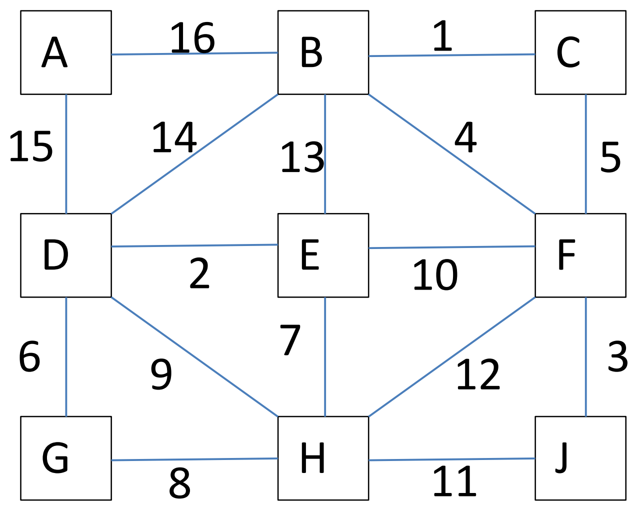
MST

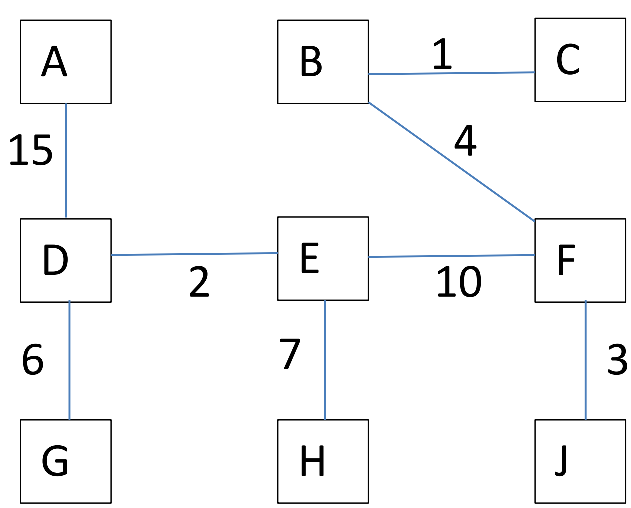
Minimum Spanning Tree:

Given a connected weighted undirected graph, find a minimum-weight set of edges that forms a subtree that connects all the vertices of the graph

Example:



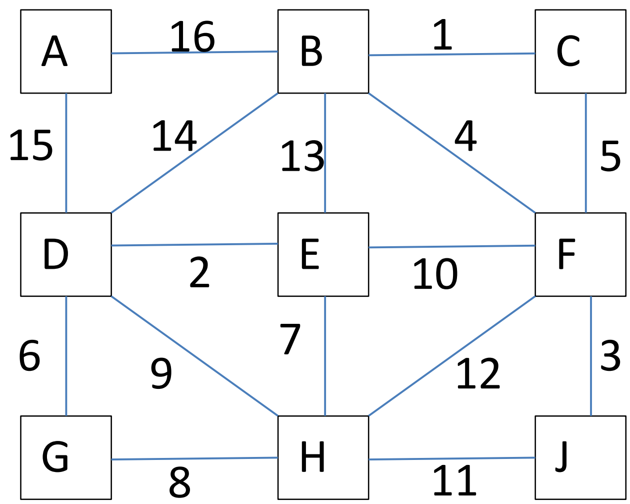
MST:



Prim's algorithm for finding a MST of undirected graph G:

```
choose a start vertex;
H = new MinHeap( );
for each vertex v {
      if (v==start)  cost[v] = 0;
      else  cost[v] = ∞;
      H.insert (v, cost[v]);           // cost[v] is the key
}
while (! H.isEmpty( )) {
      x = H.removeMin( );
      for each vertex y such that (x,y) is an edge in graph G
                  if (y is in H  and  weight(x,y) < cost[y]) {
                        cost[y] = weight(x,y);
                        parent[y] = x;
                        H.decreaseKey (y, cost[y]);
                              // swap y up the heap as necessary
      }
}
```

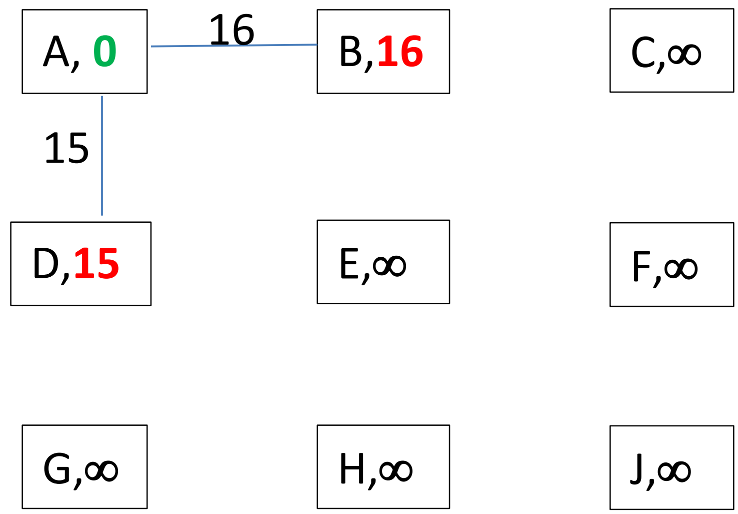
Trace Prim's algorithm:



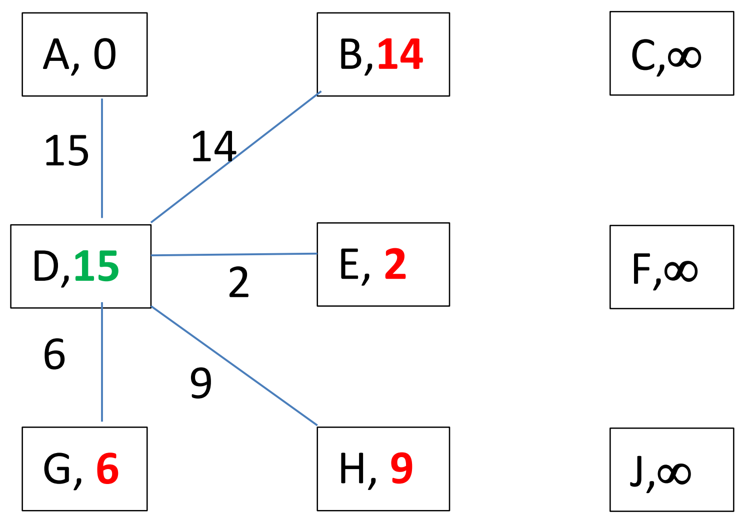
Choose start vertex = A

A, **0**	B,∞	C,∞
D,∞	E,∞	F,∞
G,∞	H,∞	J,∞

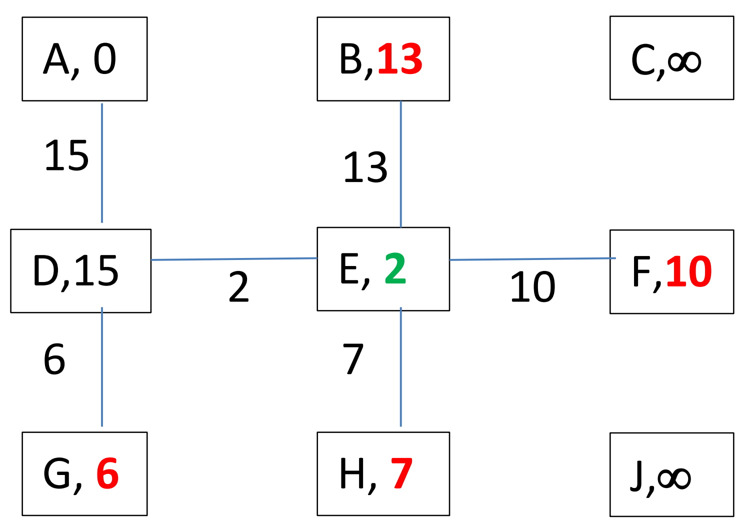
RemoveMin: A



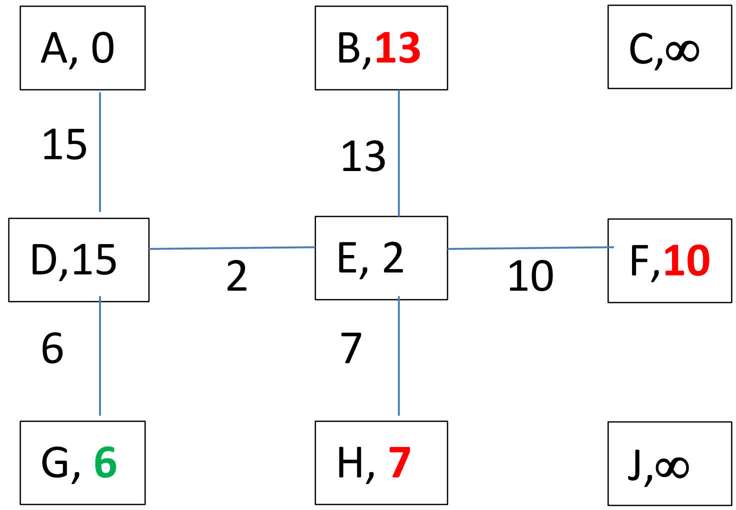
RemoveMin: D



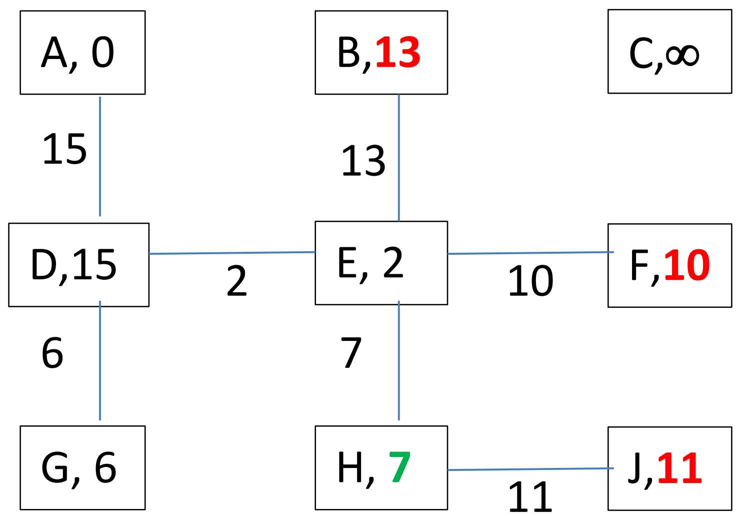
RemoveMin: E



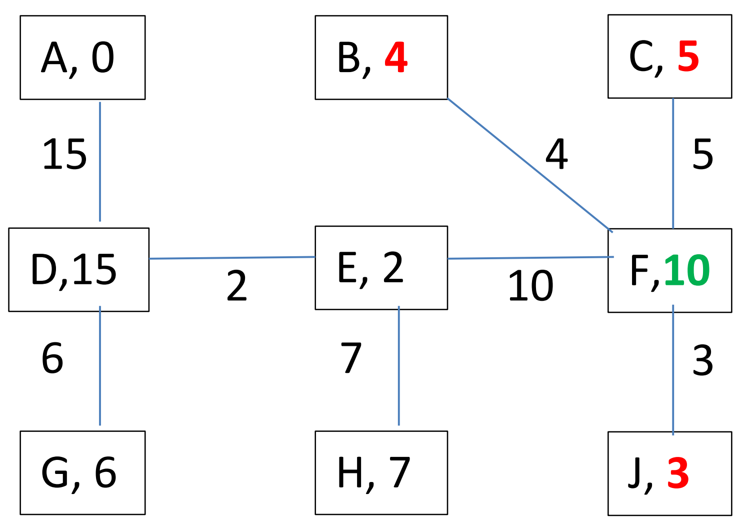
RemoveMin: G



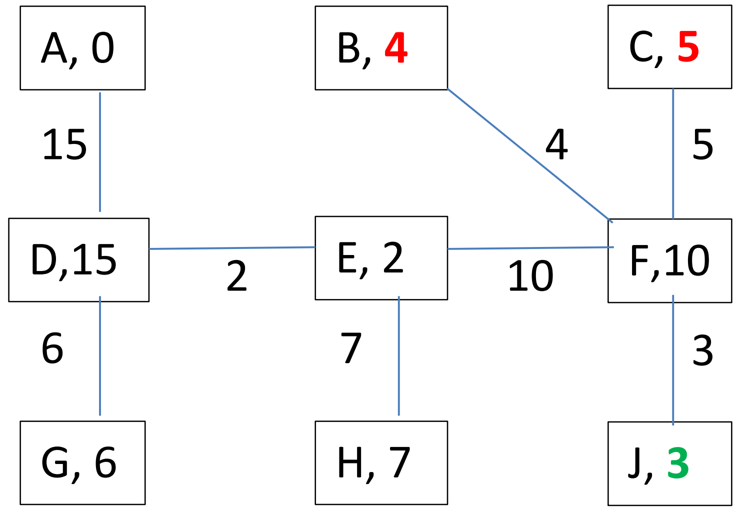
RemoveMin: H



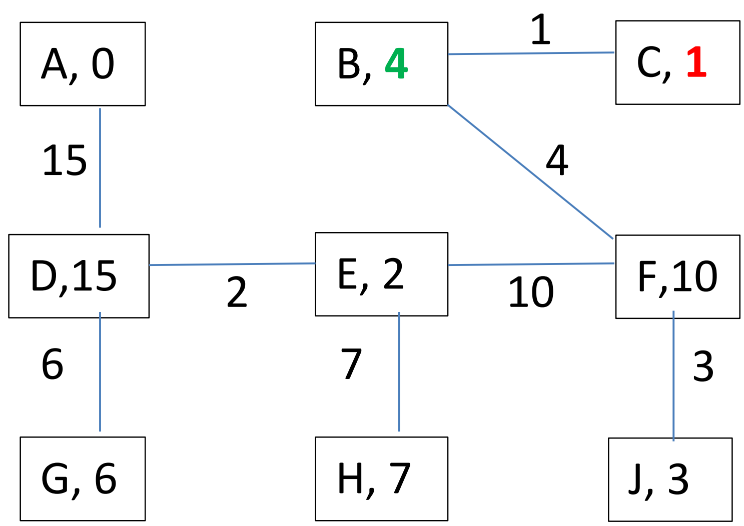
RemoveMin: F



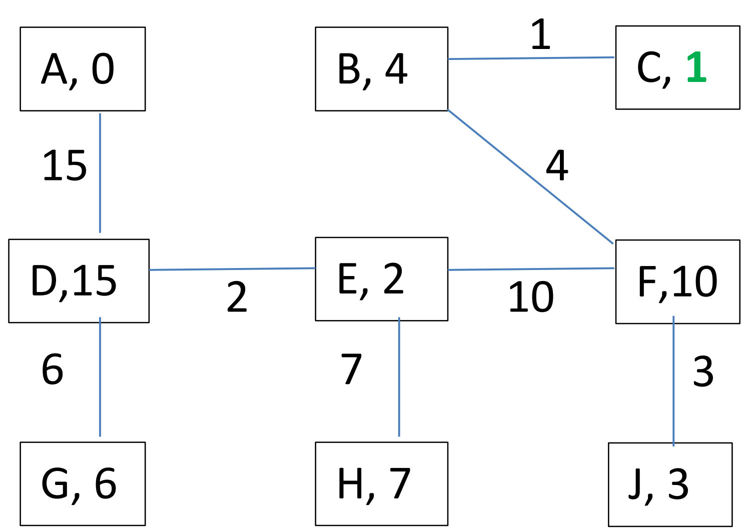
RemoveMin: J



RemoveMin: B



RemoveMin: C



Recall Dijkstra's algorithm for finding a shortest paths tree in (undirected or directed) graph G:

```
choose a start vertex;
H = new MinHeap( );
for each vertex v {
      if (v==start)  cost[v] = 0;
      else  cost[v] = ∞;
      H.insert (v, cost[v]);            // cost[v] is the key
}
while (! H.isEmpty( )) {
      x = H.removeMin( );
      for each vertex y such that (x,y) is an edge in graph G
                  if (cost[x] + weight(x,y) < cost[y]) {
                        cost[y] = cost[x] + weight(x,y);
                        parent[y] = x;
                        H.decreaseKey (y, cost[y]);
                              // swap y up the heap as necessary
      }
}
```

Note: algorithm is same as Prim's algorithm except for "**cost[x] +**"

Implementation and analysis of Prim's algorithm:

Same as for Dijkstra's algorithm

(i) Use adjacency lists representation for graph G

- Time to find all edges (x,y) in G is $\theta(n+m)$ time

Use binary heap H

- n inserts, each takes $\theta(\lg n)$ time
- n removeMins, each takes $\theta(\lg n)$ time
- ≤ m decreaseKeys, each takes $\theta(\lg n)$ time
- Time for all heap operations is $\theta((2n+m) \lg n)$ time

Total time for Prim's algorithm using method (i) is $\theta(m \lg n)$

(ii) Use adjacency matrix representation for graph G

- Time to find all edges (x,y) in G is $\theta(n^2)$ time

Use boolean array in place of heap H:

array[v] = true if v is currently in the heap

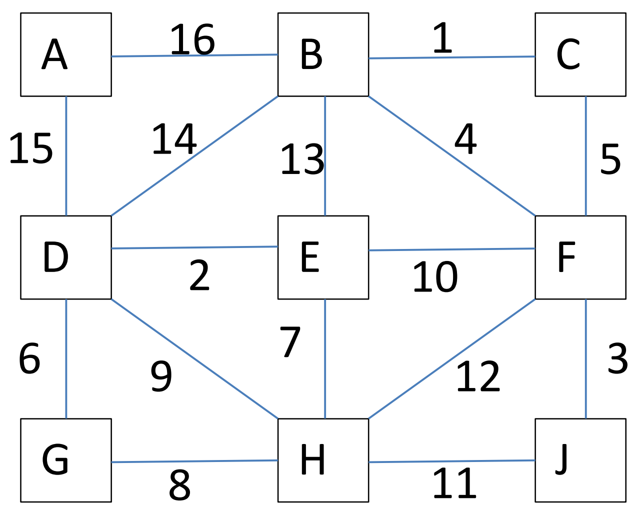
- n inserts, each takes $\theta(1)$ time
- n removeMins, each takes $\theta(n)$ time
- ≤ m decreaseKeys, each takes $\theta(1)$ time
- Time for all heap operations is $\theta(n^2)$ time

Total time for Prim's algorithm using method (ii) is $\theta(n^2)$

Kruskal's algorithm for finding a MST:

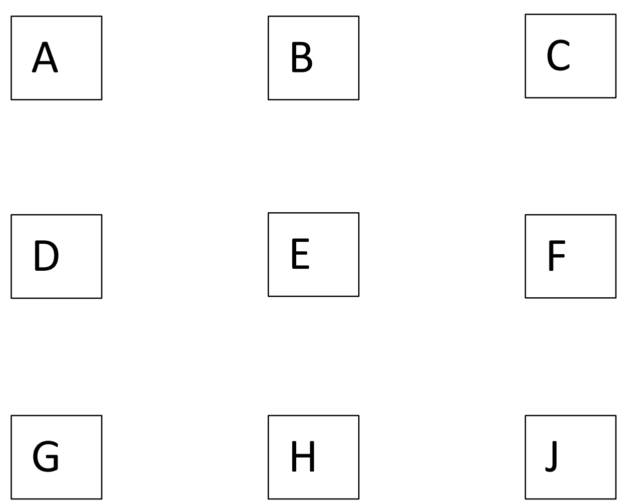
```
initially place each vertex in its own tree;
sort all edges in ascending order by weights;
for each edge (x,y) in this order
    if (x and y are in different trees)
        add edge (x,y);
```

Trace Kruskal's algorithm:



Sort edges: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16

MST:



Efficient implementation and analysis of Kruskal:

First sort edge weights using heap sort or merge sort:

θ(m lg m) time

But $n-1 \le m < n^2$ because graph is connected, undirected

$\lg(n-1) \le \lg m < \lg n^2 = 2 \lg n$, so lg m is θ(lg n)

Therefore θ(m lg m) = θ(m lg n) time

Remainder of the analysis is deferred until our next meeting

Next use disjoint sets to represent the components (trees):

```
for each edge (x,y) in this order {
      a = Find (x);  b = Find (y);
      if (a != b) {
            Merge (a, b);
            add edge (x,y);
      }
}
```

Each Find operation takes O(lg n) time

Each Merge operation takes O(1) time

2m Find operations $\Rightarrow$ O(m lg n) time

n–1 Merge operations $\Rightarrow$ θ(n) time $\Rightarrow$ O(m) time

Total time for disjoint sets is O(m lg n) time

So total time for Kruskal's algorithm is θ(m lg n)