Spanning tree

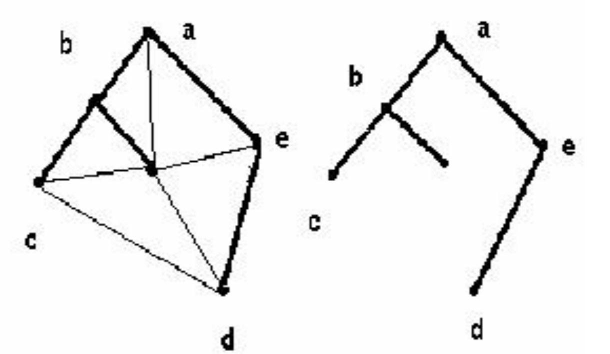
Spanning trees

Given a graph G, a tree T is a *spanning tree* of G if:

- T is a subgraph of G

  and

- T contains all the vertices of G



Spanning trees

- A simple graph with a spanning tree must be connected, because there is a path in the
- spanning tree between any two vertices. The converse is also true; that is, every connected
- simple graph has a spanning tree. We will give an example before proving this result.
-

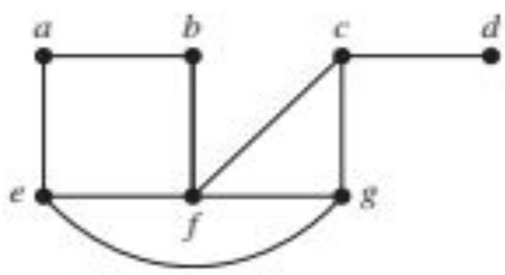


FIGURE 2 **The simple graph *G*.**

Spanning trees

EXAMPLE 1 Find a spanning tree of the simple graph G shown in Figure 2.

Solution: The graph G is connected, but it is not a tree because it contains simple circuits. Remove the edge $\{a, e\}$. This eliminates one simple circuit, and the resulting subgraph is still connected and still contains every vertex of G. Next remove the edge $\{e, f\}$ to eliminate a second simple circuit. Finally, remove edge $\{c, g\}$ to produce a simple graph with no simple circuits. This subgraph is a spanning tree, because it is a tree that contains every vertex of G. The sequence of edge removals used to produce the spanning tree is illustrated in Figure 3.

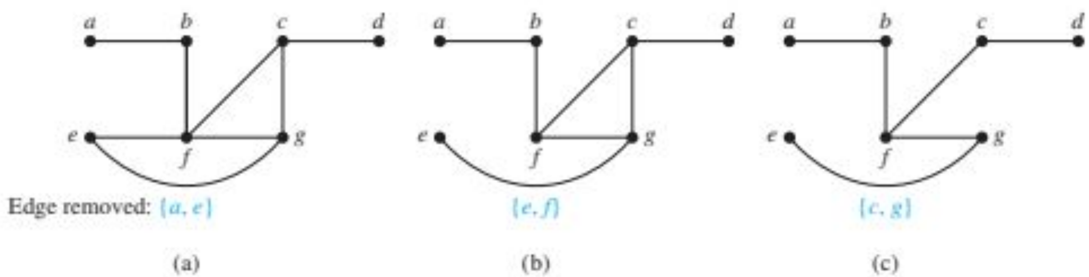
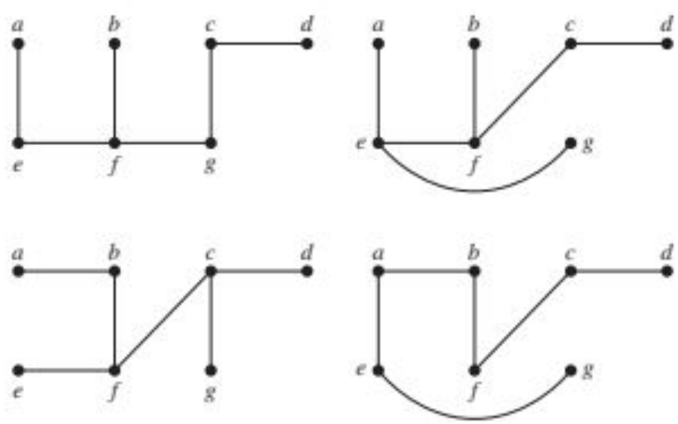
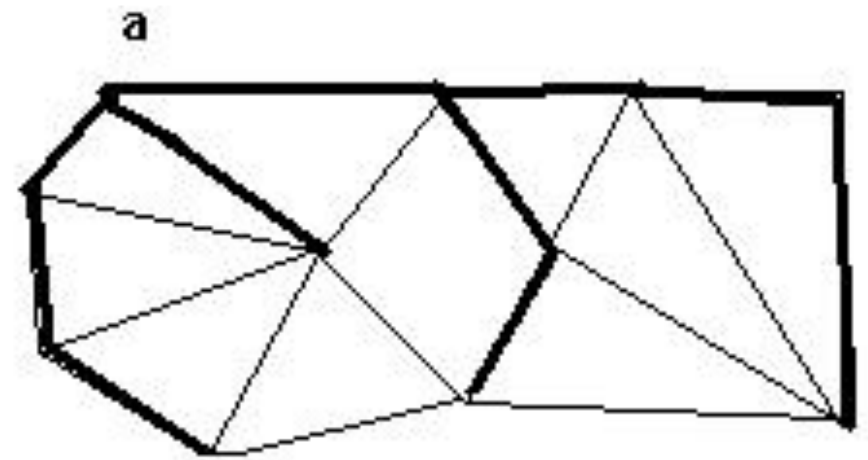


FIGURE 3 **Producing a spanning tree for G by removing edges that form simple circuits.**

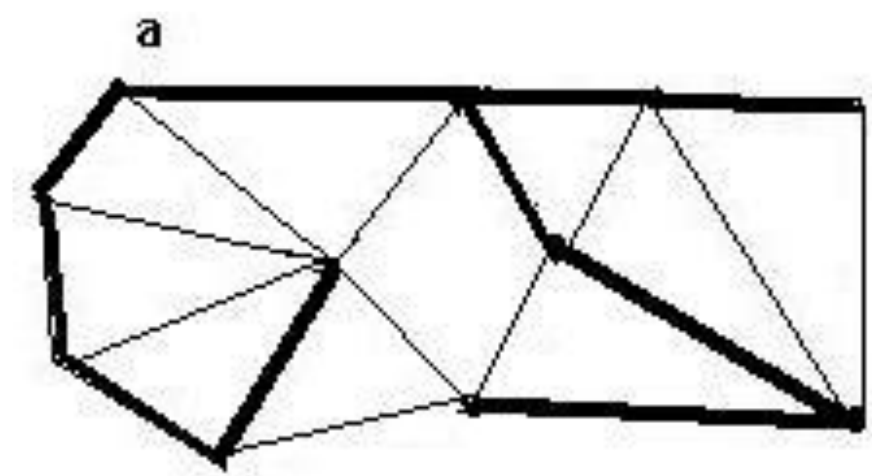


Spanning tree search

- Breadth-first search method



- Depth-first search method (backtracking)



Spanning tree search

EXAMPLE 3 Use depth-first search to find a spanning tree for the graph G shown in Figure 6.

Extra Examples

Solution: The steps used by depth-first search to produce a spanning tree of G are shown in Figure 7. We arbitrarily start with the vertex f. A path is built by successively adding edges incident with vertices not already in the path, as long as this is possible. This produces a path f, g, h, k, j (note that other paths could have been built). Next, backtrack to k. There is no path beginning at k containing vertices not already visited. So we backtrack to h. Form the

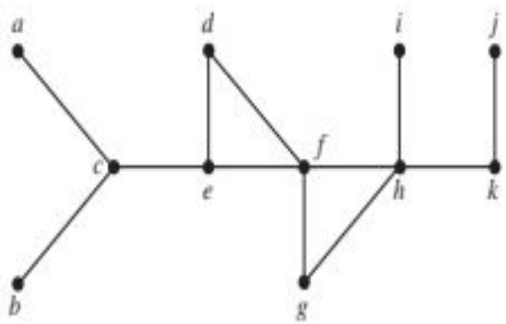


FIGURE 6 The graph G.

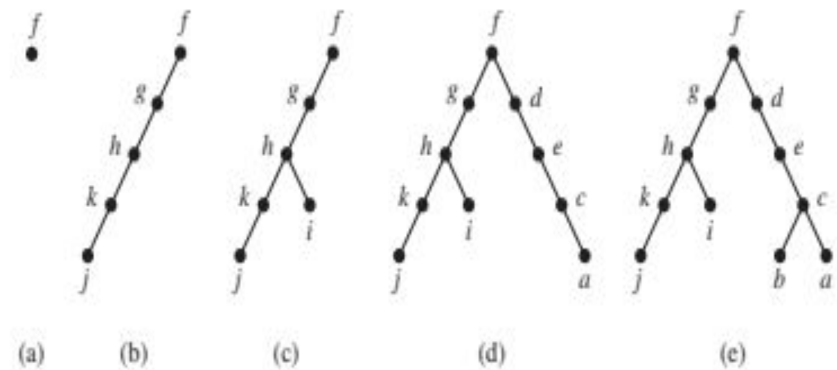


FIGURE 7 Depth-first search of G.

path h, i. Then backtrack to h, and then to f . From f build the path f , d, e, c, a. Then backtrack to c and form the path c, b. This produces the spanning tree. ◂

The edges selected by depth-first search of a graph are called tree edges. All other edges of the graph must connect a vertex to an ancestor or descendant of this vertex in the tree. These edges are called back edges.

Spanning tree search

EXAMPLE 5 Use breadth-first search to find a spanning tree for the graph shown in Figure 9.

Extra Examples

Solution: The steps of the breadth-first search procedure are shown in Figure 10. We choose the vertex e to be the root. Then we add edges incident with all vertices adjacent to e, so edges from e to b, d, f, and i are added. These four vertices are at level 1 in the tree. Next, add the edges from these vertices at level 1 to adjacent vertices not already in the tree. Hence, the edges from b to a and c are added, as are edges from d to h, from f to j and g, and from i to k. The new

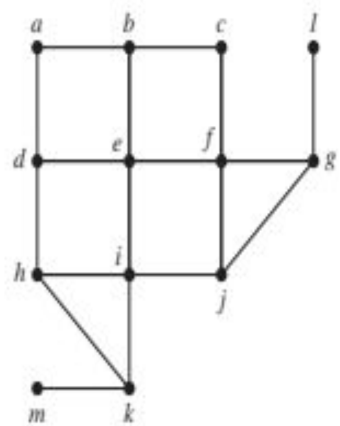


FIGURE 9 **A graph G.**

Spanning tree search

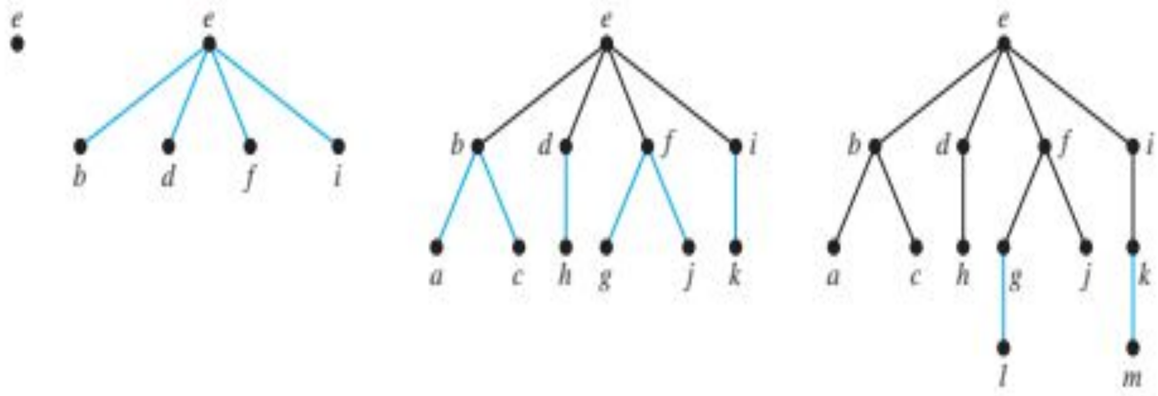


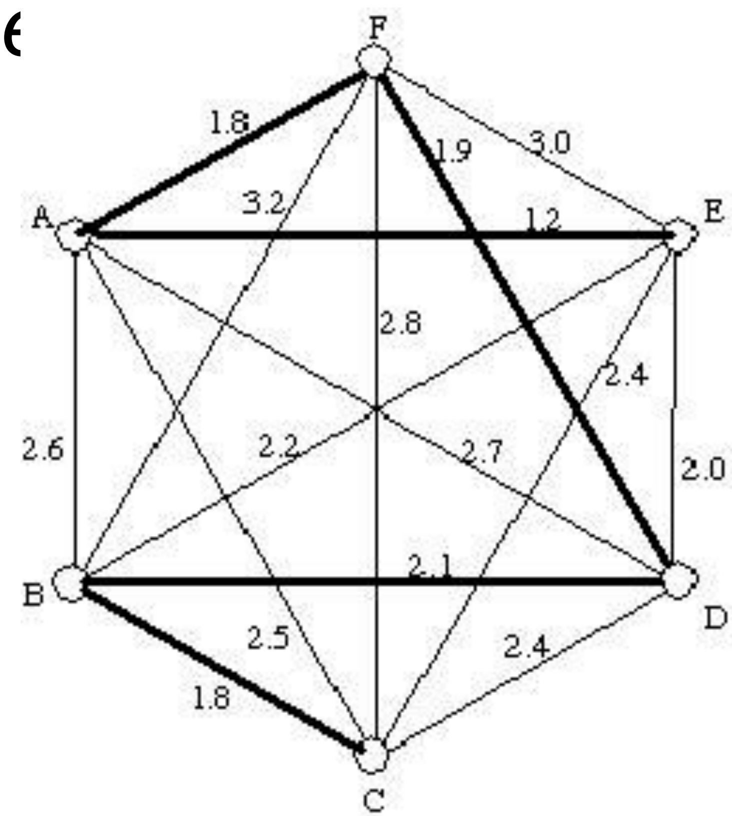
FIGURE 10 **Breadth-first search of *G*.**

vertices *a*, *c*, *h*, *j*, *g*, and *k* are at level 2. Next, add edges from these vertices to adjacent vertices not already in the graph. This adds edges from *g* to *l* and from *k* to *m*. ◀

7.4 Minimal spanning trees

Given a weighted graph G, a *minimum spanning tree* is

- a spanning tree of G
- that has minimum "weight"

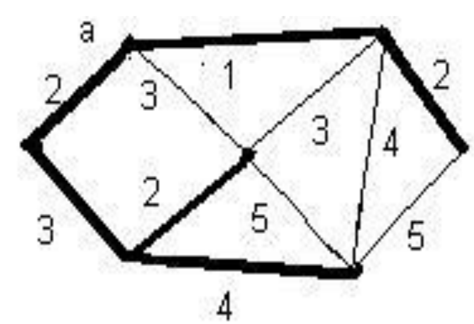


Minimum spanning Trees (1/4)

- A *weighted graph* is a graph for
which each edge has an associated real number weight.
The sum of tree weights of all the edges is the *total weight* of the graph.
A *minimum spanning tree* for a weighted graph is a spanning tree
that has the least possible total weight compared to all other spanning trees
for the graph. If G is a weighted graph and e is an edge of G,
then $w(e)$ denotes the weight of e and $w(G)$ denotes the total weight of G.

1. Prim's algorithm

- <u>Step 0</u>: Pick any vertex as a starting vertex (call it *a*). T = {a}.
- <u>Step 1</u>: Find the edge with smallest weight incident to *a*. Add it to T Also include in T the next vertex and call it *b*.
- <u>Step 2</u>: Find the edge of smallest weight incident to either *a* or *b.* Include in T that edge and the next incident vertex. Call that vertex *c*.
- <u>Step 3</u>: Repeat Step 2, choosing the edge of smallest weight that does not form a cycle until all vertices are in T. The resulting subgraph T is a minimum spanning tree.



Prim's Algorithm

1. Chose an edge with the least weight.
2. Include it in spanning tree, *T.*
3. Select an edge of least weight that is incident with a vertex of an edge in *T.*
4. If it does not create a cycle (simple circuit) with the edges in *T*, then include it in *T*; otherwise discard it.
5. Repeat STEPS 3 and 4 until *T* contains *n*-1 edges.

- If there are two edges with similar smallest weight, chose either one.
- There may be more than one minimum spanning tree for a given connected weighted simple graph.

EXAMPLE: Prim's Algorithm

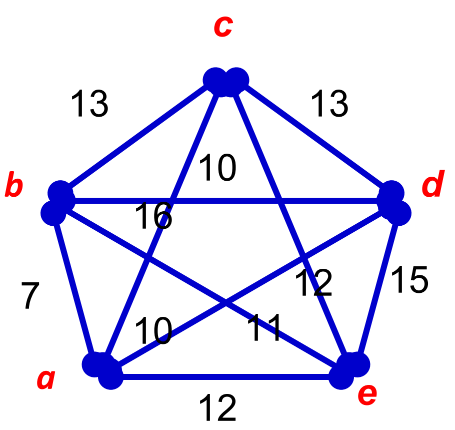
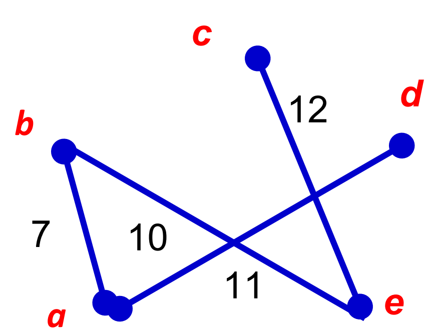
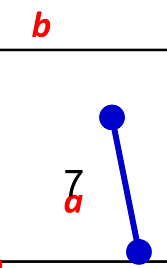
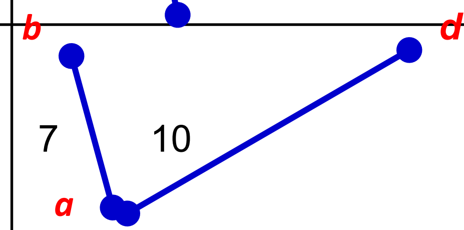
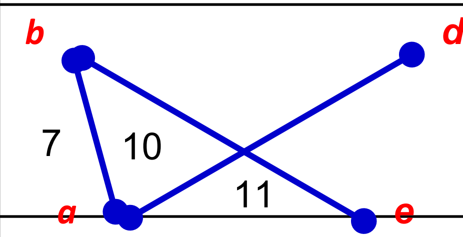
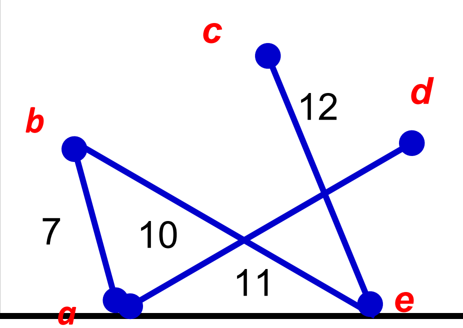


FIGURE 1

The minimum spanning tree is given by:



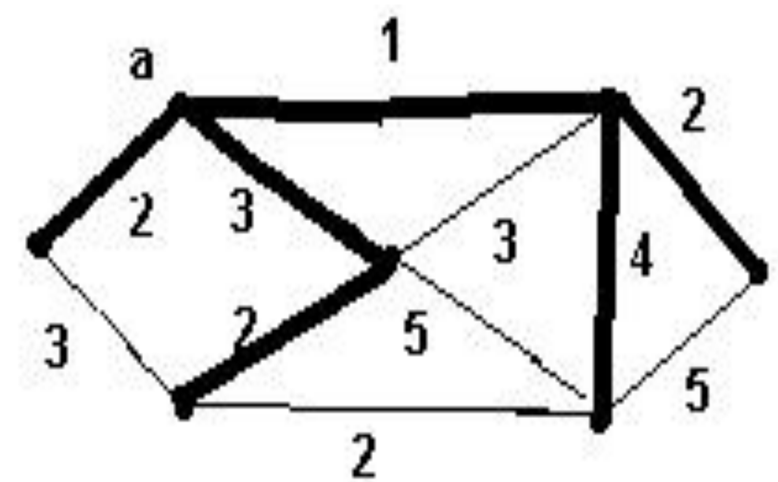
Total weight = 40

CHOICE	EDGE	WEIGHT	SPANNING TREE
1	{*a*, *b*}	7	
2	{*a*, *d*}	10	
3	{*b*, e}	11	
4	{*e*, *c*}	12	

2. Kruskal’s algorithm

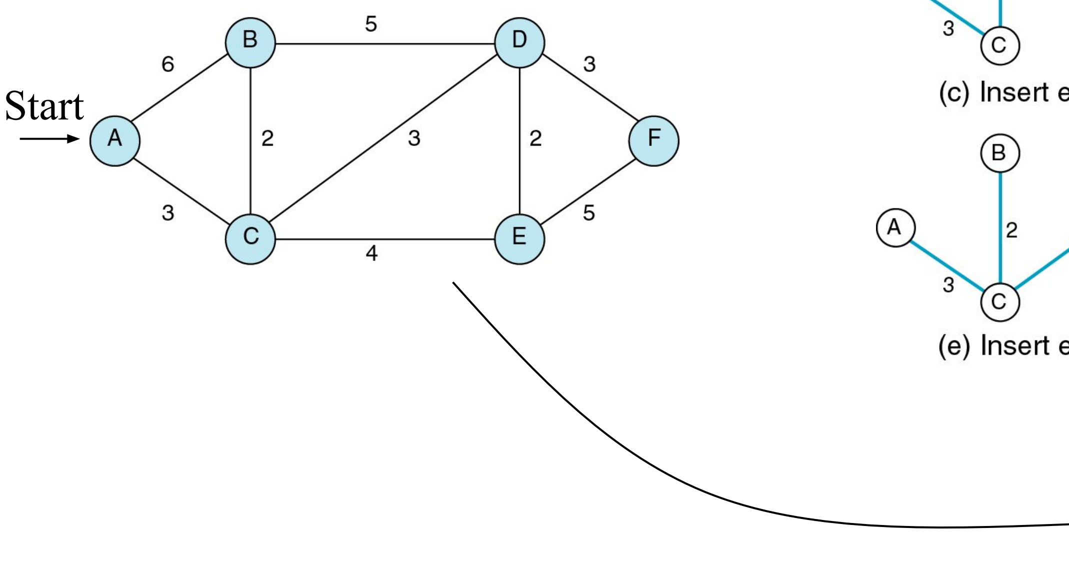
- Step 1: Find the edge in the graph with smallest weight (if there is more than one, pick one at random). Mark it with any given color, say red.
- Step 2: Find the next edge in the graph with smallest weight that doesn't close a cycle. Color that edge and the next incident vertex.

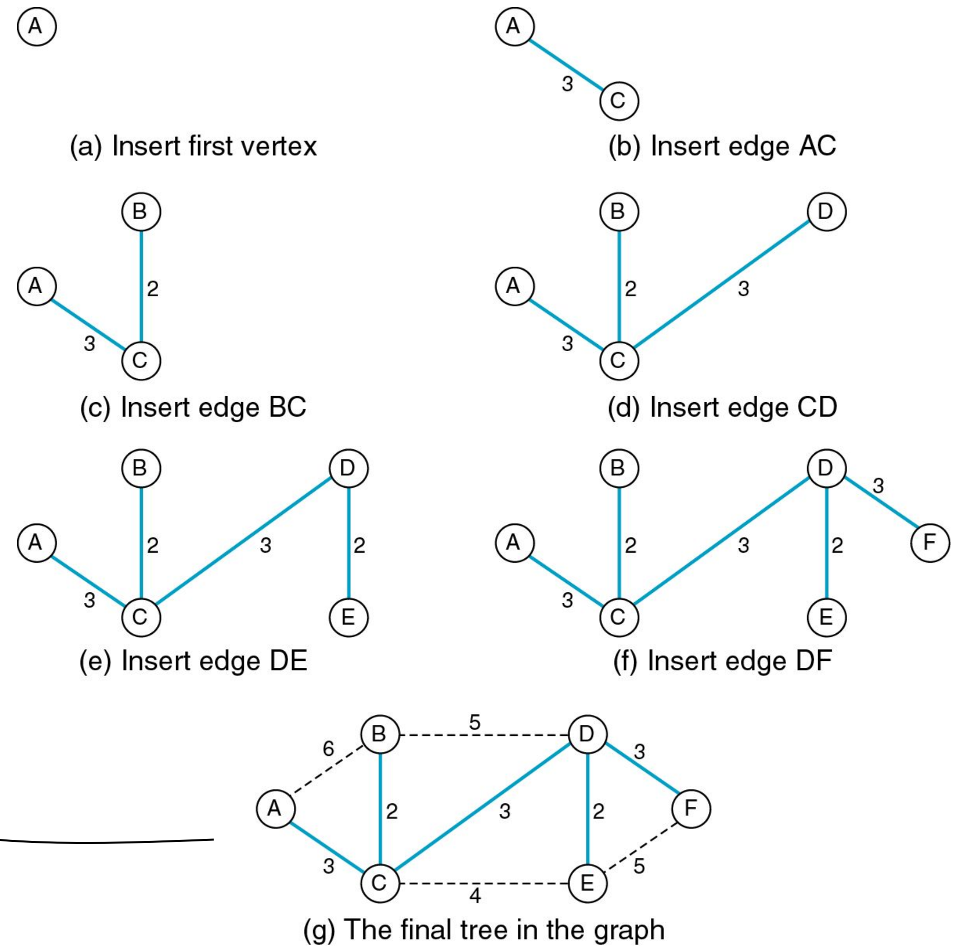
❑ Step 3: Repeat Step 2 until you reach out to every vertex of the graph. The chosen edges form the desired minimum spanning tree.



Minimum spanning Trees (2/4)

- Prim's Algorithm





EXAMPLE: Kruskal's Algorithm

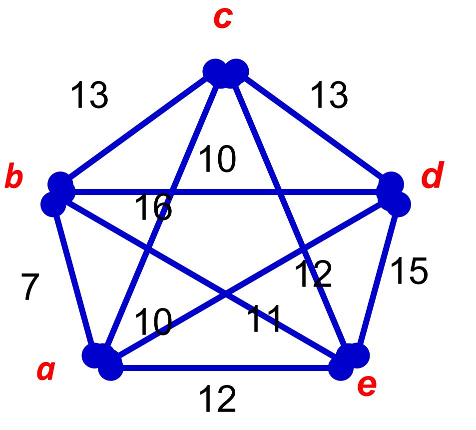
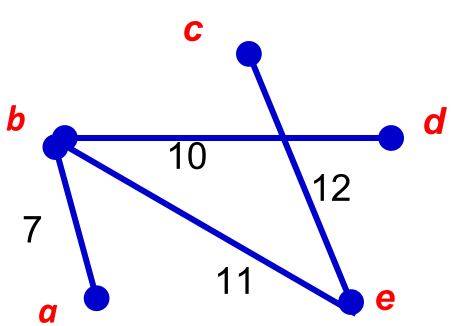
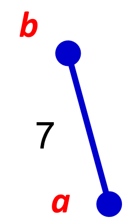
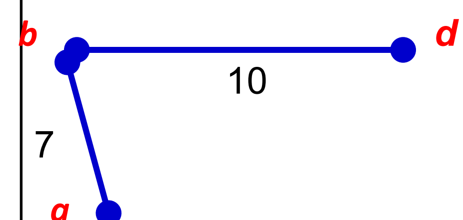
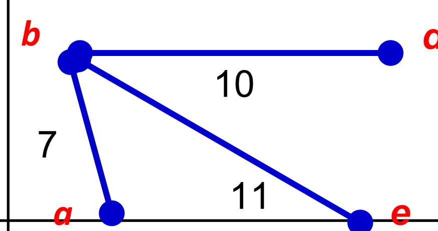
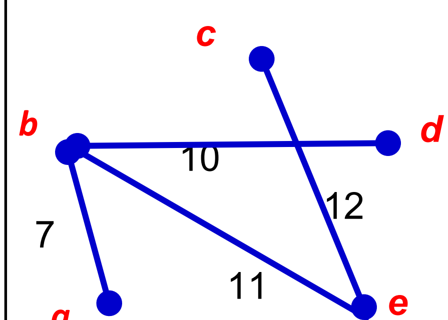


FIGURE 1

The minimum spanning tree is given by:



Total weight = 40

CHOICE	EDGE	WEIGHT	SPANNING TREE
1	{*a*, *b*}	7	
2	{*b*, *d*}	10	
3	{*b*, e}	11	
4	{*e*, *c*}	12	

Minimum spanning Trees (3/4)

- Kruskal’s Algorithm