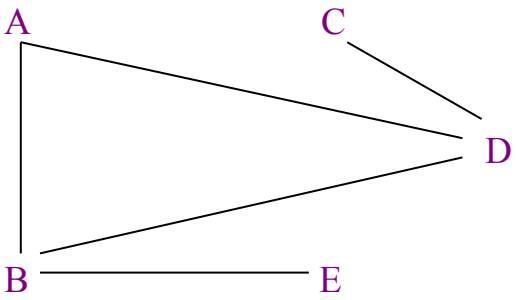
CONCEPT EXERCISES

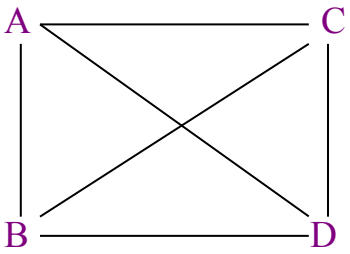
15.1 a. Draw a picture of the following undirected graph:

Vertices: A, B, C, D, E

Edges: {A, B}, {C, D}, {D, A}, {B, D}, {B, E}

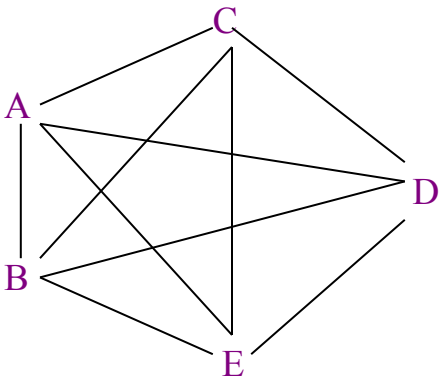


15.2 a. Draw an undirected graph that has four vertices and as many edges as possible. How many edges does the graph have?



The graph has six edges.

b. Draw an undirected graph that has five vertices and as many edges as possible. How many edges does the graph have?



The graph has 10 edges.

c. What is the maximum number of edges for an undirected graph with *V* vertices, where *V* is any non-negative integer?

V(V – 1) / 2

d. Prove the claim you made in part (c).

Hint: Use induction on n.

For V = 1, 2, …, let S_V be the statement

The maximum number of edges for an undirected graph with V vertices is V (V – 1) / 2.

1. (base case) An undirected graph with 1 vertex has no edges:

$$1\ (1 - 1) / 2 = 0$$

We conclude that S_1 is true.

2. (inductive case) Let V be any positive integer, and assume that S_V is true. We need to show that S_{V+1} is true. Let G be an

undirected graph with V + 1 vertices and as many edges as possible. Pick any vertex w in G. Then the number of edges to (or from) w must be V. Let G' be the undirected graph formed by removing, from G, w and all of the edges to w. Because G' has V vertices and the maximum number of edges, the induction hypothesis applies, and G' has V (V – 1) / 2 edges.

Therefore, the total number of edges in G is

$$V + V(V - 1) / 2 = (2V + V(V - 1)) / 2$$

$$= (2V + V^2 - V) / 2$$

$$= (V^2 + V) / 2$$

$$= V(V + 1) / 2$$

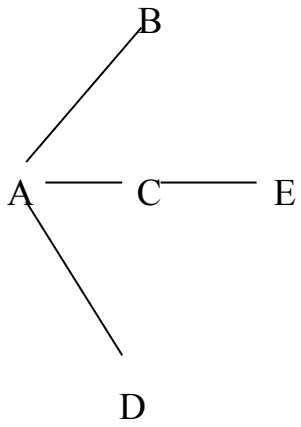
We conclude that S_{V+1} is true.

Therefore, by the Principle of Mathematical Induction, S_V is true for every positive integer V.

e. What is the maximum number of edges for a directed graph with V vertices?

V (V – 1). For each edge in an undirected graph, there will be two edges in the directed graph: one going to a vertex, and one going from that vertex.

15.3 Suppose we have the following undirected graph:



Assume the vertices were inserted into the graph in alphabetical order.

a. Perform a breadth-first traversal of the undirected graph.

queue	vertex returned by next( )
A	
B, C, D	A
C, D	B
D, E	C
E	D
	E

The iteration would visit A, B, C, D, E in that order.

b. Perform a depth-first traversal of the undirected graph.

stack (top vertex is shown **leftmost**)	vertex returned by next( )
A	
D, C, B	A
C, B	D

E, B	C
B	E
	B

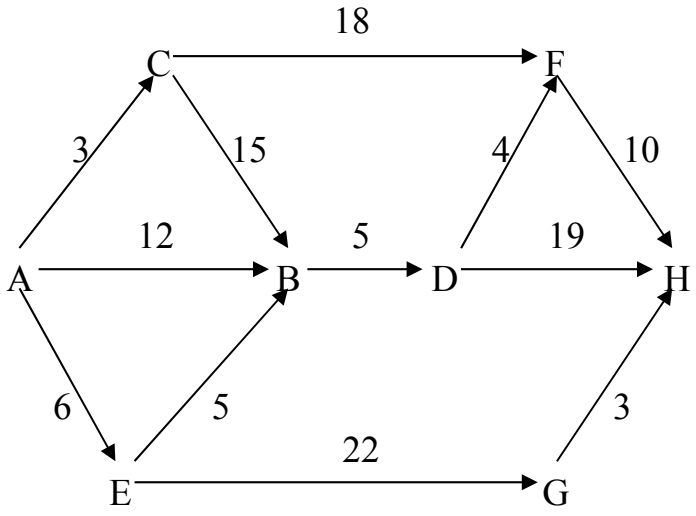
The iteration would visit A, D, C, E, B in that order.

15.4 Develop a high-level algorithm, based on the isConnected() algorithm in Section 15.5.2, to determine if an undirected graph is connected.

```
public boolean isConnected()
{
        Choose any Vertex v in this undirected graph.

        // Count the number of vertices reachable from v.
        Construct a BreadthFirstIterator, bfItr, starting at v.
        int count = 0;
        while (bfItr.hasNext())
        {
                bfItr.next();
                count++;
        } // while
        if (count < number of vertices in this digraph)
                return false;
        return true;
} // algorithm for isConnected in an undirected graph
```

15.5 For the network given below, determine the shortest path from A to H by brute force, that is, list all paths and see which one has the lowest total weight.



path	total weight	
A, C, F, H	31	
A, C, B, D, H	42	
A, C, B, D, F, H	37	
A, B, D, H	36	
A, B, D, F, H	31	
A, E, G, H	31	
A, E, B, D, H	35	
A, E, B, D, F, H	30	← winner

15.6 For the network given in Exercise 15.5, use Dijkstra's algorithm (getShortestPath) to find the shortest path from A to H.

Initially, we have

weightSum	predecessor	pq
A, 0.0	A	<A, 0.0>
B, 12.0	**null**	
C, 3.0	**null**	

D, 10000.0	**null**
E, 6.0	**null**
F, 10000.0	**null**
G, 10000.0	**null**
H, 10000.0	**null**

The pair <A, 0.0> is removed from pq. Since that pair's weight is less than or equal to A's weightSum value, the weightSum and predecessor of each neighbor of A are updated, and the neighbor and its total weight (so far) are added to pq.

weightSum	predecessor	pq
A, 0.0	A	<C, 3.0>
B, 12.0	A	<E, 6.0>
C, 3.0	A	<B, 12.0>
D, 10000.0	**null**	
E, 6.0	A	
F, 10000.0	**null**	
G, 10000.0	**null**	
H, 10000.0	**null**	

When <C, 3.0> is removed from pq, the information on vertex F is updated, and <F, 21.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<E, 6.0>
B, 12.0	A	<B, 12.0>
C, 3.0	A	<F, 21.0>
D, 10000.0	**null**	
E, 6.0	A	
F, 21.0	C	
G, 10000.0	**null**	
H, 10000.0	**null**	

When <E, 6.0> is removed from pq, the information on vertices B and G is updated, and both <B, 11.0> and <G, 28.0> are added to pq:

weightSum	predecessor	pq
A, 0.0	A	<B, 11.0>
B, 11.0	E	<B, 12.0>

C, 3.0	A	<F, 21.0>
D, 10000.0	**null**	<G, 28.0>
E, 6.0	A	
F, 21.0	C	
G, 28.0	E	
H, 10000.0	**null**	

When <B, 11.0> is removed from pq, the information on vertex D is updated, and <D, 16.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<B, 12.0>
B, 11.0	E	<D, 16.0>
C, 3.0	A	<F, 21.0>
D, 16.0	B	<G, 28.0>
E, 6.0	A	
F, 21.0	C	
G, 28.0	E	
H, 10000.0	**null**	

When <B, 12.0> is removed from pq, nothing is updated. When <D, 16.0> is removed from pq, the information on vertices F and H is updated, and both <F, 20.0> and <H, 35.0> are added to pq:

weightSum	predecessor	pq
A, 0.0	A	<F, 20.0>
B, 11.0	E	<F, 21.0>
C, 3.0	A	<G, 28.0>
D, 16.0	B	<H, 35.0>
E, 6.0	A	
F, 20.0	D	
G, 28.0	E	
H, 35.0	D	

When <F, 20.0> is removed from pq, the information on vertex H is updated, and <H, 30.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<F, 21.0>
B, 11.0	E	<G, 28.0>

C, 3.0	A	<H, 30.0>
D, 16.0	B	<H, 35.0>
E, 6.0	A	
F, 20.0	D	
G, 28.0	E	
H, 30.0	F	

When <F, 21.0> is removed from pq, nothing happens. When <G, 28.0> is removed from pq, nothing happens. When <H, 30.0> is removed from pq, the shortest (that is, lowest-total-weight) path from A to H is printed, starting with H and pre-pending the predecessors:

A, E, B, D, F, H

15.8 gnore the direction of arrows in the figure for Exercise 15.5. Then that figure depicts an undirected network. Use Prim's algorithm to find a minimum spanning tree for that undirected network.

Initially, A is placed in T, and triples for A's edges are added to pq:

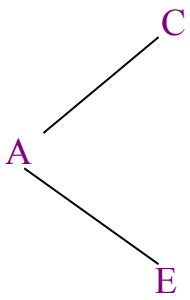
T	pq
A	<A, C, 3.0>
	<A, E, 6.0>
	<A, B, 12.0>

When <A, C, 3.0> is removed from pq, C and edge (A, C) are added to T, and C's new edge triples are added to pq:

T	pq
C	<A, E, 6.0>
A	<A, B, 12.0>
	<C, B, 15.0>
	<C, F, 18.0>

When <A, E, 6.0> is removed from pq, E and edge (A, E) are added to T, and E's new edge triples are added to pq:

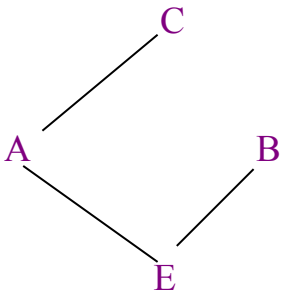
T	pq



<E, B, 5.0>
<A, B, 12.0>
<C, B, 15.0>
<C, F, 18.0>
<E, G, 22.0>

When <E, B, 5.0> is removed from pq, B and edge (E, B) are added to T, and B's new edge triple is added to pq:

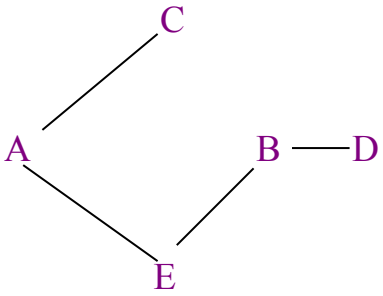
T



pq
<B, D, 5.0>
<A, B, 12.0>
<C, B, 15.0>
<C, F, 18.0>
<E, G, 22.0>

When <B, D, 5.0> is removed from pq, B and edge (B, D) are added to T, and D's new edge triples are added to pq:

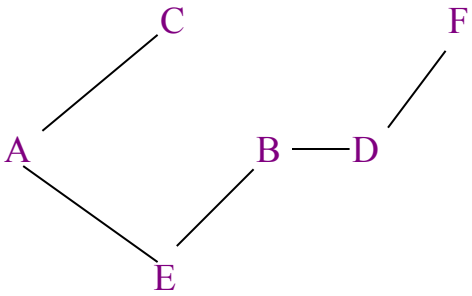
T



pq
<D, F, 4.0>
<A, B, 12.0>
<C, B, 15.0>
<C, F, 18.0>
<D, H, 19.0>
<E, G, 22.0>

When <D, F, 4.0> is removed from pq, F and edge (D, F) are added to T, and F's new edge triple is added to pq:

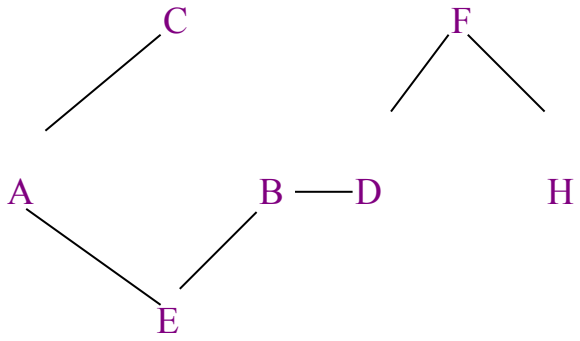
T



pq
<F, H, 10.0>
<A, B, 12.0>
<C, B, 15.0>
<C, F, 18.0>
<D, H, 19.0>
<E, G, 22.0>

When <F, H, 10.0> is removed from pq, H and (F, H) are added to T, and <H, G, 3.0> is added to pq.

T

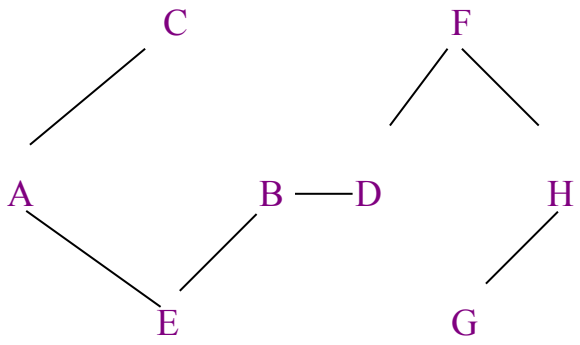


pq

<H, G, 3.0>
<A, B, 12.0>
<C, B, 15.0>
<C, F, 18.0>
<D, H, 19.0>
<E, G, 22.0>

When <H, G, 3.0> is removed from pq, G and (H, G) are added to T. The algorithm is finished because T contains all of the vertices in the network:

T



15.9 Ignore the direction of arrows and assume all weights are 1.0 in the figure for Exercise 15.5. Use Dijkstra's algorithm to find a shortest path (fewest edges) from A to H.

Initially, we have

weightSum	predecessor	pq
A, 0.0	A	<A, 0.0>
B, 1.0	**null**	

C, 1.0	**null**
D, 10000.0	**null**
E, 1.0	**null**
F, 10000.0	**null**
G, 10000.0	**null**
H, 10000.0	**null**

Since that pair's weight is less than or equal to A's weightSum value, the weightSum and predecessor of each neighbor of A are updated, and the neighbor and its total weight (so far) are added to pq.

weightSum	predecessor	pq
A, 0.0	A	<B, 1.0>
B, 1.0	A	<C, 1.0>
C, 1.0	A	<E, 1.0>
D, 10000.0	**null**	
E, 1.0	A	
F, 10000.0	**null**	
G, 10000.0	**null**	
H, 10000.0	**null**	

When <B, 1.0> is removed from pq, the information on D is updated and <D, 2.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<C, 1.0>
B, 1.0	A	<E, 1.0>
C, 1.0	A	<D, 2.0>
D, 2.0	B	
E, 1.0	A	
F, 10000.0	**null**	
G, 10000.0	**null**	
H, 10000.0	**null**	

When <C, 1.0> is removed from pq, the information on F is updated and <F, 2.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<E, 1,0>
B, 1.0	A	<D, 2.0>

C, 1.0	A	<F, 2.0>
D, 2.0	B	
E, 1.0	A	
F, 2.0	C	
G, 10000.0	**null**	
H, 10000.0	**null**	

When <E, 1.0> is removed from pq, the information on vertex G is updated, and <G, 2.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<D, 2,0>
B, 1.0	A	<F, 2.0>
C, 1.0	A	<G, 2.0>
D, 2.0	B	
E, 1.0	A	
F, 2.0	C	
G, 2.0	E	
H, 10000.0	**null**	

When <D, 2.0> is removed from pq, the information on vertex H is updated, and <H, 3.0> is added to pq:

weightSum	predecessor	pq
A, 0.0	A	<F, 2.0>
B, 1.0	A	<G, 2.0>
C, 1.0	A	<H, 3.0>
D, 2.0	B	
E, 1.0	A	
F, 2.0	C	
G, 2.0	E	
H, 3.0	D	

When <F, 2.0> and <G, 2.0> are removed from pq, the edges (F, H) and (G, H) are investigated, but neither pq, weightSum, nor predecessor are modified. When <H, 3.0> is removed from pq, the shortest path (lowest-total-weight and fewest vertices) from A to H is printed, starting with H and pre-pending the predecessors:

A, B, D, H

The length of this path is 3; recall that the weight of a path in an undirected graph is the number of edges in the path, and that number is one less than the number of vertices in the path.