Graphs

Chapter 28

Contents

- Some Examples and Terminology
  - Road Maps
  - Airline Routes
  - Mazes
  - Course Prerequisites
- Trees
  - Traversals
  - Breadth-First Traversal
  - Depth-First Traversal

Contents

- Topological Order
- Paths
  - Finding a Path
  - The Shortest Path in an Unweighted Graph
  - The Shortest Path in a Weighted Graph
- Java Interfaces for the ADT Graph

Objectives

- Describe characteristics of a graph, including vertices, edges, paths
- Give examples of graphs, undirected, directed, unweighted, weighted
- Give examples of vertices
  - Adjacent and not adjacent
  - For both directed and undirected graphs

Objectives

- Give examples of paths, simple paths, cycles, simple cycles
- Give examples of connected graphs, disconnected graphs, complete graphs
- Perform depth-first traversal, breadth-first traversal on a given graph
- List topological order for vertices of a directed graph without cycles

Objectives

- Detect whether path exists between two given vertices of a graph
- Find path with fewest edges that joins one vertex to another
- Find path with lowest cost that joins one vertex to another in a weighted graph
- Describe operations for ADT graph

Examples

- *Not* the graphs we study
  - Bar graphs, pie charts, etc.
- Graphs we study include
  - Trees
  - Networks of nodes connected by paths
  - Could include a road map

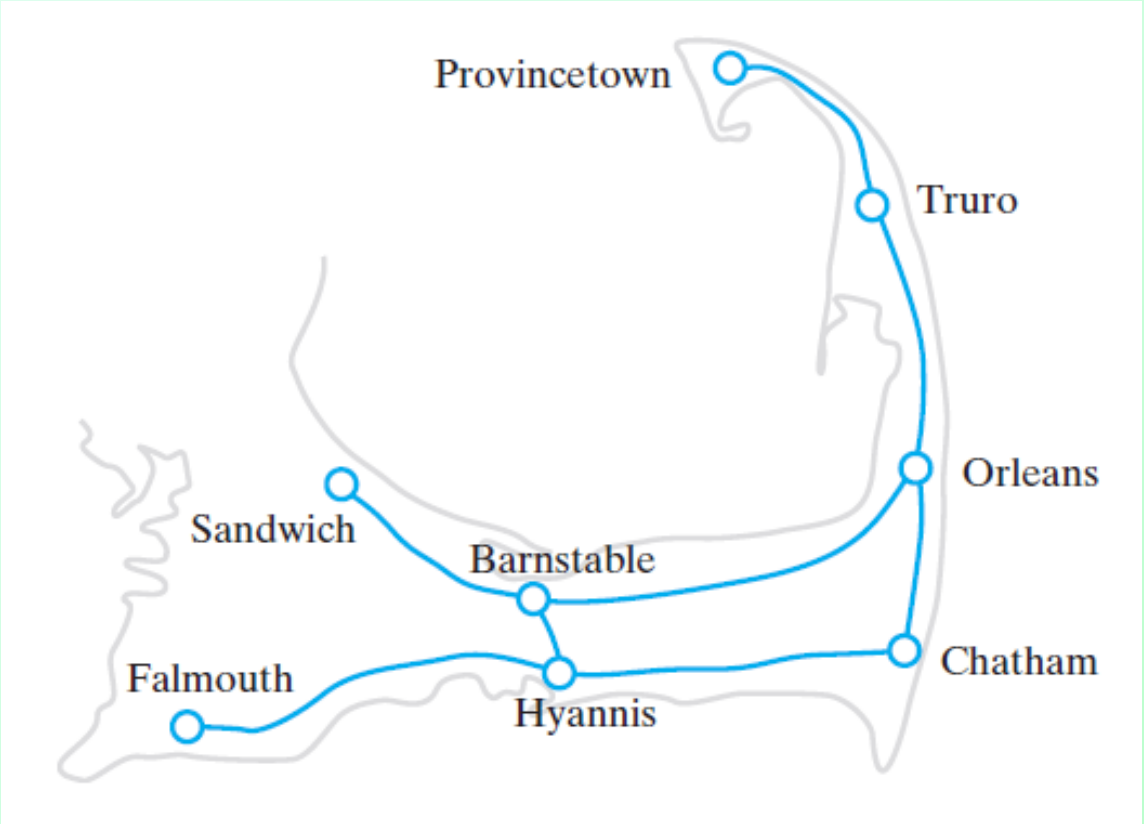


Figure 28-1 A portion of a road map

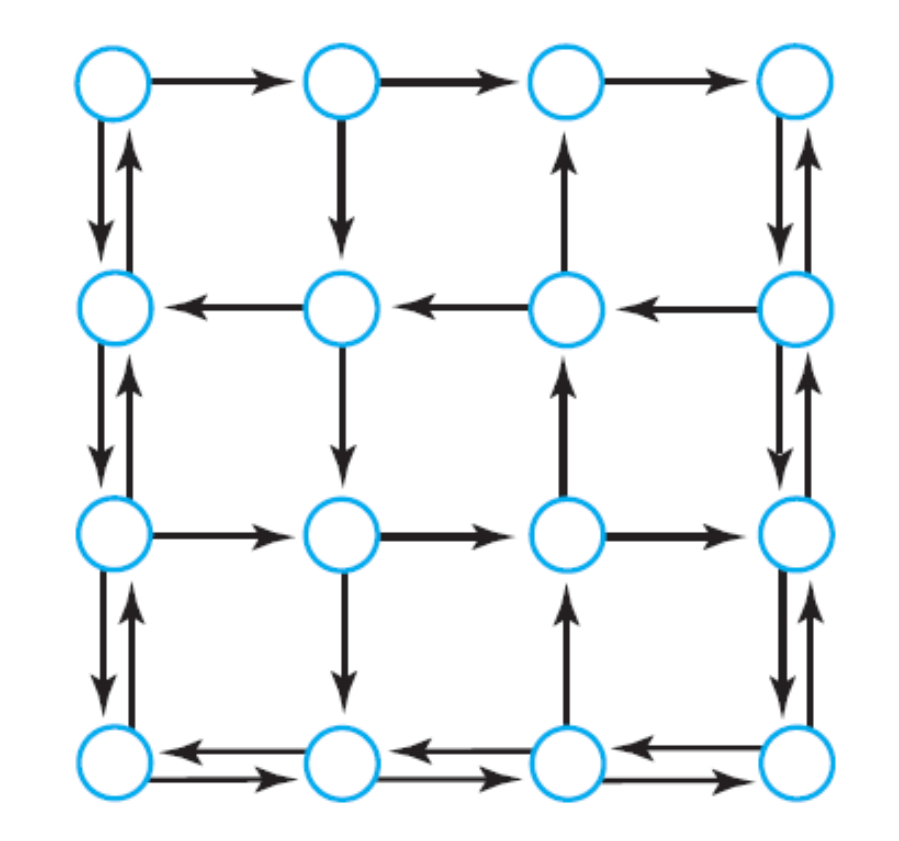


Figure 28-2 A directed graph representing a portion of a city's street map

Terminology

- Nodes connected by edges
- Edges
  - Undirected
  - Directed (digraph)
- Path between two vertices
  - Sequence of edges
- Weights
  - Shortest, fastest, cheapest, costs

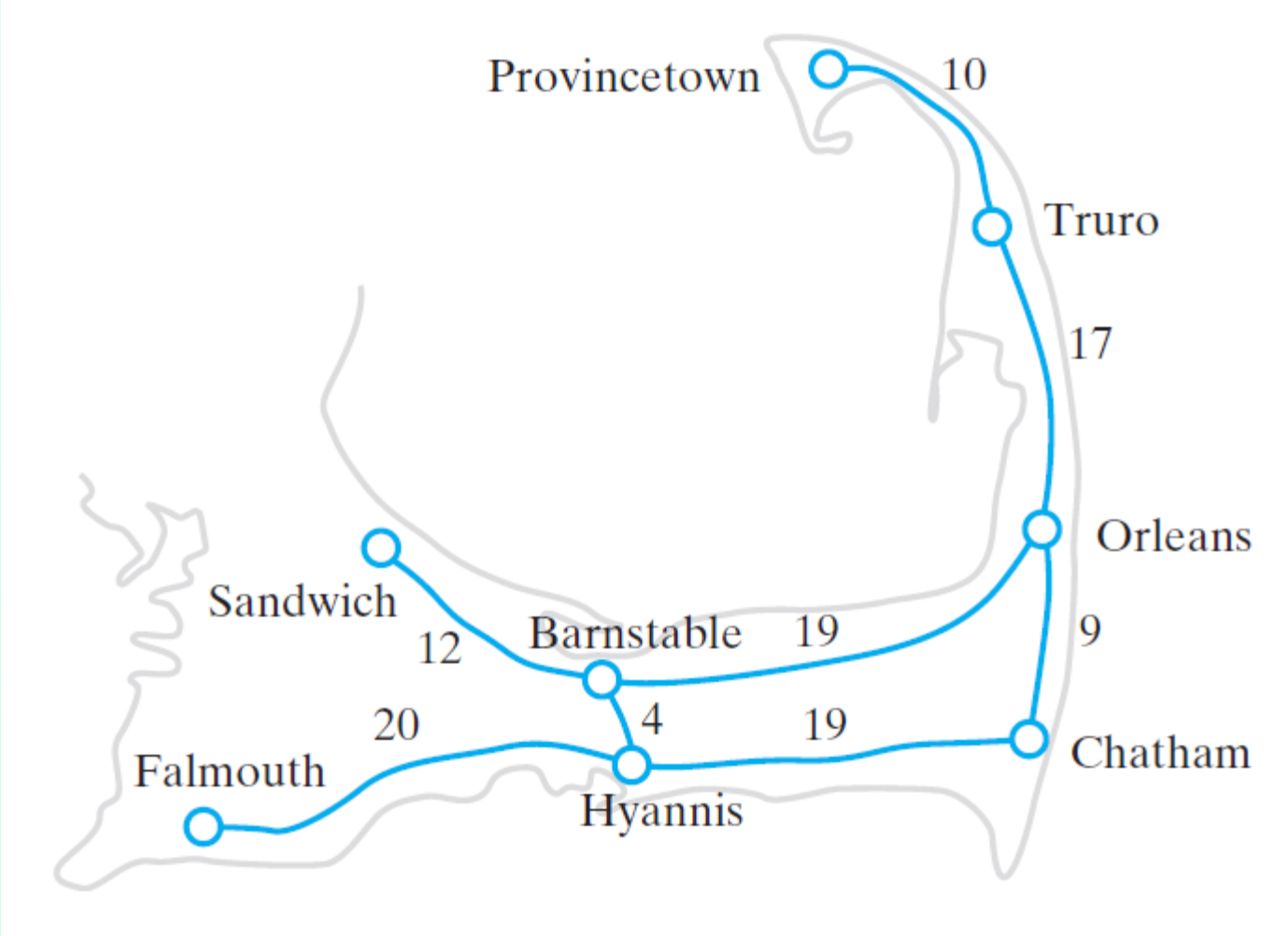


Figure 28-3 A weighted graph

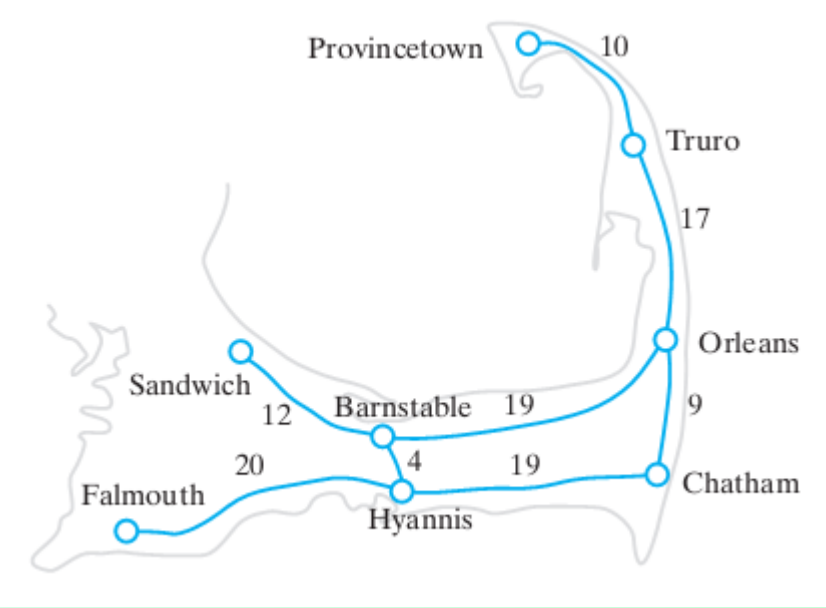
Question 1 Consider the graph in Figure 28-3.

a. What is the length of the path that begins in Provincetown, passes through Truro and Orleans, and ends in Chatham?

b. What is the weight of the path just described?

c. Consider all paths from Truro to Sandwich that do not have cycles. Which path has the shortest length?

d. Of the paths you considered in Part c , which one has the smallest weight?



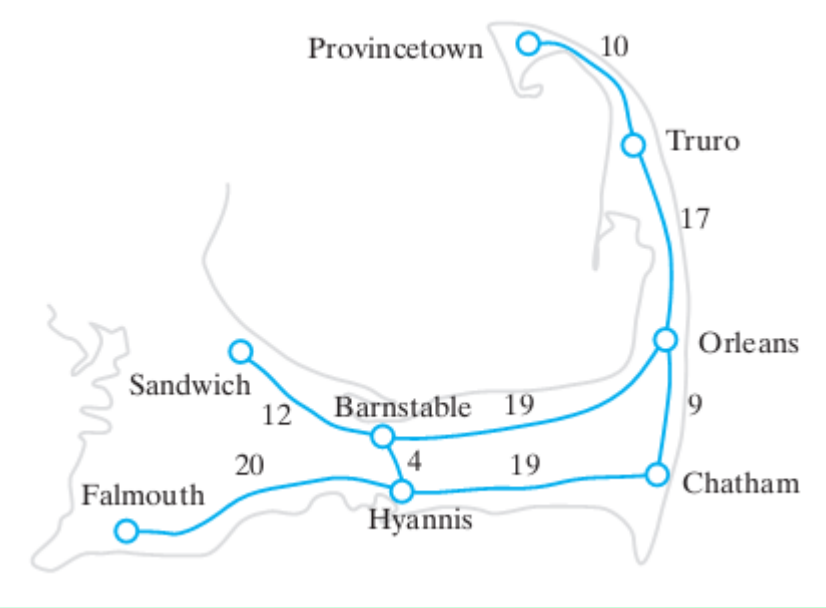
Question 1 Consider the graph in Figure 28-3.

a. What is the length of the path that begins in Provincetown, passes through Truro and Orleans, and ends in Chatham?

b. What is the weight of the path just described?

c. Consider all paths from Truro to Sandwich that do not have cycles. Which path has the shortest length?

d. Of the paths you considered in Part c , which one has the smallest weight?



a. 3.

b. 36.

c. Truro-Orleans-Barnstable-Sandwich, with a length of 3.

d. Truro-Orleans-Barnstable-Sandwich, with a weight of 48.

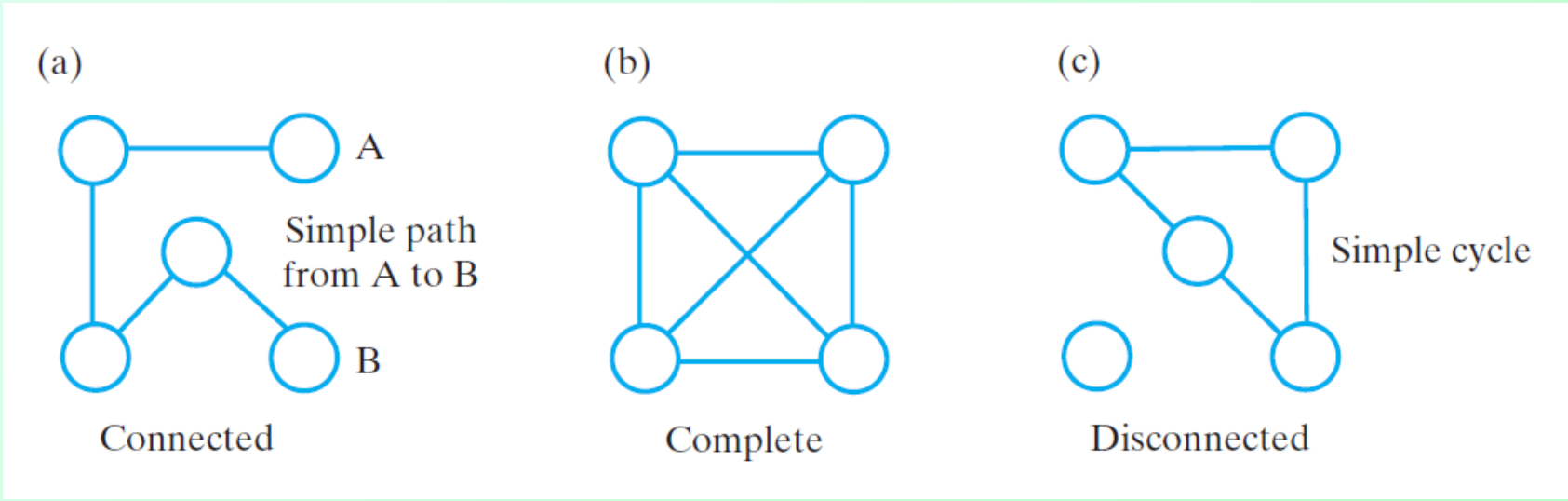


Figure 28-4 Undirected graphs

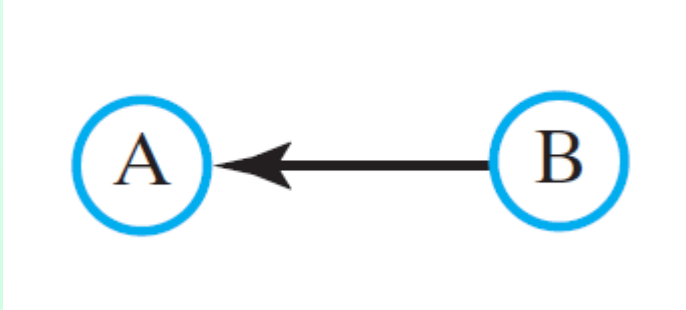


Figure 28-5 Vertex A is adjacent to vertex B, but B is not adjacent to A

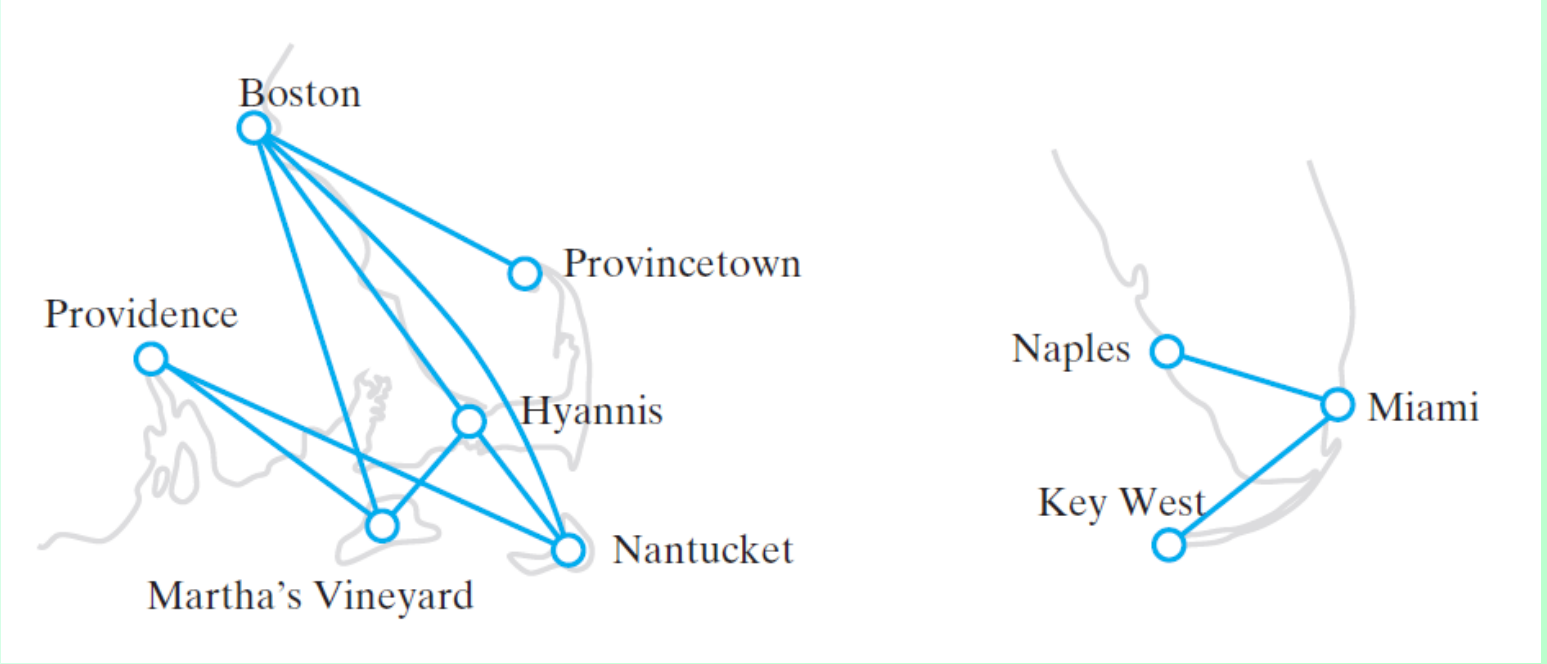


Figure 28-6 Airline routes

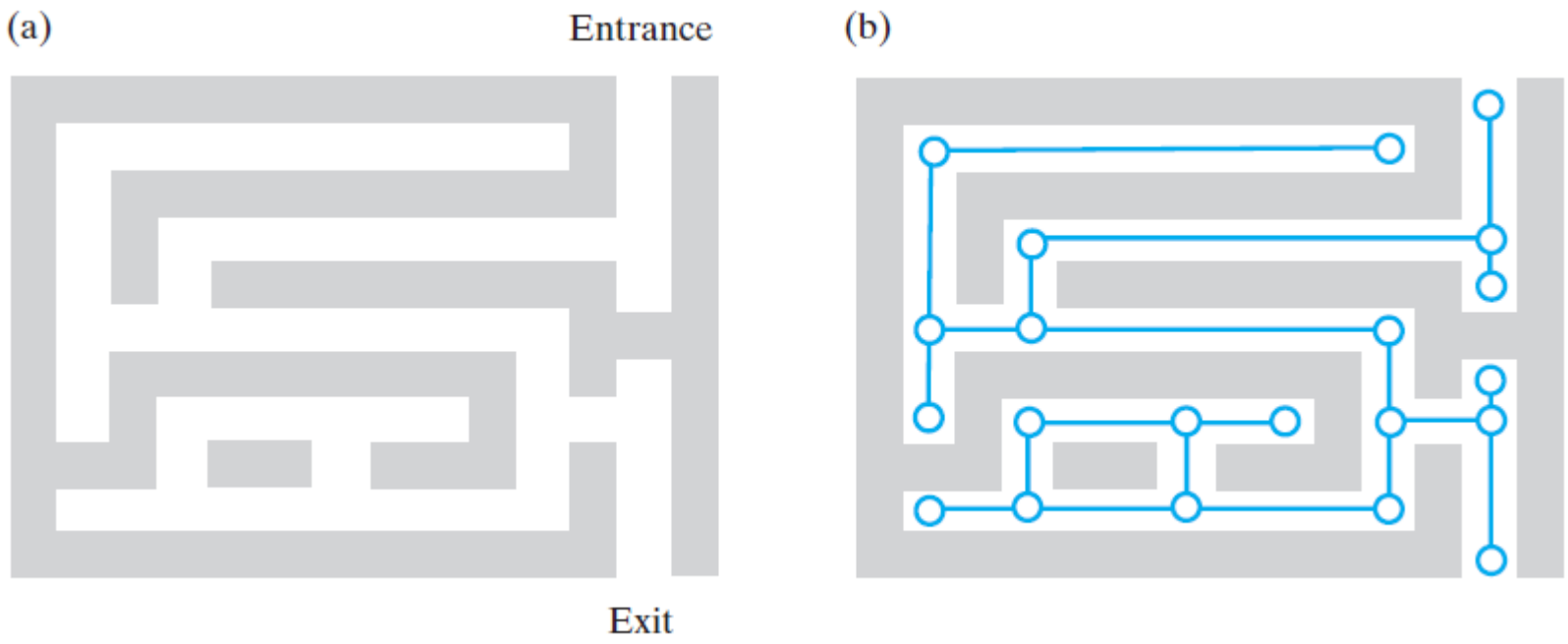


Figure 28-7 (a) A maze; (b) its representation as a graph

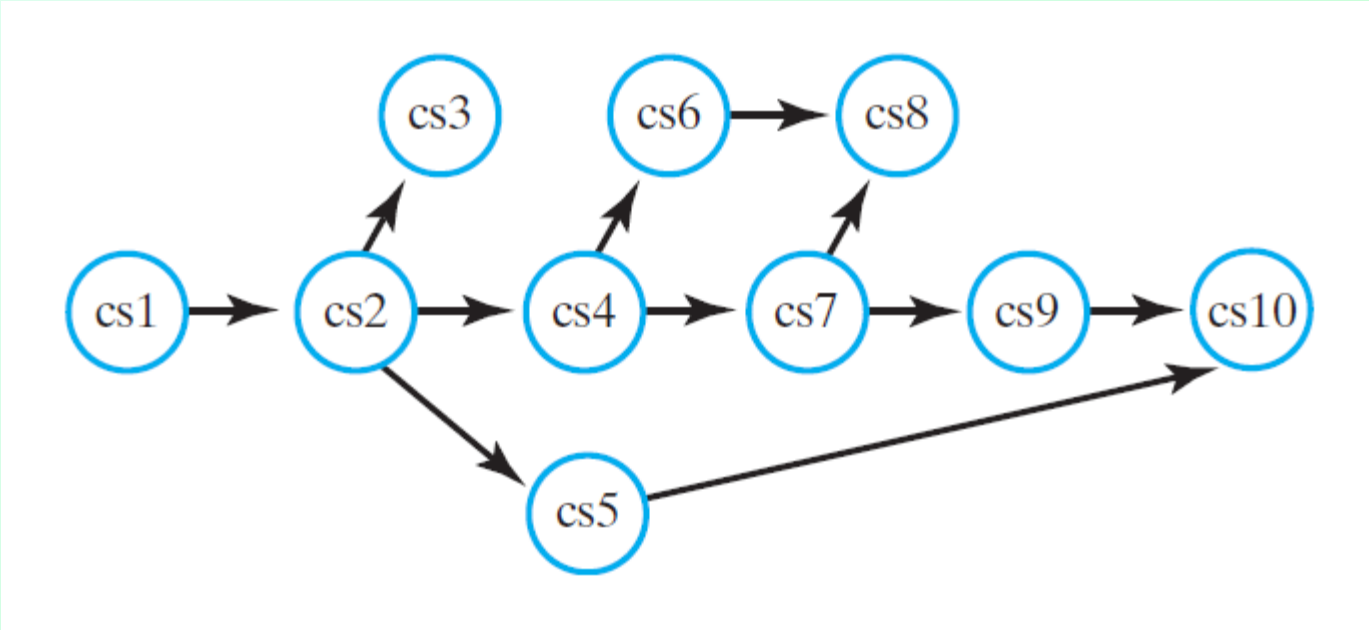
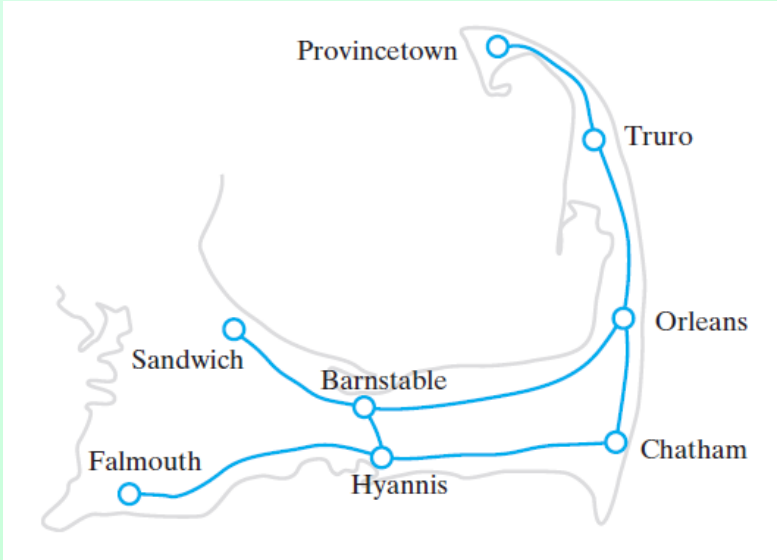


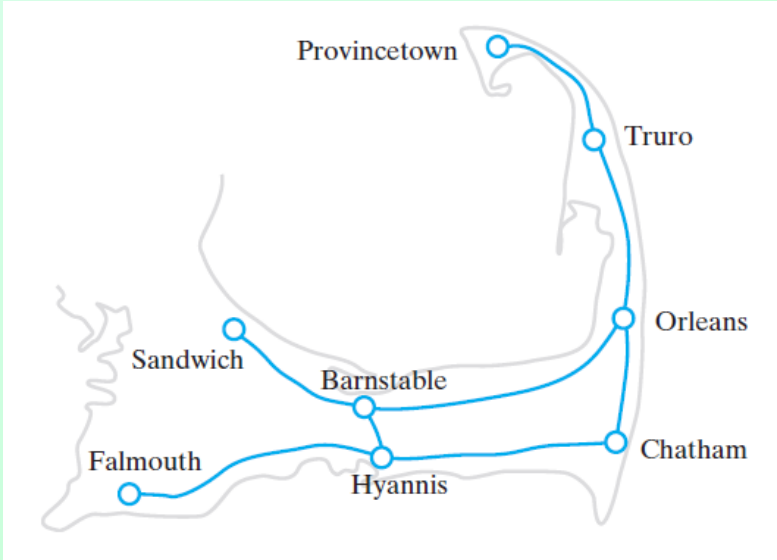
Figure 28-8 The prerequisite structure for a selection of courses as a directed graph without cycles

Question 2 What physical systems in a typical house could you represent as a graph?

Question 3 Is the graph in Figure 28-1 connected? Is it complete?



Question 2 What physical systems in a typical house could you represent as a graph?
Question 3 Is the graph in Figure 28-1 connected? Is it complete?



2. Electrical (or telephone or TV) wires, plumbing, hallways or other connections between rooms.

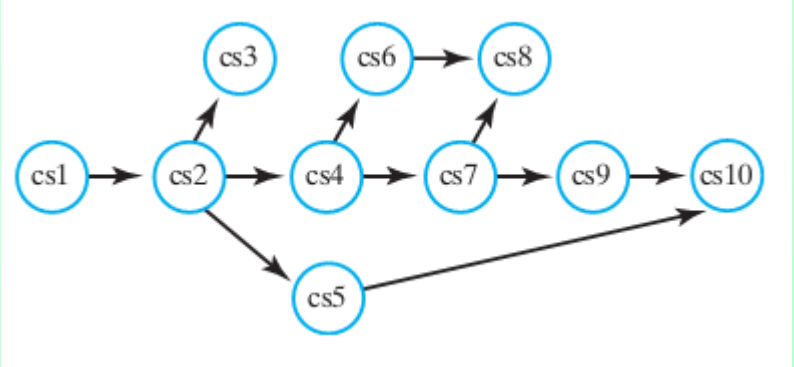
3. The graph is connected but not complete.

Question 4 Is the graph in Figure 28-8 a tree? Explain.

Question 5 For the graph in Figure 28-8,

a. Is cs1 adjacent to cs2? c. Is cs1 adjacent to cs4?

b. Is cs2 adjacent to cs1? d. Is cs4 adjacent to cs1?

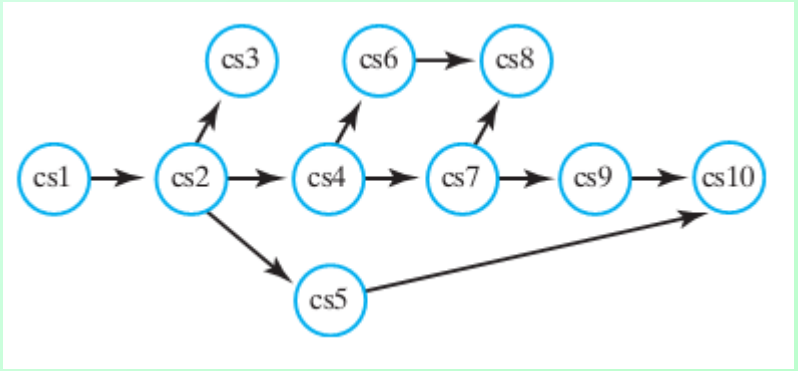


Question 4 Is the graph in Figure 28-8 a tree? Explain.

Question 5 For the graph in Figure 28-8,

a. Is cs1 adjacent to cs2? c. Is cs1 adjacent to cs4?

b. Is cs2 adjacent to cs1? d. Is cs4 adjacent to cs1?



4. No; cs8 and cs10 each would have 2 parents.

5. a. No; b. Yes c. No d. No.

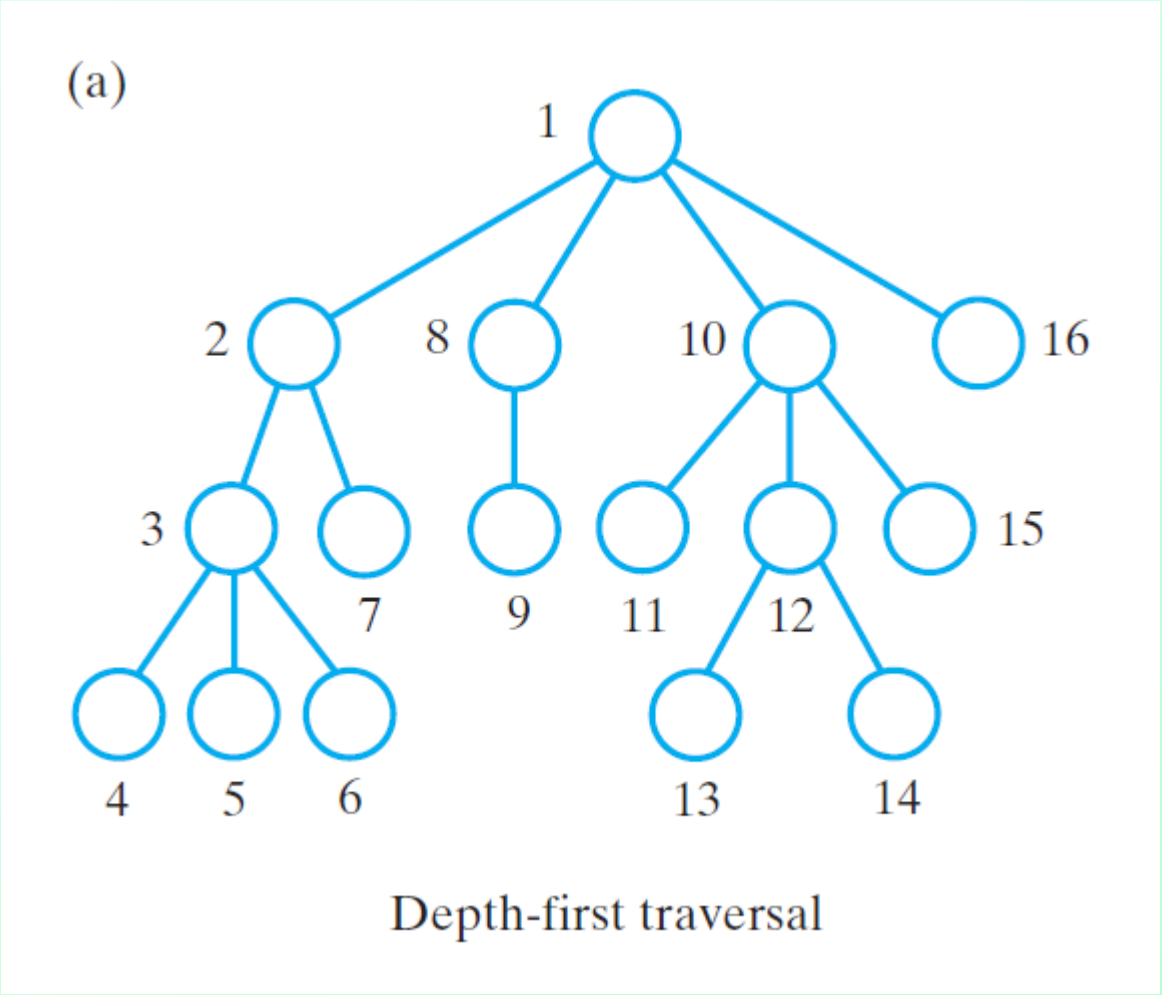


Figure 28-9 The visitation order of two traversals: (a) depth first;

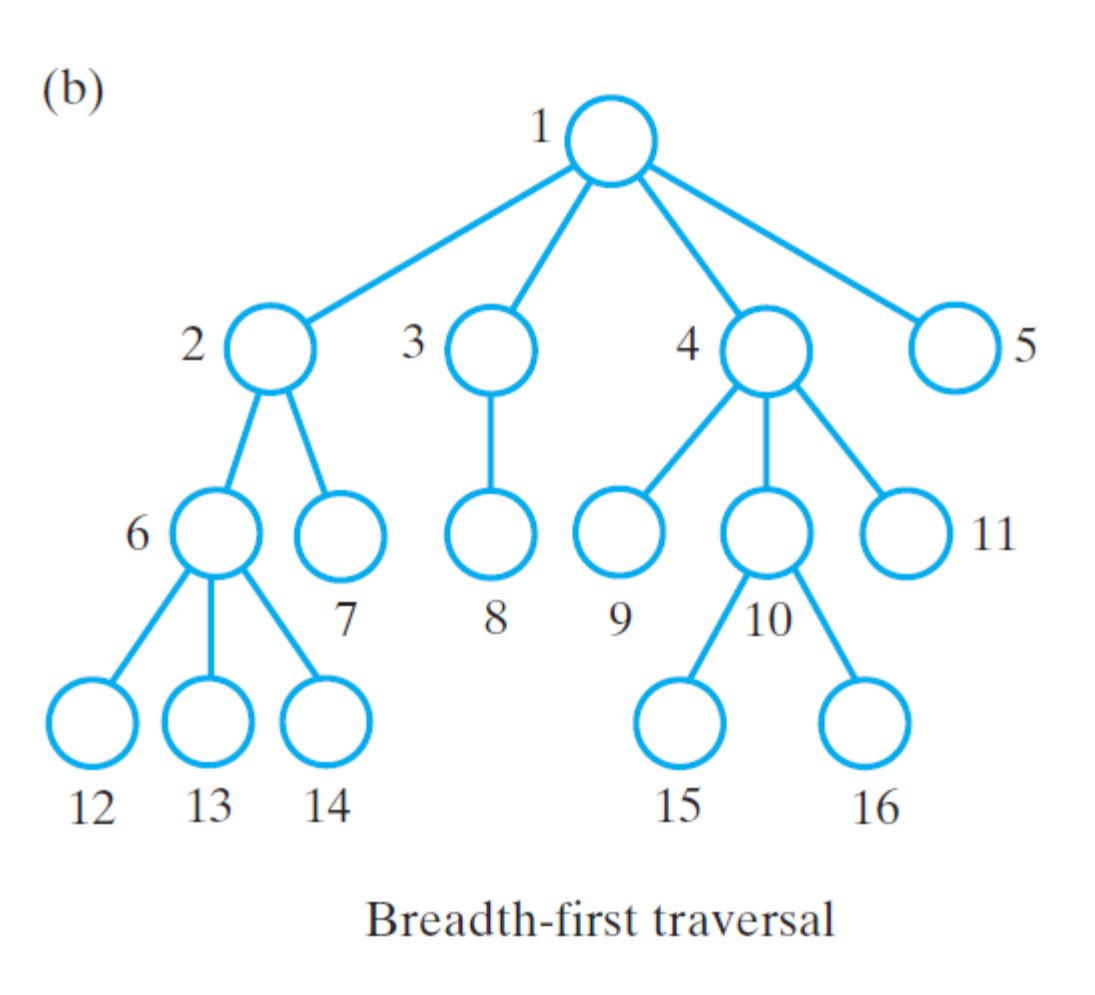
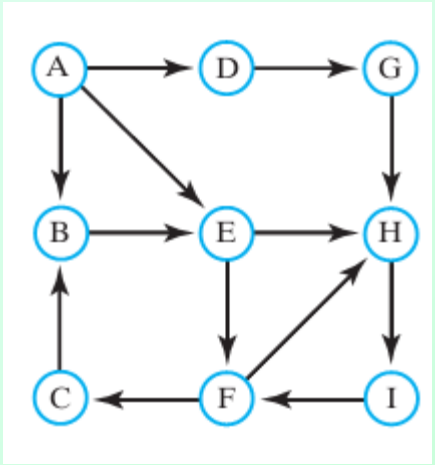
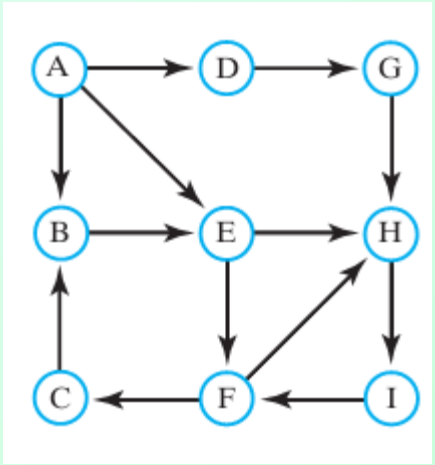


Figure 28-9 The visitation order of two traversals: (b) breadth first

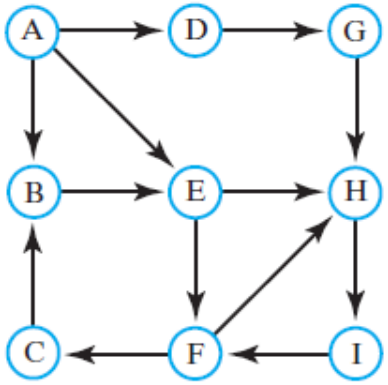
Question 6 In what order does a breadth -first traversal visit the vertices in the graph shown in Figure 28-10 when you begin at vertex E and visit neighbors in alphabetic order?



Question 6 In what order does a breadth -first traversal visit the vertices in the graph shown in Figure 28-10 when you begin at vertex E and visit neighbors in alphabetic order?

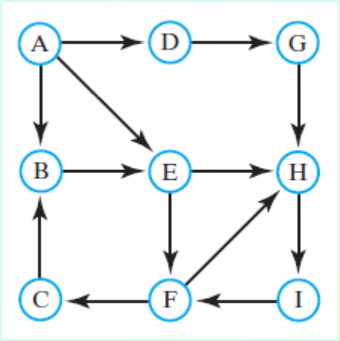


E , F, H , C , I, B.



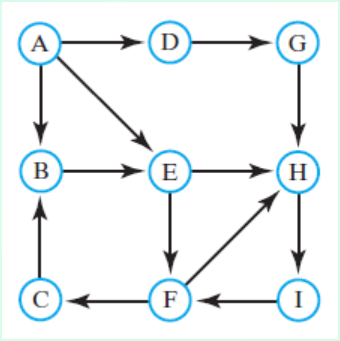
frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI
C			I	
I			*empty*	

FIGURE 28-10 A trace of a breadth-first traversal beginning at vertex A of a directed graph



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI

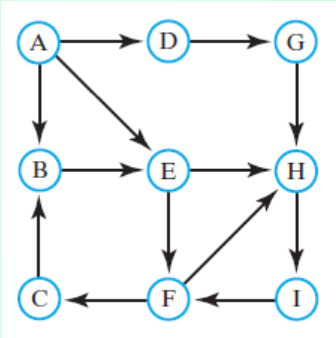
Figure 28-11 A trace of a depth-first traversal beginning at vertex A of a directed graph



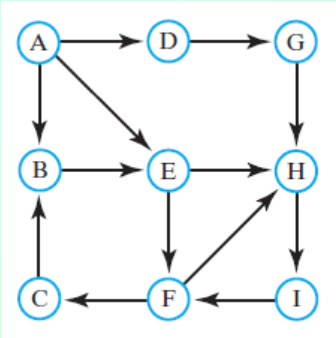
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G		GDA	ABEFCHIDG
G			DA	
D			A	
A			*empty*	ABEFCHIDG

Figure 28-11 A trace of a depth-first traversal beginning at vertex A of a directed graph

Question 7 In what order does a depth-first traversa l visit the vertices in the graph shown in Figure 28-11 when you begin at vertex E and visit neighbors in alphabetic order?



Question 7 In what order does a depth-first traversa l visit the vertices in the graph shown in Figure 28-11 when you begin at vertex E and visit neighbors in alphabetic order?



E , F, C , B , H , I.

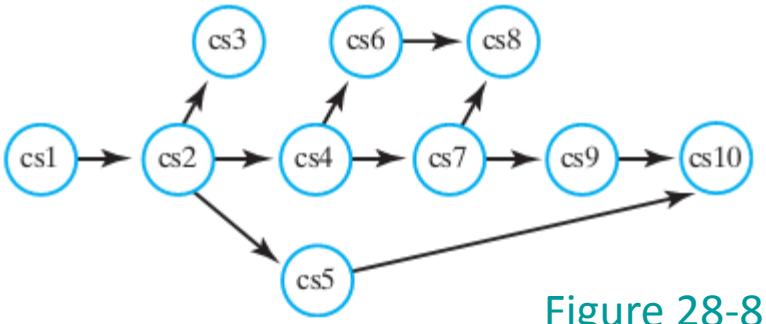


Figure 28-8

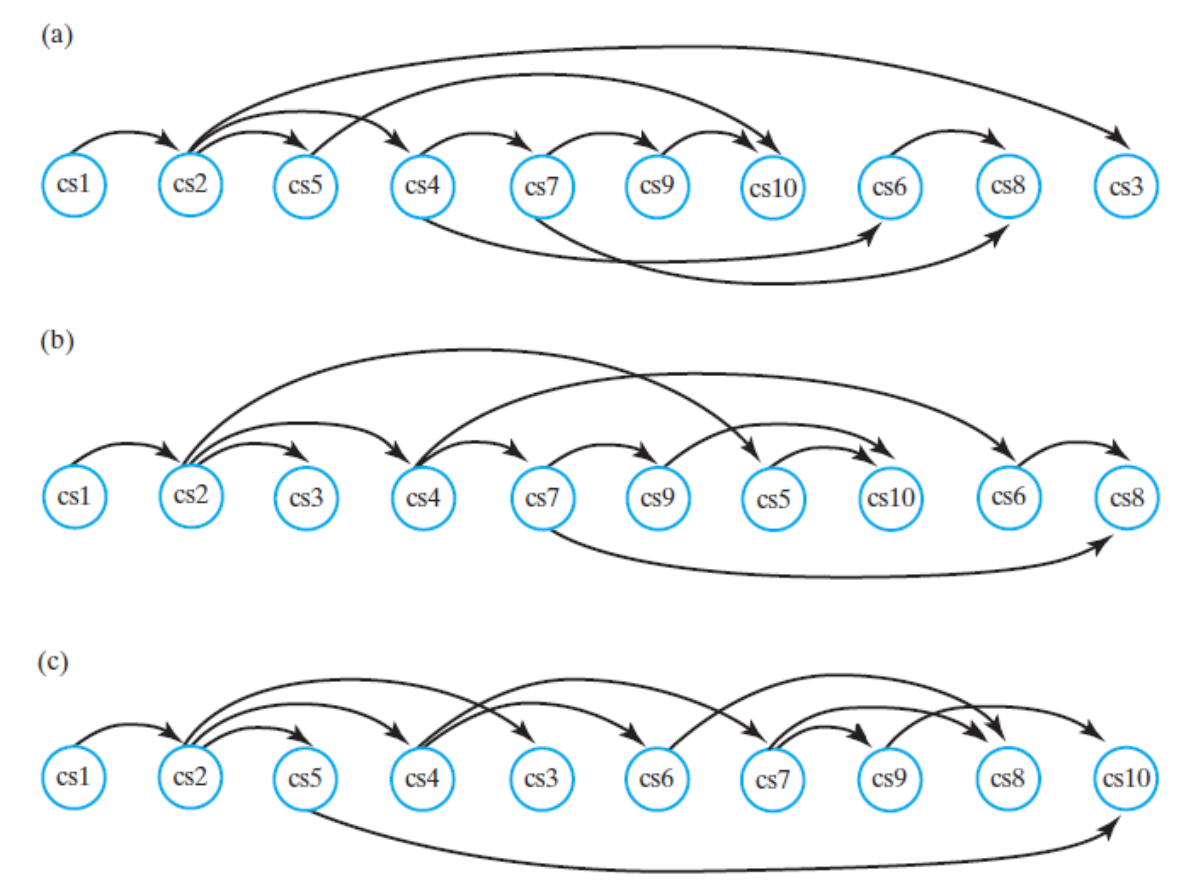


Figure 28-12 Three topological orders for the graph in Figure 28-8

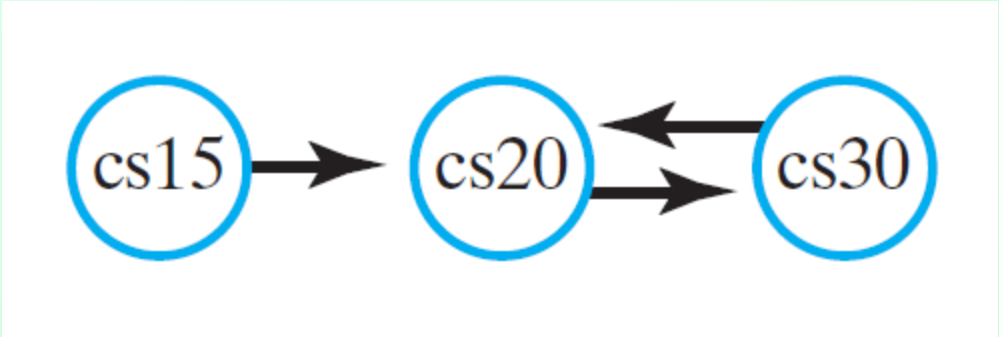
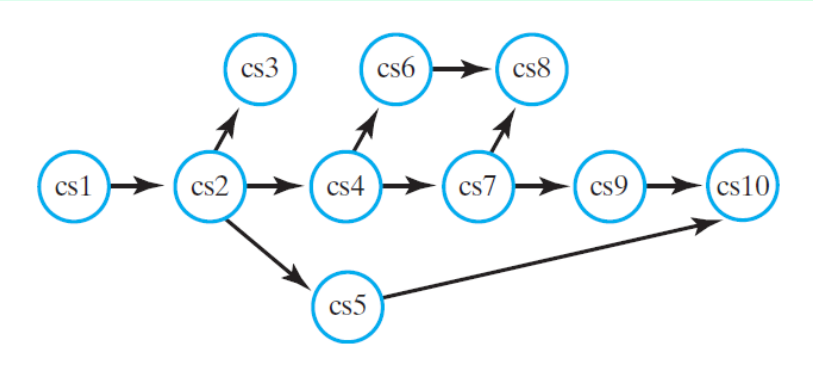
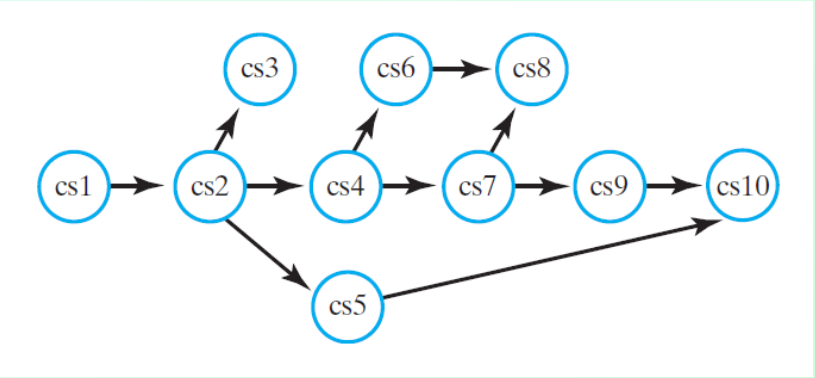


Figure 28-13 An impossible prerequisite structure for three courses, as a directed graph with a cycle

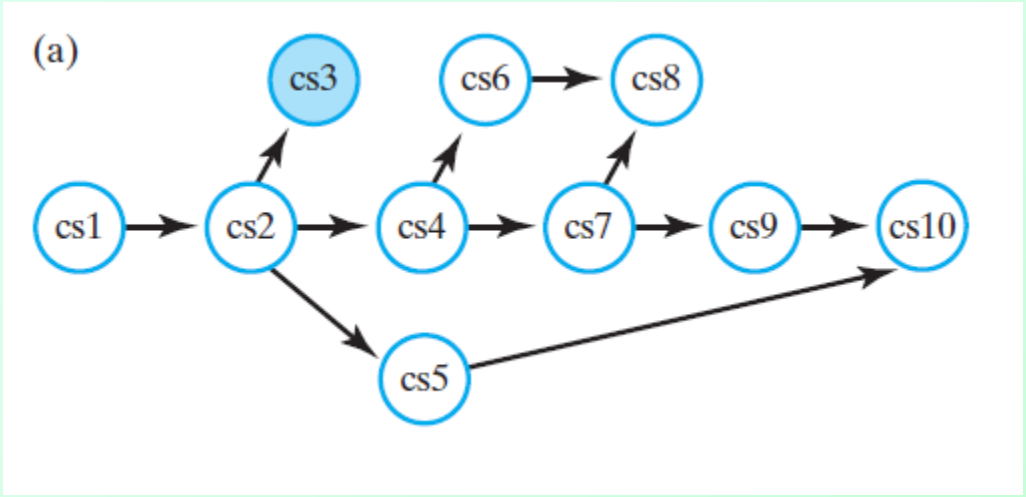
Question 8 What is another topological order of the vertices in the graph in Figure 28-8?



Question 8 What is another topological order of the vertices in the graph in Figure 28-8?



cs1, cs2, cs4, cs7, cs9, cs5, cs10, cs6, cs8, cs3;
or
cs1, cs2, cs3, cs4, cs5, cs6, cs7, cs8, cs9, cs10.



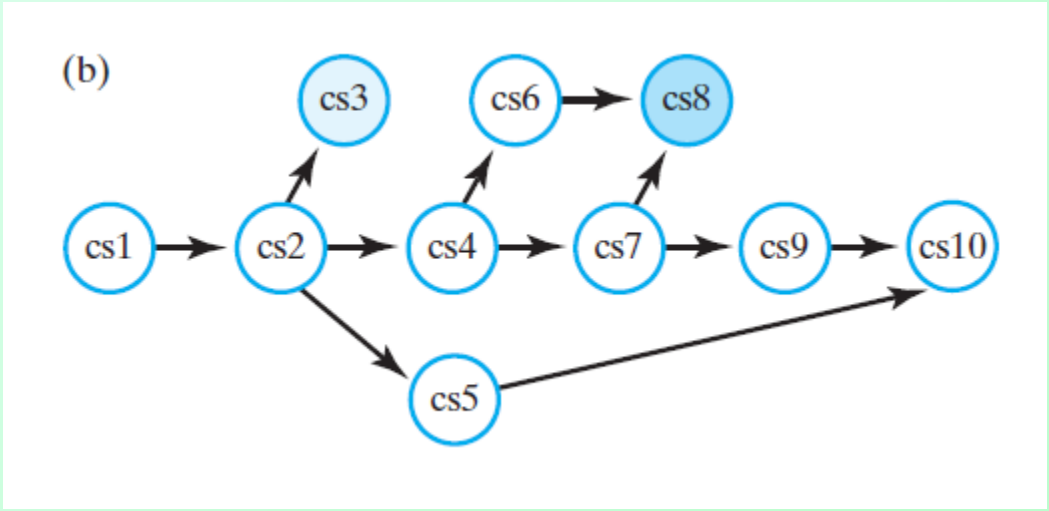


Figure 28-14 Finding a topological order for the graph in Figure 28-8

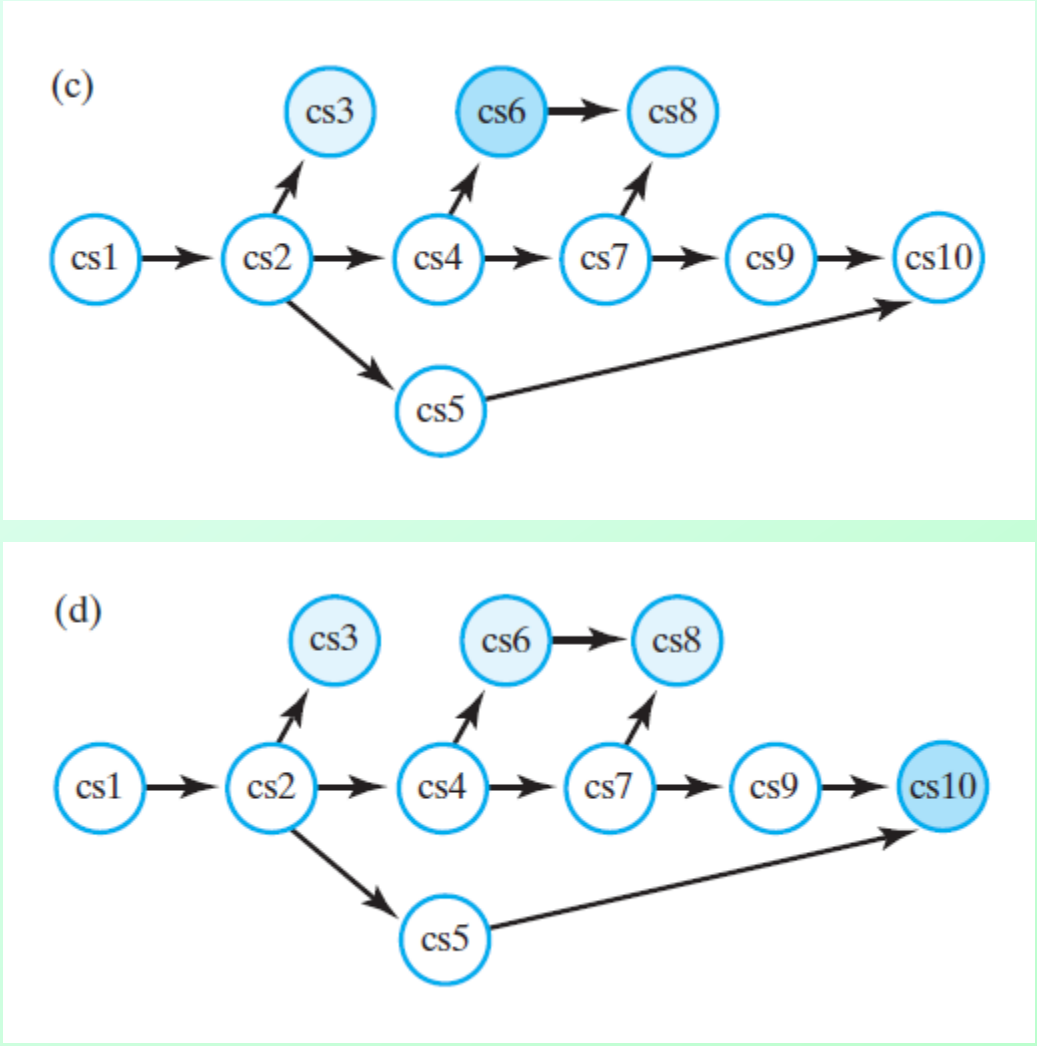


Figure 28-14 Finding a topological order for the graph in Figure 28-8

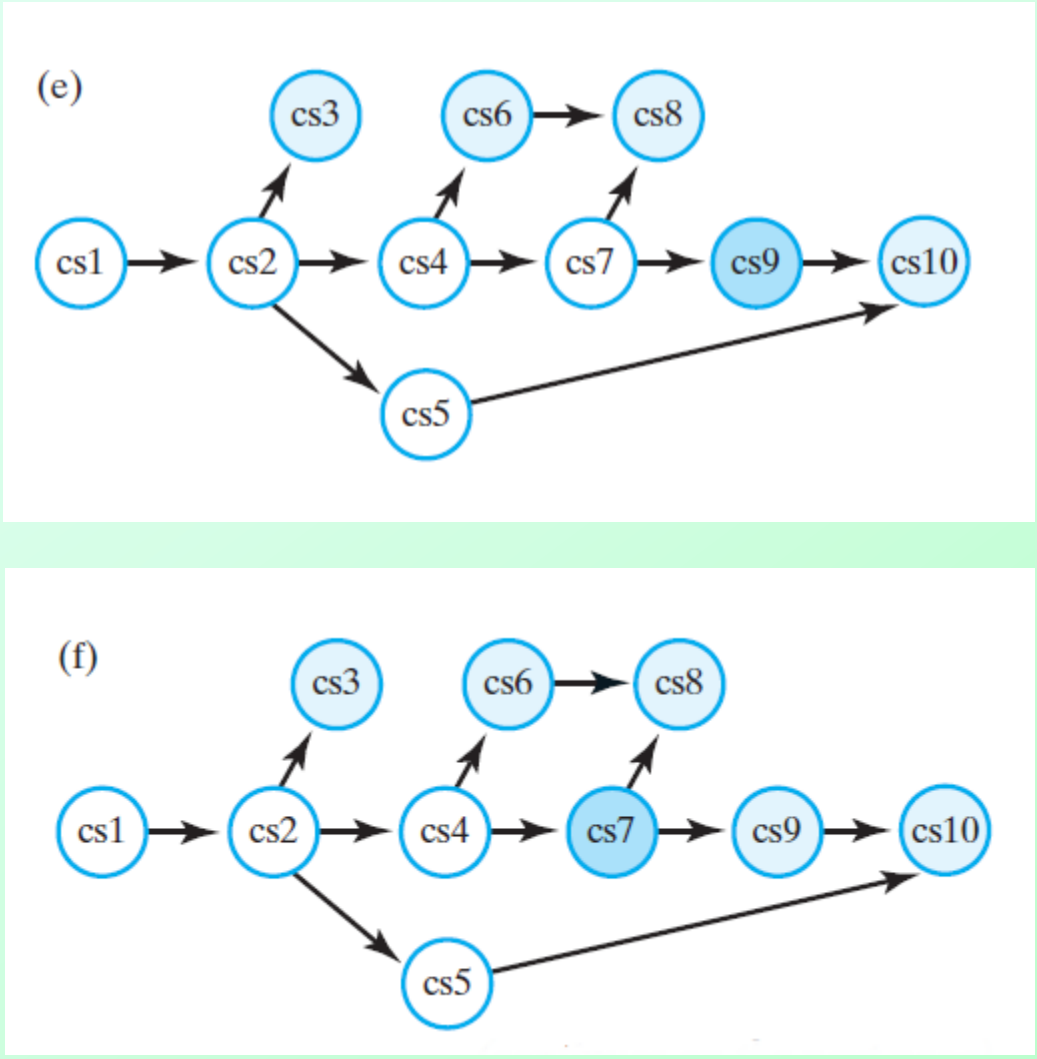


Figure 28-14 Finding a topological order for the graph in Figure 28-8

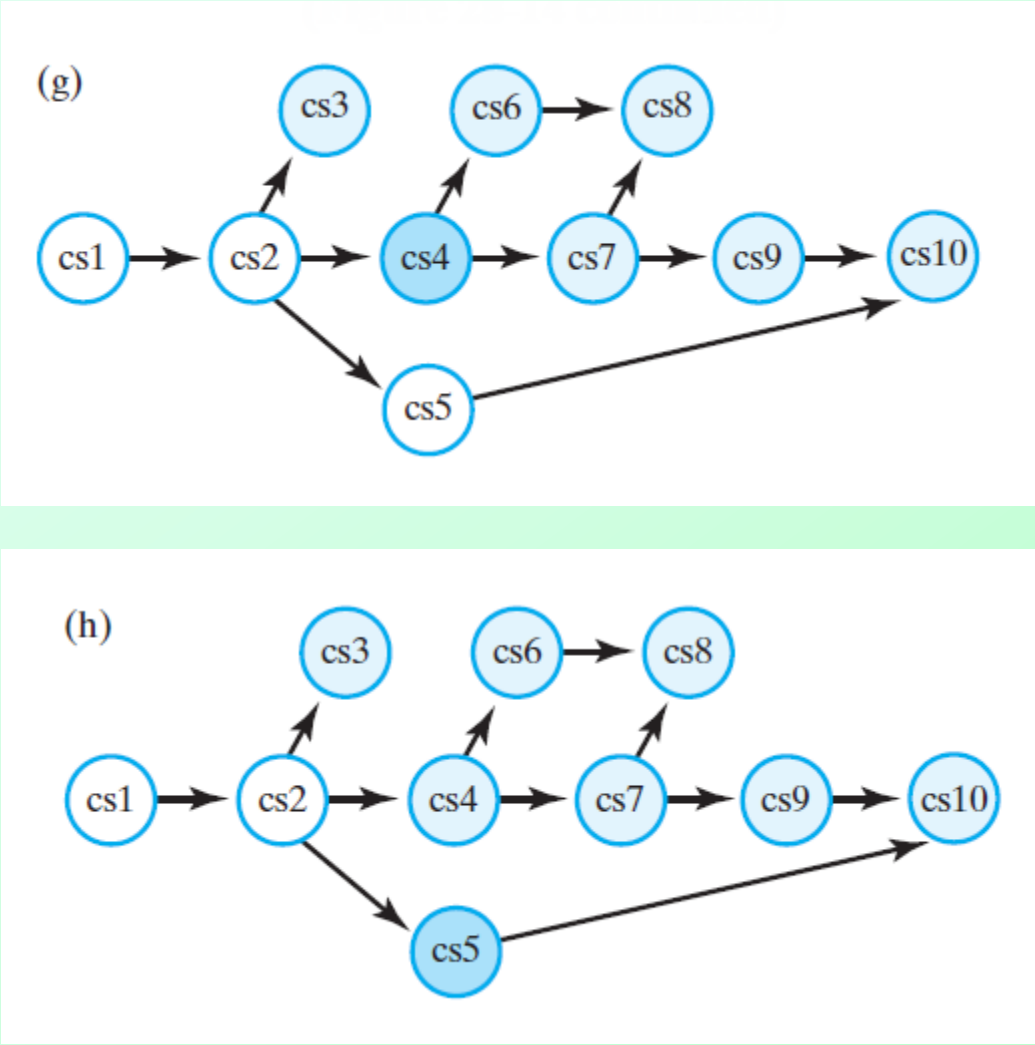


Figure 28-14 Finding a topological order for the graph in Figure 28-8

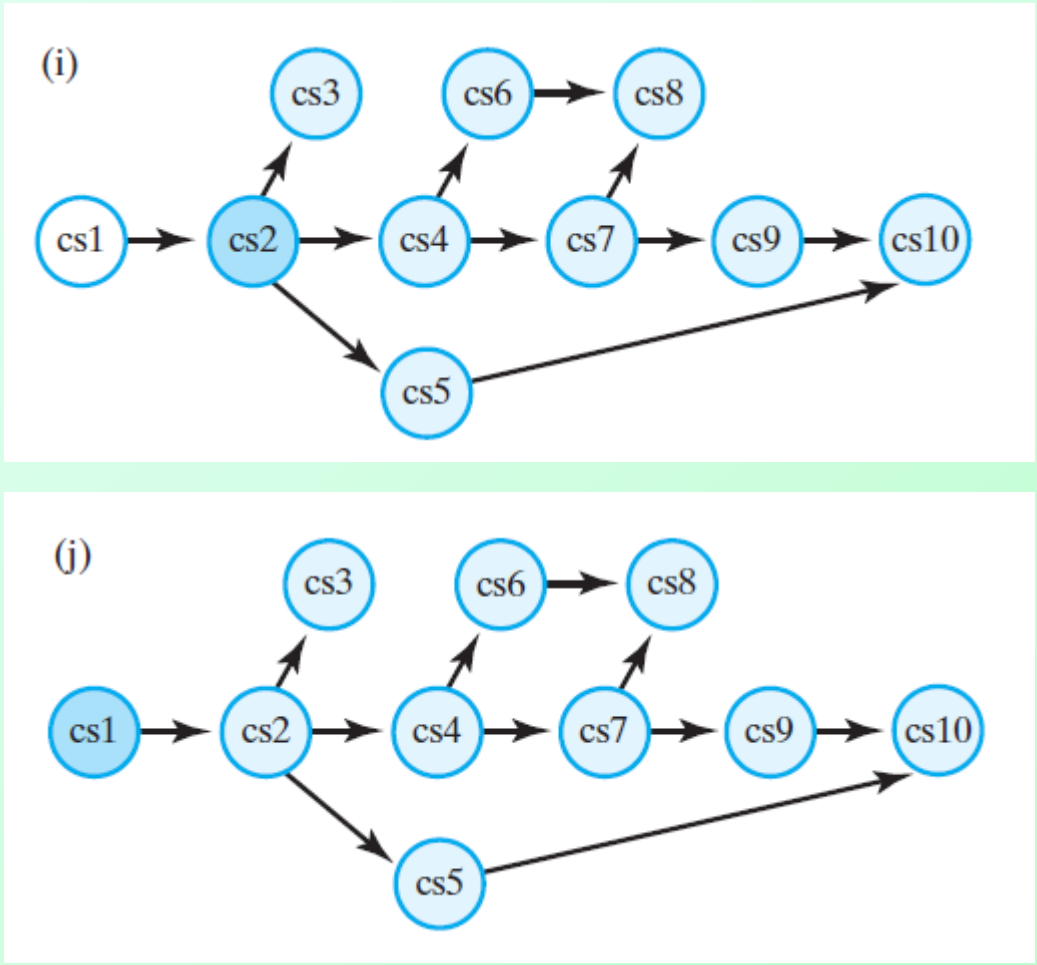
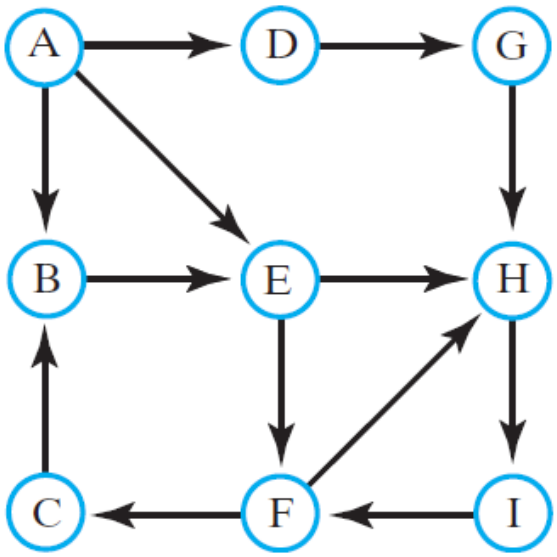


Figure 28-14 Finding a topological order for the graph in Figure 28-8

(a)



(b)

A → B → E → F → H
A → B → E → H
A → D → G → H
A → E → F → H
A → E → H

Figure 28-15 (a) An unweighted graph and
(b) the possible paths from vertex A to vertex H

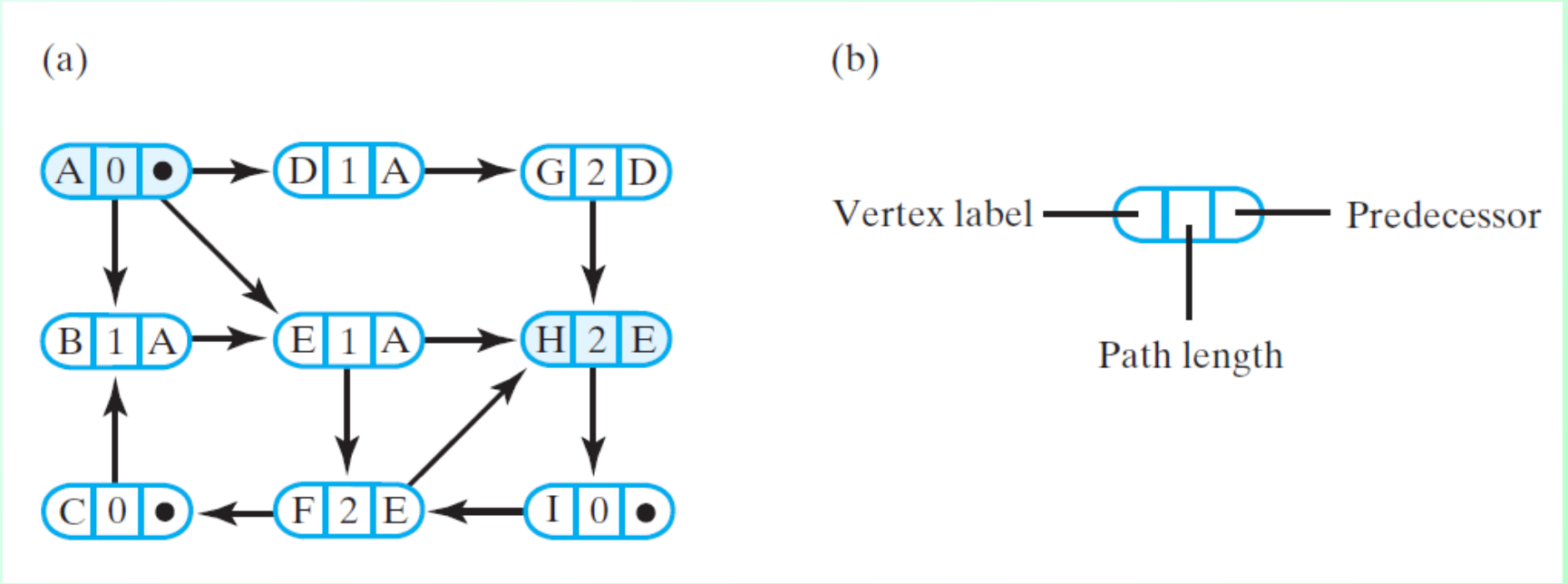
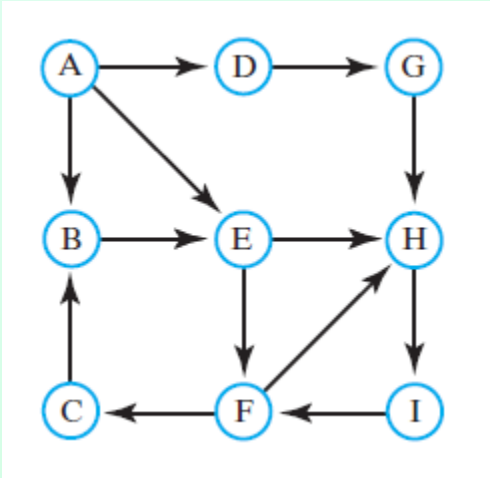
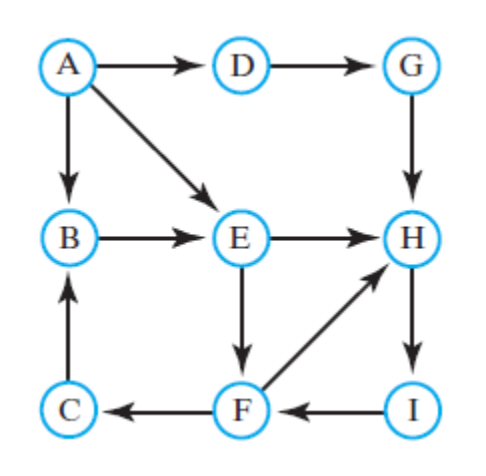


Figure 28-16 (a) The graph in Figure 28-15a after the shortest-path algorithm has traversed from vertex A to vertex H; (b) the data in a vertex



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)
		A	A 0 •
A 0 •			*empty*
	B	B	B 1 A
	D	D	B 1 A, D 1 A
	E	E	B 1 A, D 1 A, E 1 A
B 1 A			D 1 A, E 1 A

Figure 28-17 A trace of the traversal in the algorithm to find the shortest path from vertex A to vertex H in an unweighted graph



D 1 A			E 1 A
	G	G	E 1 A G 2 D
E 1 A			G 2 D
	F	F	G 2 D F 2 E
	H	H	G 2 D F 2 E H 2 E

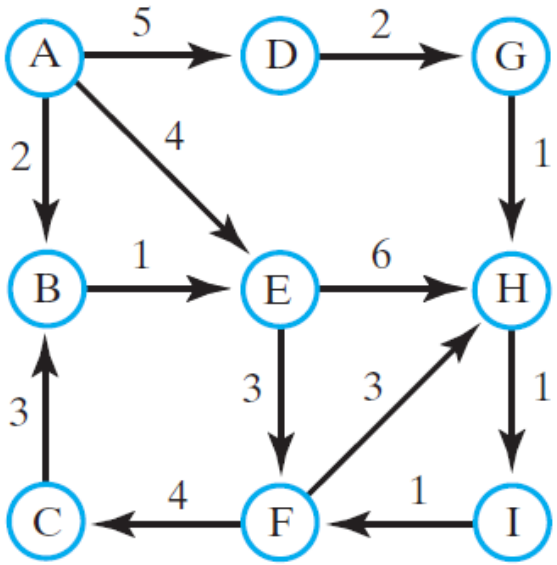
Figure 28-17 A trace of the traversal in the algorithm to find the shortest path from vertex A to vertex H in an unweighted graph

Question 9 Continue the trace begun in Figure 28-17 to find the shortest path from vertex A to vertex C .

Question 9 Continue the trace begun in Figure 28-17 to find the shortest path from vertex A to vertex C .

Remove vertex G from the queue. G 's neighbor H has been visited already. Now remove F from the queue. F 's neighbor C is unvisited. Set C 's predecessor to F and its path-length field to 3 (1 + the length recorded in F). Add C to the queue. Since C is the destination, construct the path by working backward from C, as we did in Segment 28.18. The shortest path is A → E → F → C , with a length of 3.

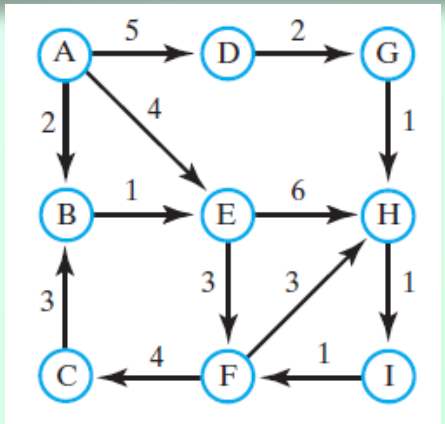
(a)



(b)

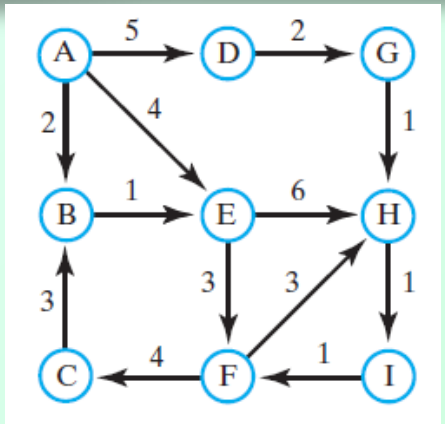
Path	Weight
A → B → E → F → H	9
A → B → E → H	9
A → D → G → H	8
A → E → F → H	10
A → E → H	10

Figure 28-18 (a) A weighted graph and (b) the possible paths from vertex A to vertex H, with their weights



frontVertex	Visited vertex	nextNeighbor	Priority queue (front to back)
			A 0 •
A 0 •	A		*empty*
		B	B 2 A
		D	B 2 A, D 5 A
		E	B 2 A, E 4 A, D 5 A
B 2 A	B		E 4 A, D 5 A
		E	E 3 B, E 4 A, D 5 A
E 3 B	E		E 4 A, D 5 A
		F	E 4 A, D 5 A, F 6 E

Figure 28-19 A trace of the traversal in the algorithm to find the cheapest path from vertex A to vertex H in a weighted graph



		H	E 4 A	D 5 A	F 6 E	H 9 E
			D 5 A	F 6 E	H 9 E	
D 5 A	D		F 6 E	H 9 E		
		G	F 6 E	G 7 D	H 9 E	
F 6 E	F		G 7 D	H 9 E		
		H	G 7 D	H 9 E	H 9 F	
		C	G 7 D	H 9 E	H 9 F	C 10 F
G 7 D	G		H 9 E	H 9 F	C 10 F	
		H	H 8 G	H 9 E	H 9 F	C 10 F
H 8 G	H		H 9 E	H 9 F	C 10 F	

Figure 28-19 A trace of the traversal in the algorithm to find the cheapest path from vertex A to vertex H in a weighted graph

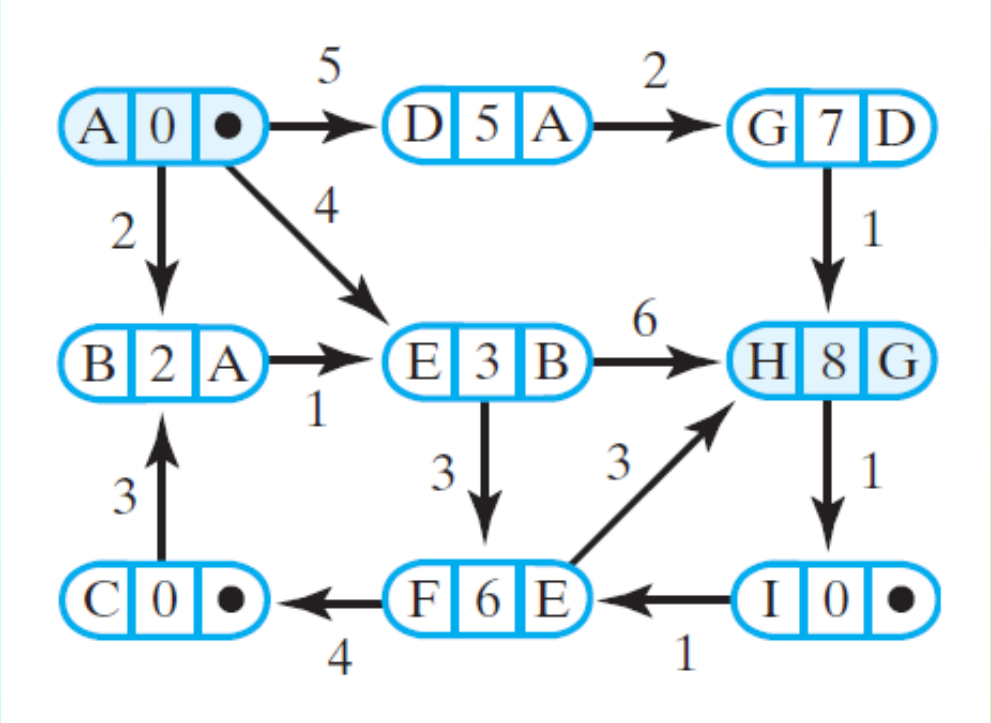


FIGURE 28-20 The graph in Figure 28-18a after finding the cheapest path from vertex A to vertex H

Q 10 Continue the trace begun in Figure 28-19 to find the shortest (cheapest) path from vertex A to vertex C .

Q 10 Continue the trace begun in Figure 28-19 to find the shortest (cheapest) path from vertex A to vertex C .

10. Vertex H has one unvisited neighbor, I. The cost of the path to I is the cost of the path to H plus the weight of the edge from H to I. This total cost is 9. Encapsulate vertex I, the cost 9, and the predecessor H into an object and add it to the priority queue.

Now remove the front entry from the priority queue. The entry contains H , and since H is visited already, ignore the entry. Remove the next entry. This entry also contains H , which is visited, so ignore it and remove the next entry. This entry contains I , which is unvisited. Visit I . I's neighbor F is visited, so remove the next entry from the priority queue. This entry contains C , so visit C .

Since C is the destination, construct the path by working backward from C , as we did in Segment 28.18. The shortest (cheapest) path is A → B → E → F → C , with a weight (cost) of 10.

Q 11 Why do we place instances of EntryPQ into the priority queue, instead of placing vertices?

Q 11 Why do we place instances of EntryPQ into the priority queue, instead of placing vertices?

11. Two or more entries in the priority queue can record data about the same vertex. For example, consider the trace in Figure 28-19. After we visit vertex B , the first two entries in the priority queue record data about vertex E. The first entry involves the path from B to E , while the second entry involves the path from A to E . While vertex E can record similar data for one path, it cannot do so for multiple paths.

Java Interfaces for the ADT Graph

- ADT graph different from other ADTs
  - Once instance created, we do not add, remove, or retrieve components
- Interface to create the graph and to obtain basic information
  - Listing 28-1

Note: Code listing files must be in same folder as PowerPoint files for links to work

Java Interfaces for the ADT Graph

- Interface to specify operations such as traversals and path searches
  - Listing 28-2
- For convenience, a third interface to combine first two
  - Listing 28-3

The following statements create the graph shown

```
BasicGraphInterface<String> airMap = new UndirectedGraph<String>();
airMap.addVertex("Boston");
airMap.addVertex("Provincetown");
airMap.addVertex("Nantucket");
airMap.addEdge("Boston", "Provincetown");
airMap.addEdge("Boston", "Nantucket");
```

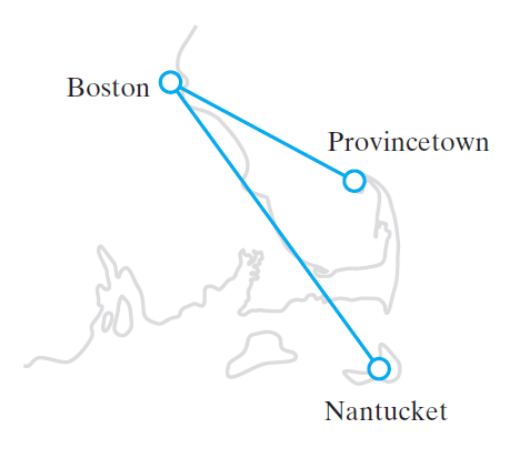


Figure 28-21 A portion of the flight map in Figure 28-6

End

Chapter 28