

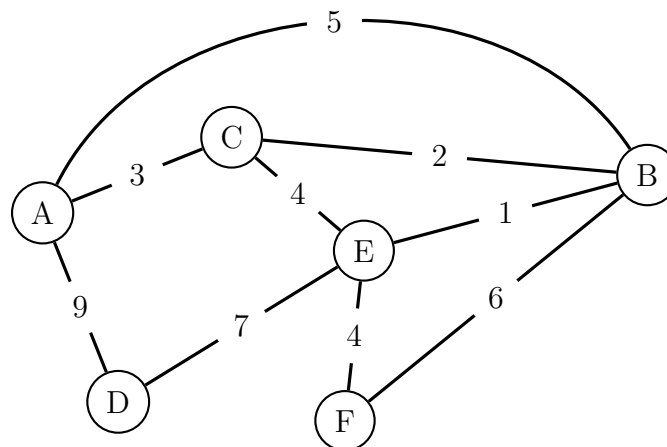
## CS 430 Lecture 26 Activities

### Opening Questions

1. What is the difference between a tree and a graph? All trees are graphs, but not all graphs are trees. In a tree, there is only 1 path between any 2 vertices. Trees are also acyclic since no item can have multiple parents. If trees have  $|V|$  nodes, then there are  $|V| - 1$  edges.
2. Give a recursive definition for a tree. Base case: single node with no children. A tree is ... a node pointing to other trees with no cycles.
3. In a weighted undirected graph, what is the difference between a minimum spanning tree and a shortest path in a graph? A minimum spanning tree is made up of the edges of minimum total weight that still span (connect) all vertices in the graph. This does not assure the minimum path between all vertices. A shortest path is the minimum path distance between two vertices.
4. Since the shortest paths contain the shortest sub-paths (optimal substructure), name an algorithmic approach that we might try to find a shortest path in a graph. Greedy algorithms and Dynamic Programming.

### Minimum Spanning Trees (MST)

1. Give a definition of a Minimum Spanning Tree, and find an MST of the below graph. An MST is a graph that spans over all vertices with no cycles.  $|V| - 1$  edges.



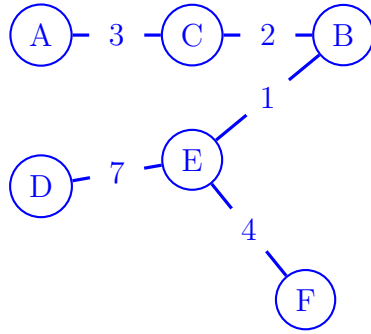


Figure 26.1: One spanning tree of Figure 1..  
Total weight: 22.

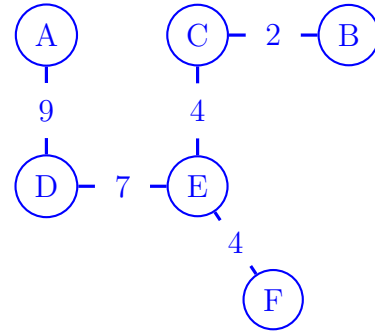
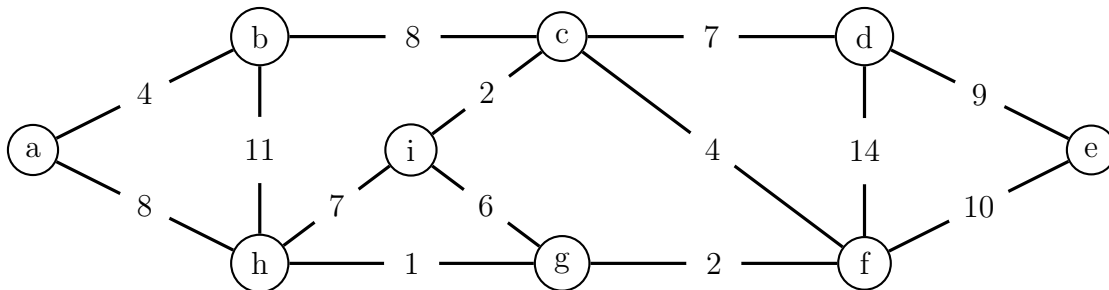


Figure 26.2: Another spanning tree of Figure 1..  
Total Weight: 26.

2. Prove a Minimum Spanning Tree has optimal substructure. Pick any subsets of adjacent vertices in an optimal MST. Those edges that connect that subset of vertices in the MST must also be MST.
3. What are some possible greedy approaches to find a Minimum Spanning Tree? Prove correct or show counterexample.
  - Grow min edges first; no cycles.
  - Prune/remove max edge, but stay connected.
  - Creating edges from visited nodes to unvisited nodes.
  - **Prim**: Min edge from visited vertex set to unvisited vertex set.
  - **Kruskal**: Pick min edge that its vertices are not already in the connected component.
4. Demonstrate your MST algorithm on the following graph and write pseudocode.



Node	Visited?
<i>a</i>	Visited
<i>b</i>	
<i>c</i>	
<i>d</i>	
<i>e</i>	
<i>f</i>	
<i>g</i>	
<i>h</i>	
<i>i</i>	
Started at <i>c</i> .	

Algorithm 26.1 Prim’s Algorithm (MST)	Algorithm 26.2 Kruskal’s Algorithm (MST)
1: <b>function</b> MST-PRIM( $G, W, r$ )	1: <b>function</b> MST-KRUSKAL( $G, w$ )
2: <b>for all</b> $u \in G.V$ <b>do</b>	2: $A \leftarrow \emptyset$
3: $u.key \leftarrow \infty$	3: <b>for all</b> vertex $v \in V[G]$ <b>do</b>
4: $u.\pi \leftarrow \text{NIL}$	4:     MAKE-SET( $v$ )
5: <b>end for</b>	5: <b>end for</b>
6: $r.key \leftarrow 0$	6:   sort the edges of $E$ into increasing
7: $Q \leftarrow G.B$	7: <b>for all</b> edge $(u, v) \in E$ , taken in increas-
8: <b>while</b> $Q \neq \emptyset$ <b>do</b>	ing order <b>do</b>
9: $u \leftarrow \text{EXTRACT-MIN}(Q)$	8: <b>if</b> FIND-SET( $u$ ) $\neq$ FIND-SET( $v$ ) <b>then</b>
10: <b>for all</b> $v \in G.Adj[u]$ <b>do</b>	9: $A \leftarrow A \cup \{(u, v)\}$
11: <b>if</b> $v \in Q$ and $w(u, v) < v.key$ <b>then</b>	10:       UNION( $u, v$ )
12: $v.\pi \leftarrow u$	11: <b>end if</b>
13: $v.key \leftarrow w(u, v)$	12: <b>end for</b>
14: <b>end if</b>	13: <b>end function</b>
15: <b>end for</b>	
16: <b>end while</b>	
17: <b>end function</b>	

Demonstration of Prim (Deleted): <http://en.wikipedia.org/wiki/File:Prim-algorithm-animation-2.gif>

Demonstration of Kruskal: <https://www.cs.usfca.edu/~galles/visualization/Kruskal.html>

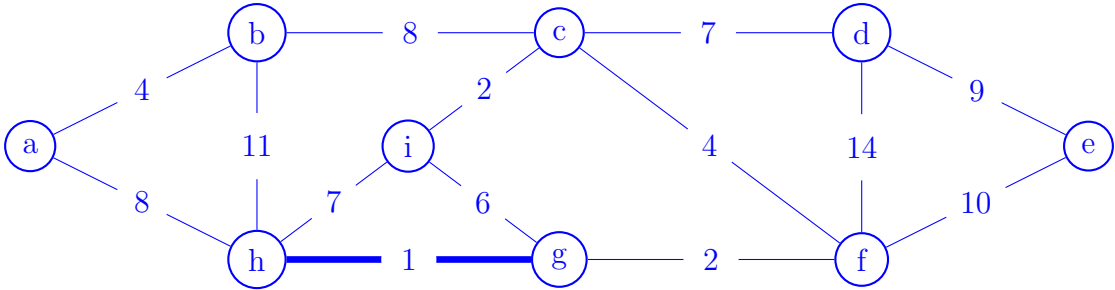
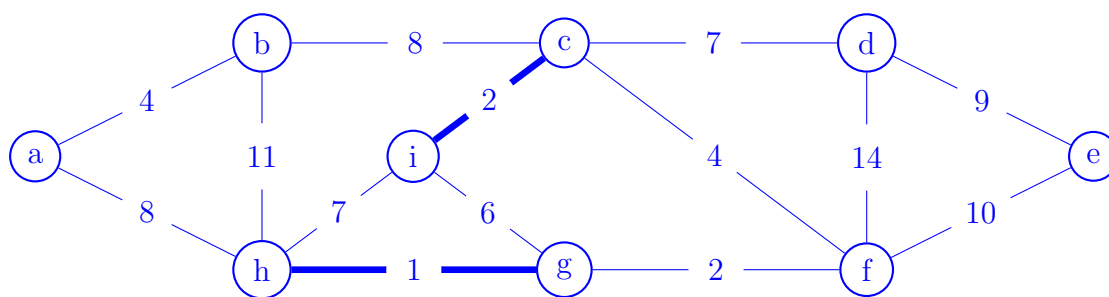
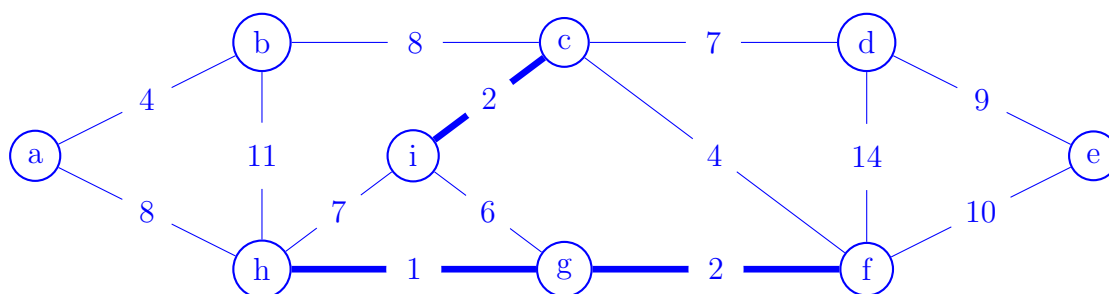
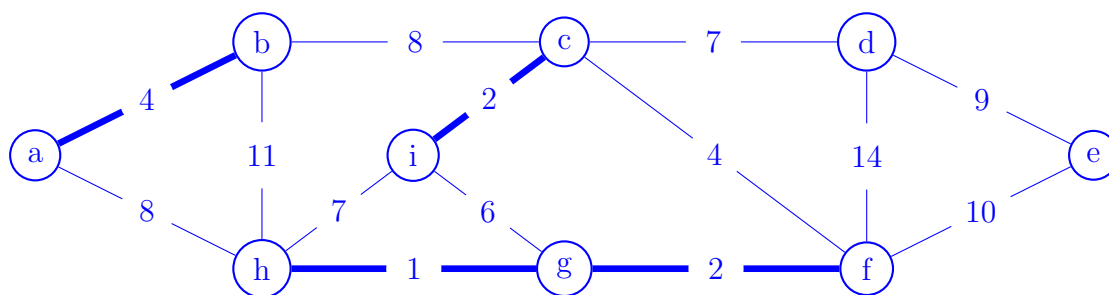
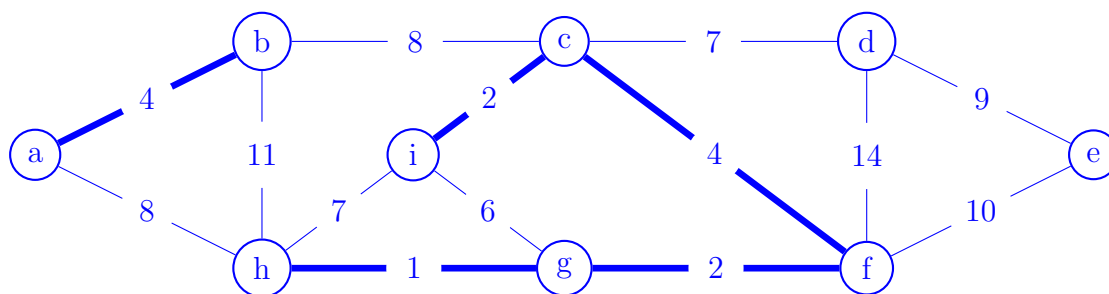
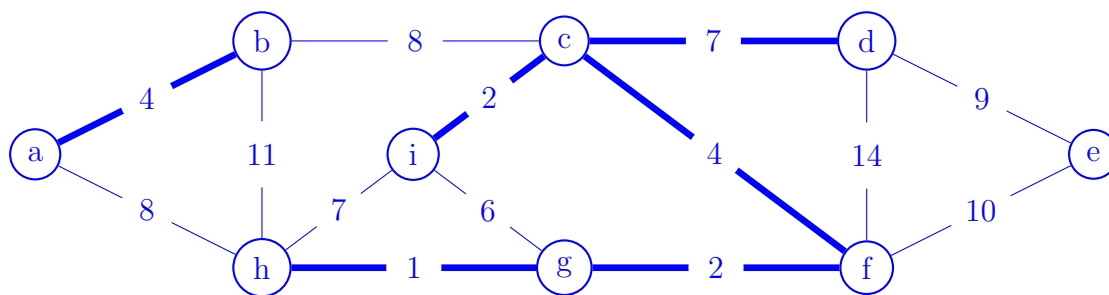
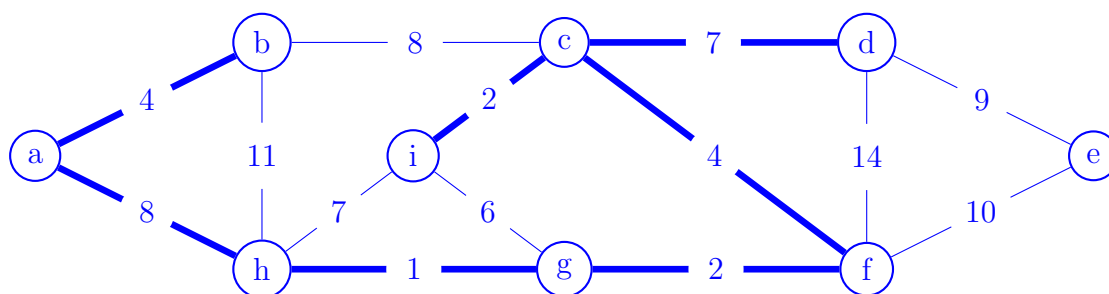
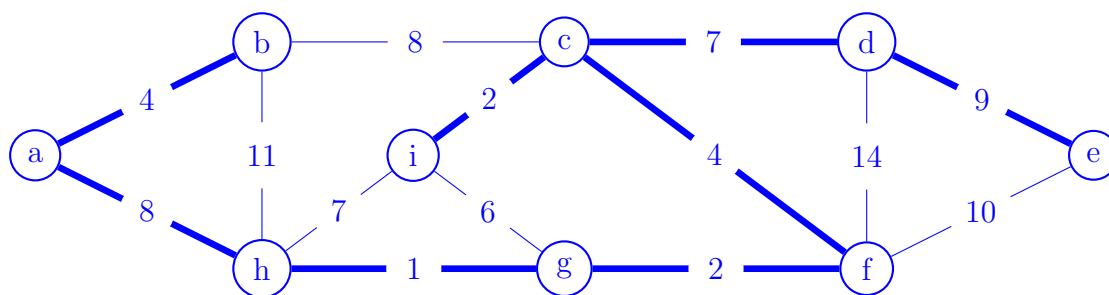


Figure 26.3: Connected vertices  $h$  and  $g$  with an edge with weight 3.

Figure 26.4: Connected vertices  $c$  and  $i$  with an edge with weight 3.Figure 26.5: Connected vertices  $g$  and  $f$  with an edge with weight 3.Figure 26.6: Connected vertices  $a$  and  $b$  with an edge with weight 3.Figure 26.7: Connected vertices  $c$  and  $f$  with an edge with weight 3.

Figure 26.8: Connected vertices  $c$  and  $d$  with an edge with weight 3.Figure 26.9: Connected vertices  $a$  and  $h$  with an edge with weight 3.Figure 26.10: Connected vertices  $d$  and  $e$  with an edge with weight 3.