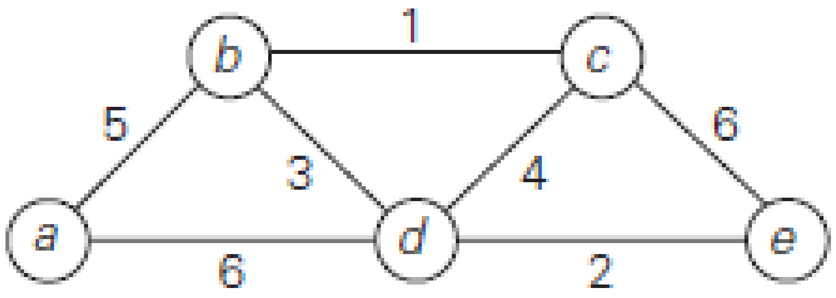
ALGORITHM *Kruskal(G)*

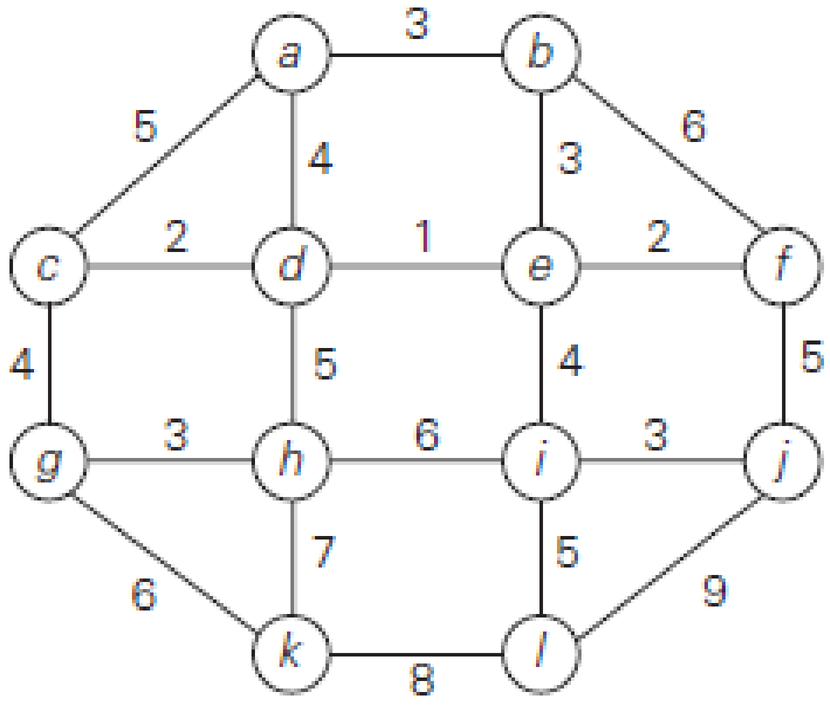
```
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = ⟨V, E⟩
//Output: E_T, the set of edges composing a minimum spanning tree of G
sort E in nondecreasing order of the edge weights w(e_{i_1}) ≤ ... ≤ w(e_{i_|E|})
E_T ← ∅;   ecounter ← 0       //initialize the set of tree edges and its size
k ← 0                         //initialize the number of processed edges
while ecounter < |V| − 1 do
    k ← k + 1
    if E_T ∪ {e_{i_k}} is acyclic
        E_T ← E_T ∪ {e_{i_k}};   ecounter ← ecounter + 1
return E_T
```

- **Algorithm** *Kruskal*(G)
- sort E by the edge weights // Note this is a Priority Queue
- make disjoint sets of vertices, $d(v)$
- **while** $ecounter < |V|-1$ **and** E is not empty **do**
- $(u, v) \leftarrow$ remove minimum from E
- if $find(u) \neq find(v)$ then
- $T \leftarrow union(u, v)$
- $ecounter$++
- **return** T



bc	ef	ab	bf	cf	af	df	ae	cd	de
1	2	3	4	4	5	5	6	6	8


1
b
c
5
6
3
4
a
d
e
6
2


a
b
c
d
e
f
g
h
i
j
k
l
3
5
4
3
6
2
1
2
4
5
4
5
3
6
3
6
7
5
9
8