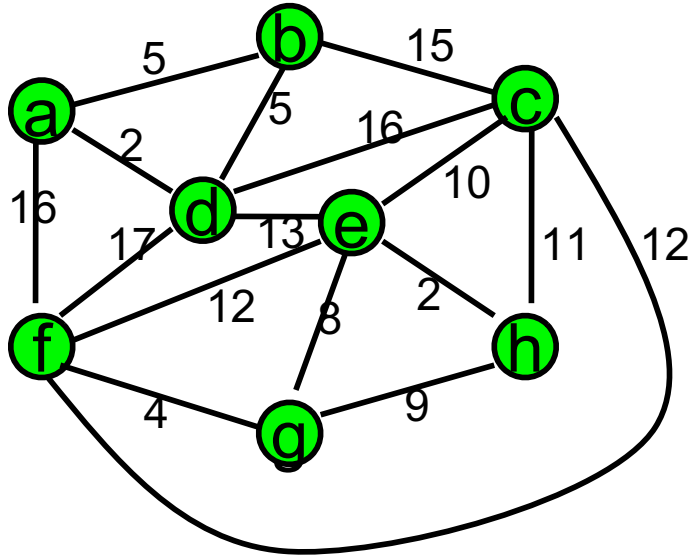
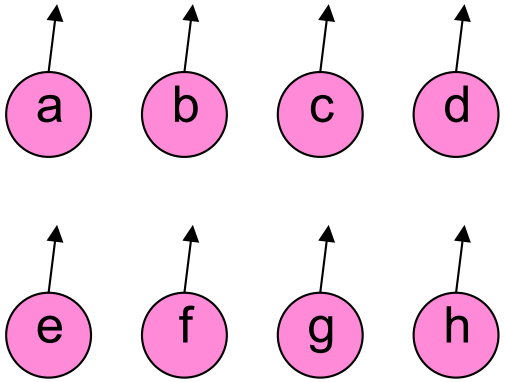
Today's announcements:

MP7 available. Due 5/2, 11:59p.

Kruskal's Algorithm (1956)





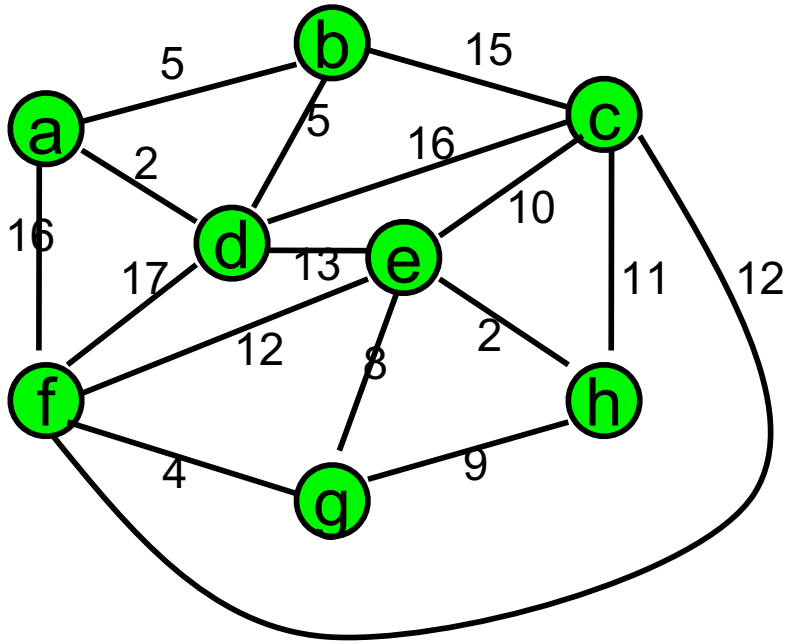
(a,d)
(e,h)
(f,g)
(a,b)
(b,d)
(g,e)
(g,h)
(e,c)
(c,h)
(e,f)
(f,c)
(d,e)
(b,c)
(c,d)
(a,f)
(d,f)

1. Init graph T to be our output. Let it consist of all n vertices and no edges.

2. Init a disjoint sets structure where each vertex is represented by a set.

3. RemoveMin from PQ. If that edge connects 2 vertices from different sets, add the edge to T and take Union of the vertices' two sets, otherwise do nothing. Repeat until _____ edges are added to T.

Kruskal's Algorithm - preanalysis

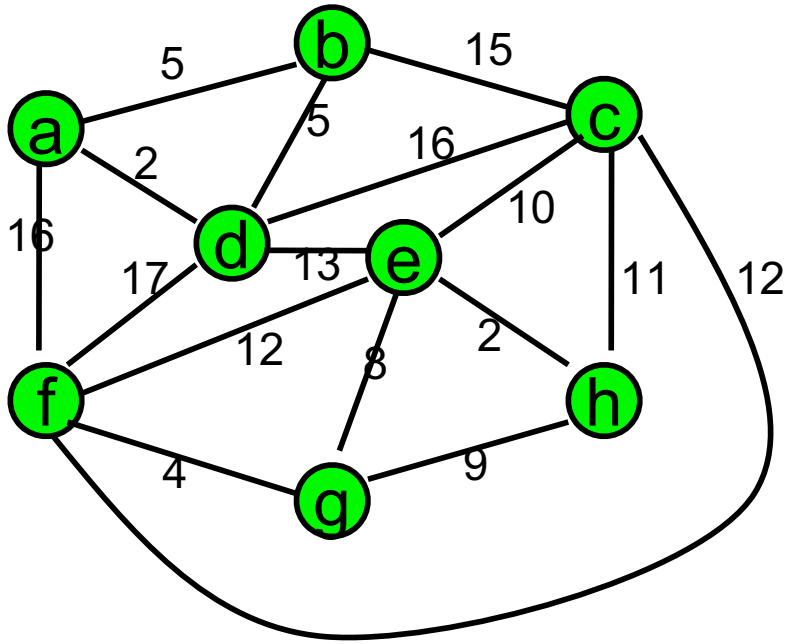


Algorithm *KruskalMST*(*G*)

```
disjointSets forest;
for each vertex v in V do
    forest.makeSet(v);

priorityQueue Q;
Insert edges into Q, keyed by weights

graph T = (V,E) with E = ∅;

while T has fewer than n-1 edges do
    edge e = Q.removeMin()
    Let u, v be the endpoints of e
    if forest.find(v) ≠ forest.find(u) then
        Add edge e to E
        forest.smartUnion
            (forest.find(v),forest.find(u))

return T
```

Priority Queue:	Heap	Sorted Array
To build		
Each removeMin		

Kruskal's Algorithm - analysis



Algorithm *KruskalMST*(*G*)

```
disjointSets forest;
for each vertex v in V do
    forest.makeSet(v);

priorityQueue Q;
Insert edges into Q, keyed by weights

graph T = (V,E) with E = ∅;

while T has fewer than n-1 edges do
    edge e = Q.removeMin()
    Let u, v be the endpoints of e
    if forest.find(v) ≠ forest.find(u) then
        Add edge e to E
        forest.smartUnion
            (forest.find(v),forest.find(u))

return T
```

Priority Queue:	Total Running time:
Heap	
Sorted Array	

Prim's algorithms (1957) is based on the Partition Property:

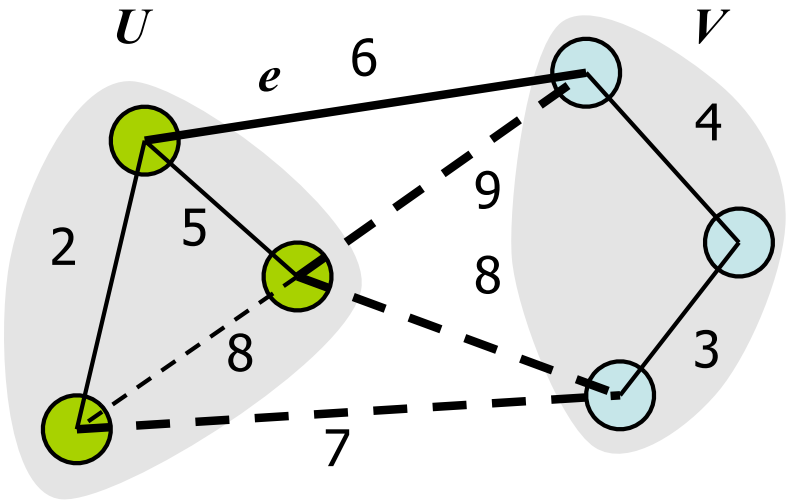
Consider a partition of the vertices of G into subsets U and V.

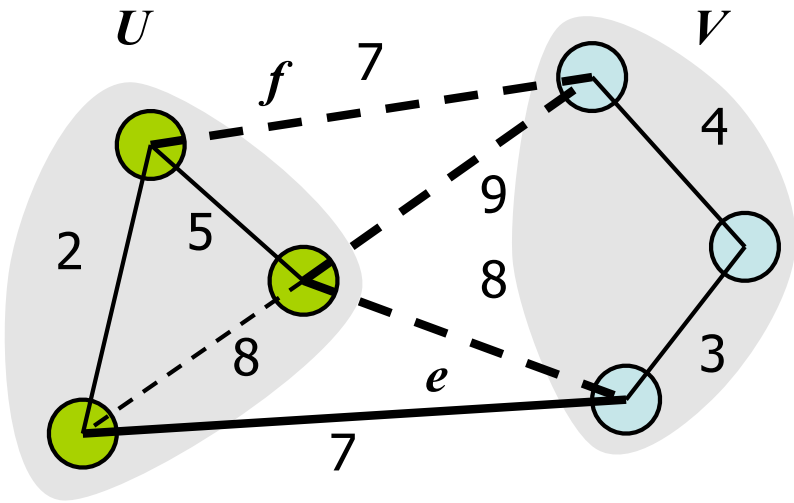
Let e be an edge of minimum weight across the partition.

Then e is part of some minimum spanning tree.

Proof:

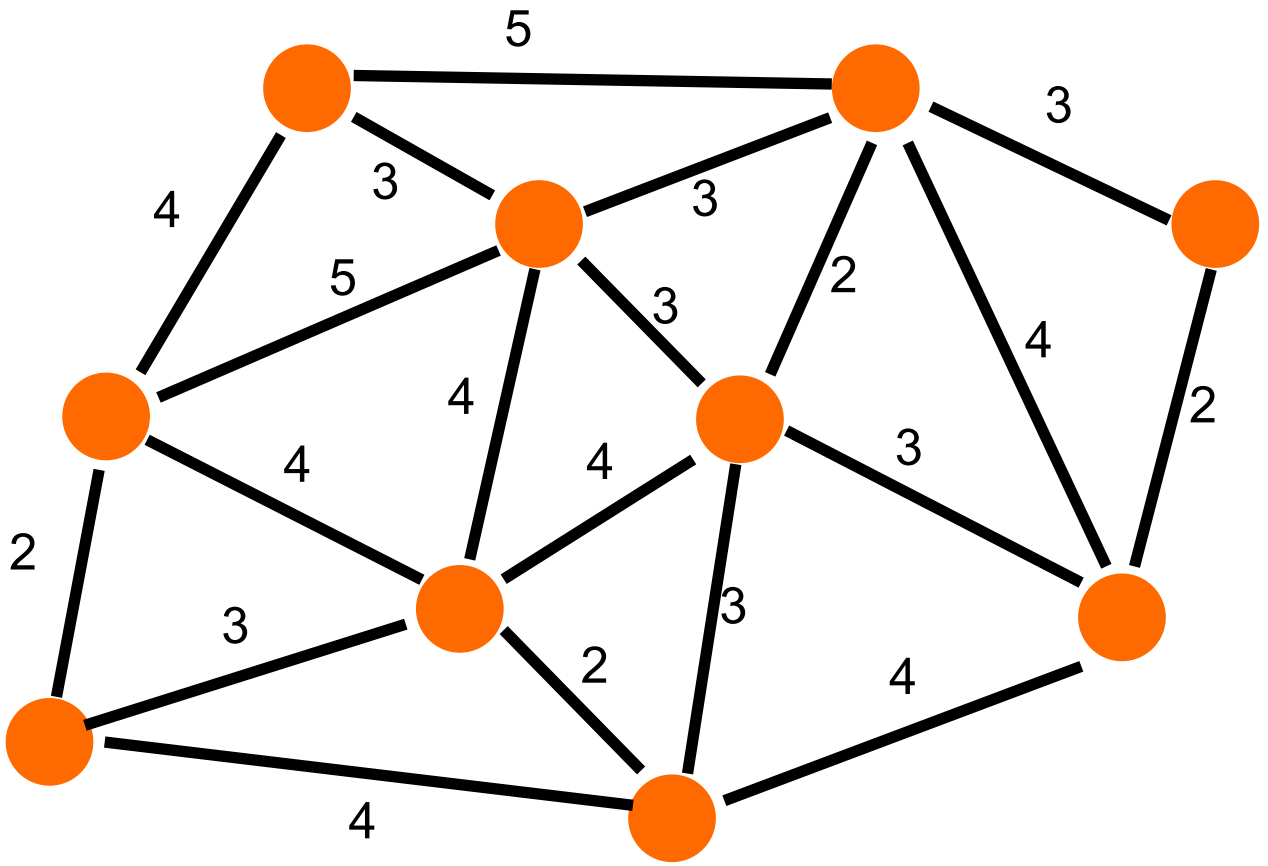
See cs374





MST - minimum total weight spanning tree

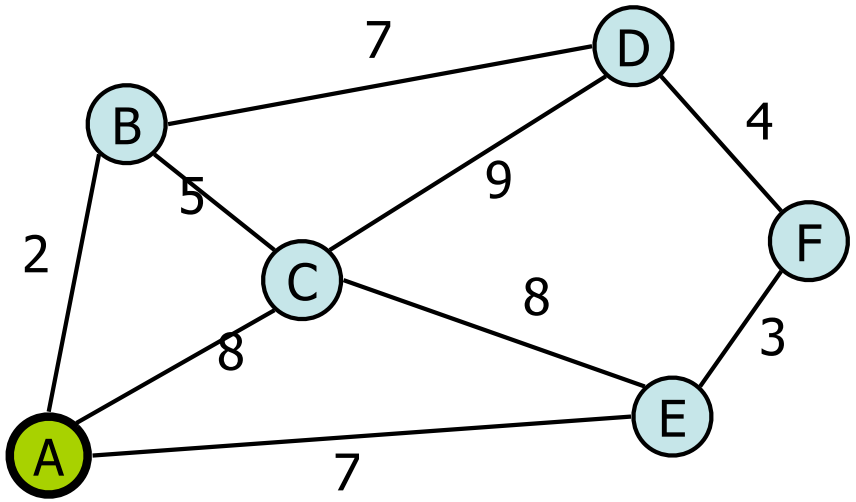
Theorem suggests an algorithm...



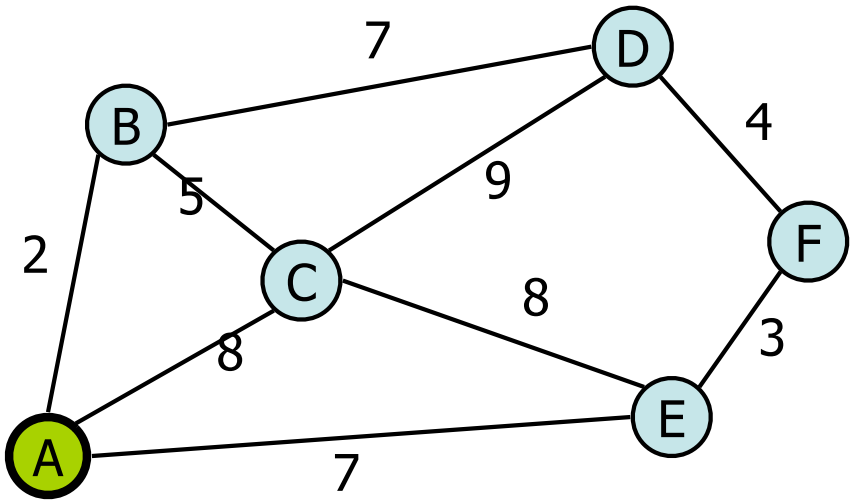
Example of Prim's algorithm -

Initialize structure:

1. For all v, d[v] = "infinity", p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.



Example of Prim's algorithm -



Initialize structure:

1. For all v, d[v] = "infinity", p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.

Repeat these steps n times:

- Find & remove minimum d[] unlabelled vertex: v
- Label vertex v
- For all unlabelled neighbors w of v,

  If cost(v,w) < d[w]

  d[w] = cost(v,w)

  p[w] = v

Prim's Algorithm (undirected graph with unconstrained edge weights):

Initialize structure:

1. For all v, d[v] = "infinity", p[v] = null
2. Initialize source: d[s] = 0
3. Initialize priority (min) queue
4. Initialize set of labeled vertices to ∅.

Repeat these steps n times:

- Remove minimum d[] unlabeled vertex: v
- Label vertex v (set a flag)
- For all unlabeled neighbors w of v,

  If cost(v,w) < d[w]

  d[w] = cost(v,w)

  p[w] = v

	adj mtx	adj list
heap	$O(n^2 + m \log n)$	$O(n \log n + m \log n)$
Unsorted array	$O(n^2)$	$O(n^2)$

Which is best?

Depends on density of the graph:

Sparse

Dense