10.1.1

In the graph shown in Figure 10.1, what vertices are adjacent to D? In Figure 10.3?

The vertices adjacent to D in Figure 10.1 are A and E.
The vertix adjacent to D in Figure 10.3 is A.

10.1.3

In Figure 10.4, what is the shortest path from Philadelphia to Chicago?

The shortest path from Philadelphia to Chicago as shown in Figure 10.4 is Philadelphia → Pittsburg →Columbus → Indianapolis → Chicago.

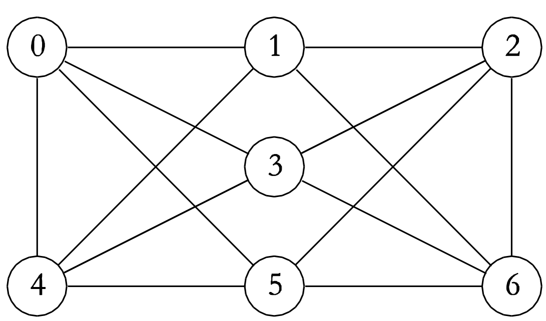
10.2.1

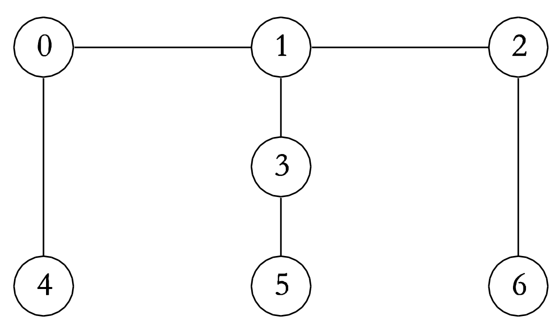
Use the constructors in Table 10.1 to create the Edge objects connecting vertices 9 through 12 for the graph in Figure 10.8.

new Edge(9, 10);
new Edge(10, 9);
new Edge(9, 11);
new Edge(11, 9);
new Edge(9, 12);
new Edge(12, 9);
new Edge(11, 12);
new Edge(12, 11);

10.3.1

Represent the following graphs using adjacency lists.





Left Graph:

0	1, 3, 4, 5
1	0, 2, 3, 4
2	1, 3, 5, 6
3	0, 1, 2, 4, 5, 6

Right Graph:

0	1, 4
1	0, 2, 3
2	1, 6
3	1, 5

4	0, 1, 3, 5
5	0, 2, 3, 4, 6
6	1, 2, 3, 5

4	0
5	3
6	2

10.3.3

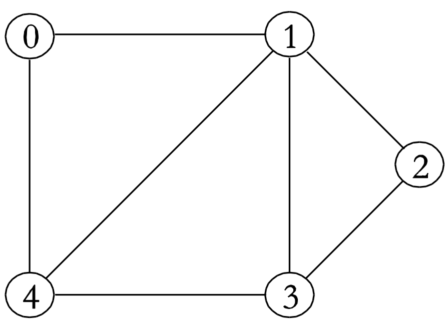
For each graph in Exercise 1, what are the $|V|$, the $|E|$, and the density? Which representation is best for each graph? Explain your answers.

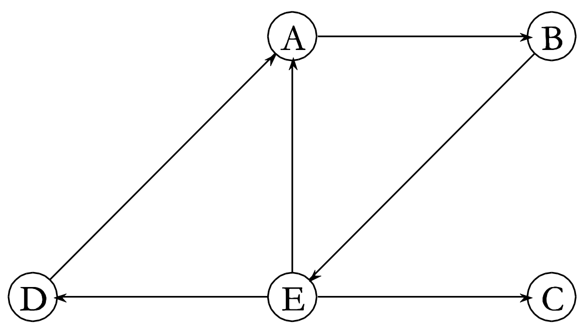
The left graph has a $|V|$ of 7 and $|E|$ of 16. Its density is 2.29. An adjacency matrix is most efficient since $|E|$ is 67% of $\frac{1}{2}|V|^2$.

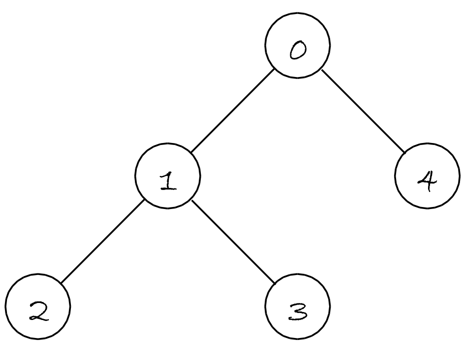
The right graph has a $|V|$ of 7 and $|E|$ of 6. Its density is 0.86. An adjacency list is most efficient since $|E|$ is less than $|V|$.

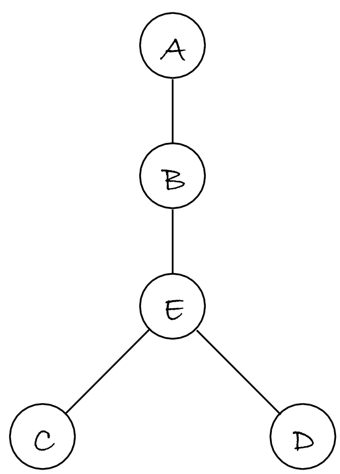
10.4.1

Show the breadth-first search trees for the following graphs.



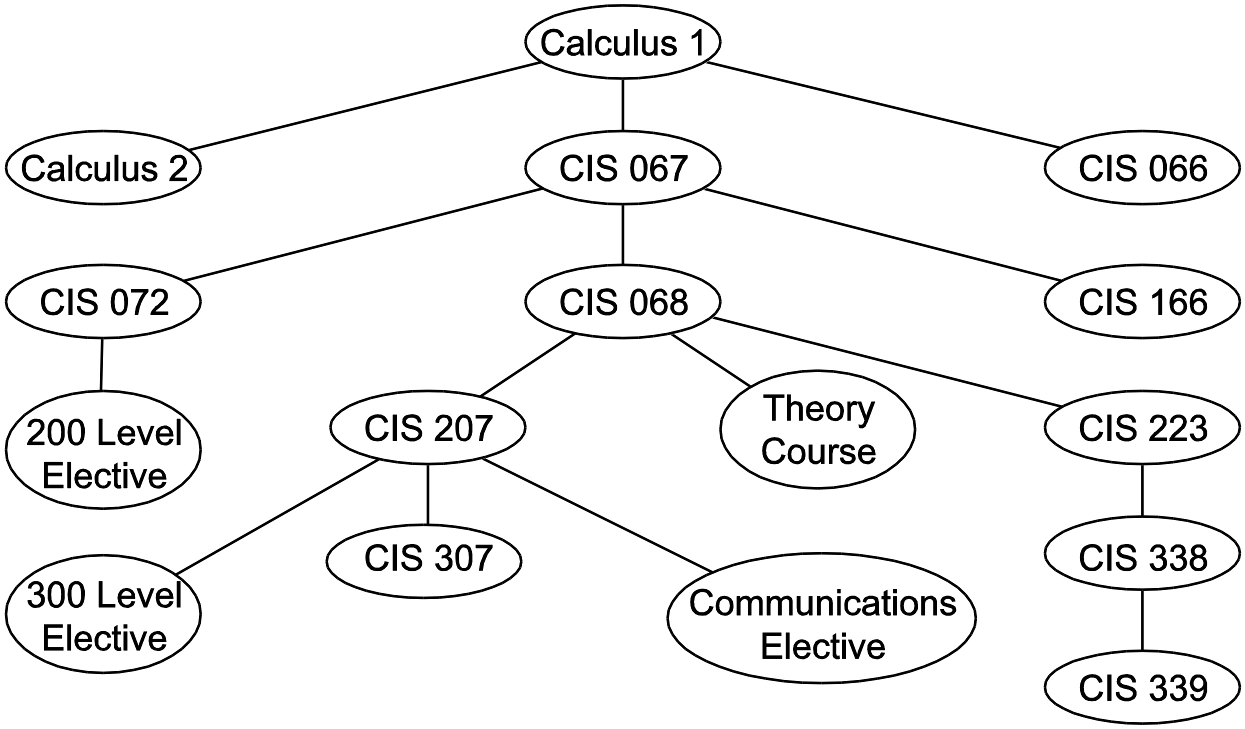






10.5.1

Draw the depth-first search tree of the graph in Figure 10.24 and then list the vertices in reverse finish order.

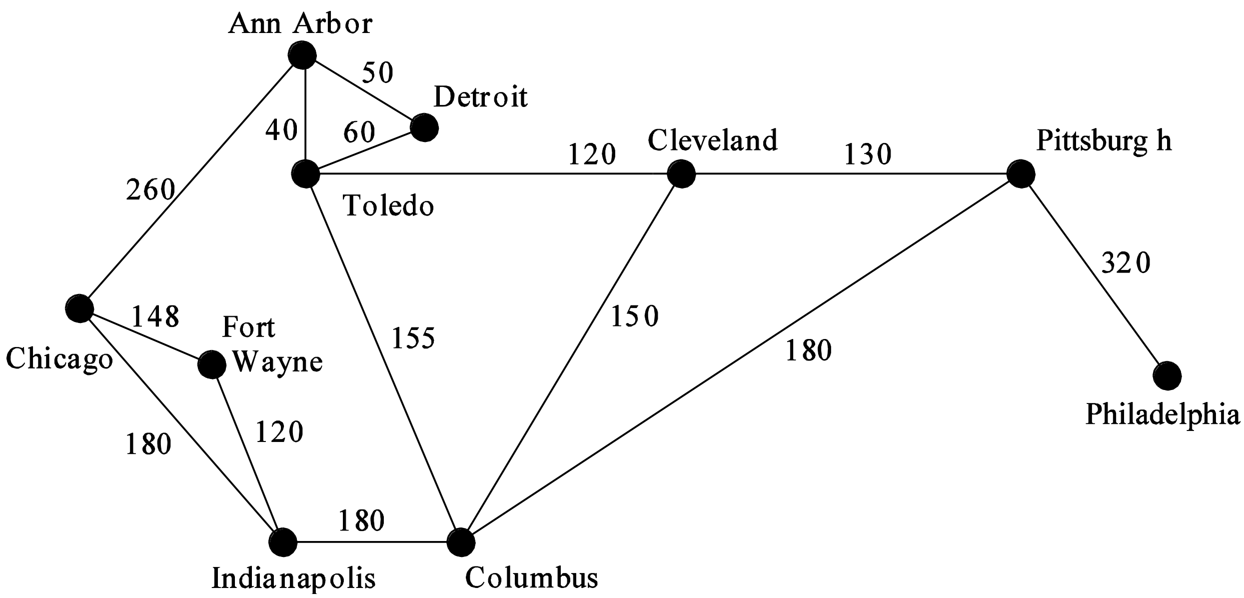


The reverse finish order is:

Calculus 1
CIS 066
CIS 067
CIS 166
CIS 068
CIS 223
CIS 338
CIS 339
Theory Course
CIS 207
Communications Elective
CIS 307
300 Level Elective
CIS 072
200 Level Elective
Calculus 2

10.6.1

Trace the execution of Dijkstra's algorithm to find the shortest path from Philadelphia to the other cities shown in the following graph.



Assigning the vertices as follows:

0	Philadelphia
1	Pittsburgh
2	Cleveland
3	Columbus
4	Toledo
5	Detroit
6	Ann Arbor
7	Indianapolis
8	Fort Wayne
9	Chicago

V - S: [1, 2, 3, 4, 5, 6, 7, 8, 9]

v	pred	dist
1	0	320
2	0	Infinity
3	0	Infinity
4	0	Infinity
5	0	Infinity
6	0	Infinity
7	0	Infinity
8	0	Infinity
9	0	Infinity

dist[1] is minDist = 320

dist[2] = 450
pred[2] = 1
dist[3] = 500
pred[3] = 1
V - S: [2, 3, 4, 5, 6, 7, 8, 9]

v	pred	dist
2	1	450
3	1	500
4	0	Infinity
5	0	Infinity
6	0	Infinity
7	0	Infinity
8	0	Infinity
9	0	Infinity

dist[2] is minDist = 450
dist[4] = 570
pred[4] = 2
V - S: [3, 4, 5, 6, 7, 8, 9]

v	pred	dist
3	1	500
4	2	570
5	0	Infinity
6	0	Infinity
7	0	Infinity
8	0	Infinity
9	0	Infinity

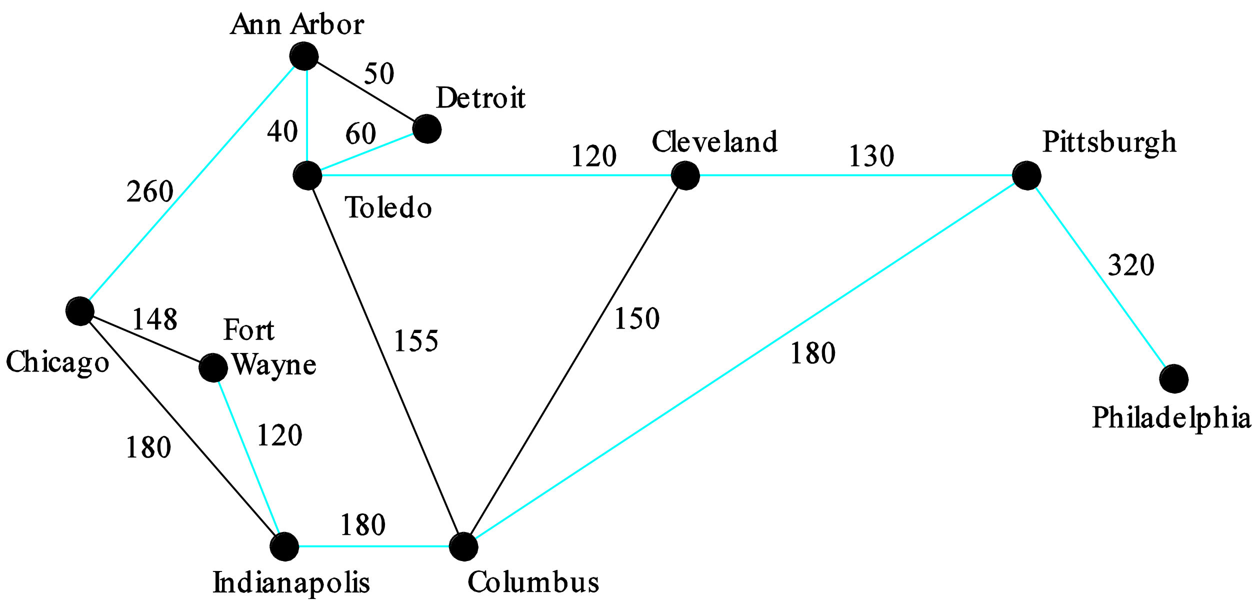
dist[3] is minDist = 500
dist[7] = 680
pred[7] = 3
V - S: [4, 5, 6, 7, 8, 9]

v	pred	dist
4	2	570
5	0	Infinity
6	0	Infinity
7	3	680
8	0	Infinity
9	0	Infinity

dist[4] is minDist = 570
dist[5] = 630
pred[5] = 4
dist[6] = 610

```
pred[6] = 4
V - S: [5, 6, 7, 8, 9]
   v pred dist
   5   4      630
   6   4      610
   7   3      680
   8   0  Infinity
   9   0  Infinity
dist[6] is minDist = 610
dist[9] = 790
pred[9] = 6
V - S: [5, 7, 8, 9]
   v pred dist
   5   4      630
   7   3      680
   8   0  Infinity
   9   6      790
dist[5] is minDist = 630
V - S: [7, 8, 9]
   v pred dist
   7   3      680
   8   0  Infinity
   9   6      790
dist[7] is minDist = 680
dist[8] = 800
pred[8] = 7
V - S: [8, 9]
   v pred dist
   8   7      800
   9   6      790
dist[9] is minDist = 790
V - S: [8]
   v pred dist
   8   7      800
dist[8] is minDist = 800
V - S: []
   v pred dist
   v      dist path
   1      320  0 --> 1
   2      450  0 --> 1 --> 2
   3      500  0 --> 1 --> 3
```

4 570 0 --> 1 --> 2 --> 4
5 630 0 --> 1 --> 2 --> 4 --> 5
6 610 0 --> 1 --> 2 --> 4 --> 6
7 680 0 --> 1 --> 3 --> 7
8 800 0 --> 1 --> 3 --> 7 --> 8
9 790 0 --> 1 --> 2 --> 4 --> 6 --> 9

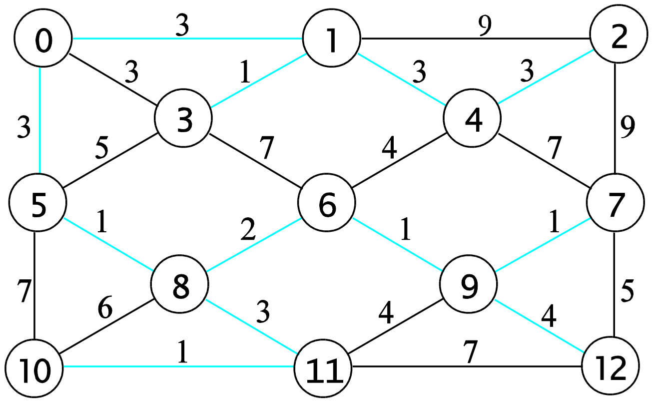


10.6.3

Trace the execution of Prim's algorithm to find the minimum spanning tree for the graph shown in Exercise 2.

V - S: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Priority Queue: [[(0, 1): 3.0], [(0, 3): 3.0], [(0, 5): 3.0]]
Shortest Edge remaining: [(0, 1): 3.0]
V - S: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Priority Queue: [[(1, 3): 1.0], [(0, 5): 3.0], [(1, 2): 9.0], [(0, 3): 3.0], [(1, 4): 3.0]]
Shortest Edge remaining: [(1, 3): 1.0]
V - S: [2, 4, 5, 6, 7, 8, 9, 10, 11, 12]
Priority Queue: [[(1, 4): 3.0], [(0, 5): 3.0], [(3, 6): 7.0], [(0, 3): 3.0], [(3, 5): 5.0], [(1, 2): 9.0]]
Shortest Edge remaining: [(1, 4): 3.0]
V - S: [2, 5, 6, 7, 8, 9, 10, 11, 12]
Priority Queue: [[(0, 5): 3.0], [(0, 3): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(3, 5): 5.0], [(3, 6): 7.0], [(4, 7): 7.0], [(1, 2): 9.0]]
Shortest Edge remaining: [(0, 5): 3.0]
V - S: [2, 6, 7, 8, 9, 10, 11, 12]
Priority Queue: [[(5, 8): 1.0], [(0, 3): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(3, 5): 5.0], [(3, 6): 7.0], [(4, 7): 7.0], [(1, 2): 9.0], [(5, 10): 7.0]]

Shortest Edge remaining: [(5, 8): 1.0]
V - S: [2, 6, 7, 9, 10, 11, 12]
Priority Queue: [[(8, 6): 2.0], [(0, 3): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(8, 11): 3.0], [(3, 6): 7.0], [(4, 7): 7.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0]]
Shortest Edge remaining: [(8, 6): 2.0]
V - S: [2, 7, 9, 10, 11, 12]
Priority Queue: [[(6, 9): 1.0], [(0, 3): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(8, 11): 3.0], [(3, 6): 7.0], [(4, 7): 7.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0]]
Shortest Edge remaining: [(6, 9): 1.0]
V - S: [2, 7, 10, 11, 12]
Priority Queue: [[(9, 7): 1.0], [(0, 3): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(8, 11): 3.0], [(9, 11): 4.0], [(4, 7): 7.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0], [(3, 6): 7.0], [(9, 12): 4.0]]
Shortest Edge remaining: [(9, 7): 1.0]
V - S: [2, 10, 11, 12]
Priority Queue: [[(0, 3): 3.0], [(8, 11): 3.0], [(4, 2): 3.0], [(4, 6): 4.0], [(9, 12): 4.0], [(9, 11): 4.0], [(7, 12): 5.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0], [(3, 6): 7.0], [(7, 2): 9.0], [(4, 7): 7.0]]
Shortest Edge remaining: [(8, 11): 3.0]
V - S: [2, 10, 12]
Priority Queue: [[(11, 10): 1.0], [(4, 6): 4.0], [(4, 2): 3.0], [(4, 7): 7.0], [(9, 12): 4.0], [(9, 11): 4.0], [(7, 12): 5.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0], [(7, 2): 9.0], [(3, 6): 7.0], [(11, 12): 7.0]]
Shortest Edge remaining: [(11, 10): 1.0]
V - S: [2, 12]
Priority Queue: [[(4, 2): 3.0], [(4, 6): 4.0], [(9, 11): 4.0], [(4, 7): 7.0], [(9, 12): 4.0], [(11, 12): 7.0], [(7, 12): 5.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 5): 5.0], [(7, 2): 9.0], [(3, 6): 7.0]]
Shortest Edge remaining: [(4, 2): 3.0]
V - S: [12]
Priority Queue: [[(4, 6): 4.0], [(9, 12): 4.0], [(9, 11): 4.0], [(4, 7): 7.0], [(3, 5): 5.0], [(11, 12): 7.0], [(7, 12): 5.0], [(1, 2): 9.0], [(5, 10): 7.0], [(8, 10): 6.0], [(3, 6): 7.0], [(7, 2): 9.0]]
Shortest Edge remaining: [(9, 12): 4.0]
Resulting minimum spanning tree: [[(0, 1): 3.0], [(1, 3): 1.0], [(1, 4): 3.0], [(0, 5): 3.0], [(5, 8): 1.0], [(8, 6): 2.0], [(6, 9): 1.0], [(9, 7): 1.0], [(8, 11): 3.0], [(11, 10): 1.0], [(4, 2): 3.0], [(9, 12): 4.0]]


0
1
2
3
4
5
6
7
8
9
10
11
12
3
9
3
1
3
3
3
9
5
7
4
7
1
2
1
1
7
6
3
4
4
5
1
7