 Answers to review questions from Chapter 18

1. What is a graph?

**A *graph* is a set of nodes combined with a set of arcs, each of which connects two nodes in the graph.**

2. True or false: Trees are a subset of graphs, which form a more general class.

**True.**

3. What is the difference between a directed and an undirected graph?

**In an *undirected graph,* each arc connects the two nodes in both directions. In a *directed graph,* each arc has a specified direction that allows you to get from the start of the arc to the end of the arc, but not vice versa.**

4. If you are using a graph package that supports only directed graphs, how can you represent an undirected graph?

**You can use two directed arcs for every arc in the undirected graph, one in each direction.**

5. Define the following terms as they apply to graphs: *path, cycle, simple path, simple cycle*.

**A *path* is a sequence of arcs from one node to another in which the end of the first arc is the beginning of the second, and so on. A *cycle* is a path that begins and ends at the same node. A *simple path* contains no duplicated nodes; a *simple cycle* contains no duplicated nodes other than the start and end.**

6. What is relationship between the terms *neighbor* and *degree?*

**A *neighbor* is a node that can be reached by a single arc. The *degree* of a node is the number of arcs that connect it to other nodes in the graph.**

7. What is the difference between a strongly connected and a weakly connected graph?

**A *strongly connected* graph is a graph in which there is a path from every node to every other node following the direction of the arcs. A *weakly connected* graph is one that would be strongly connected if you replaced every directed arc with an undirected arc.**

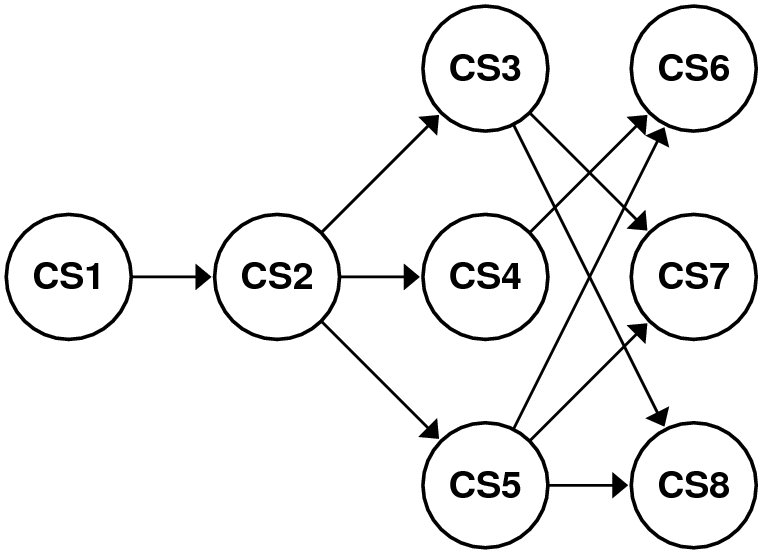
8. True or false: The term *weakly connected* has no practical relevance to undirected graphs because all such graphs are automatically strongly connected if they are connected at all.

**True.**

9. What terms do mathematicians typically use in place of the words *node* and *arc?*

**Mathematicians tend to use the words *vertex* and *edge.***

10. Suppose that the computer science offerings at some university consisted of eight courses with the following prerequisite structure:



Using the mathematical formulation for graphs described in this chapter, define this graph as a pair of sets.

**Nodes: { CS1, CS2, CS3, CS4, CS5, CS6, CS7, CS8 }**

**Arcs: { CS1→CS2, CS2→CS3, CS2→CS4, CS2→CS5, CS3→CS7, CS3→CS8, CS4→CS6, CS5→CS6, CS5→CS7, CS5→CS8 }**

11. Draw a diagram showing the adjacency list representation of the graph in the preceding question.

**CS1** → (**CS2**)

**CS2** → (**CS3**, **CS4**, **CS5**)

**CS3** → (**CS7**, **CS8**)

**CS4** → (**CS6**)

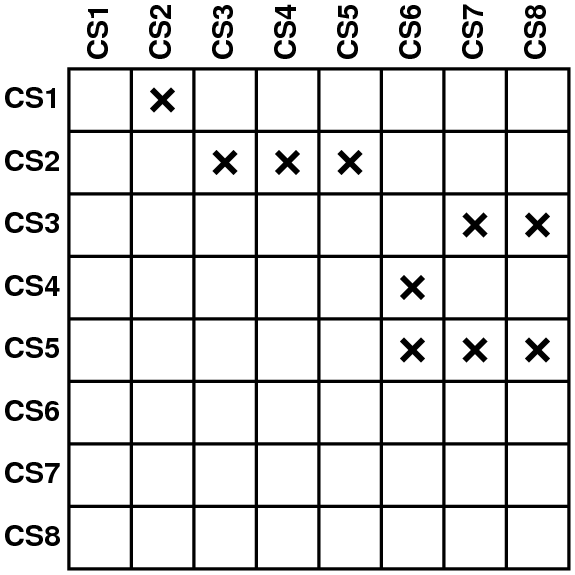
**CS5** → (**CS6**, **CS7**, **CS8**)

**CS6** → ( )

**CS7** → ()

**CS8** → ()

12. Given the prerequisite graph shown in question 10, what are the contents of the corresponding adjacency matrix?



13. What is the difference between a sparse and a dense graph?

**A graph in which the maximum degree is small in comparison to the size of the graph is said to be *sparse.* A graph in which these values are comparable is considered *dense.***

14. If you were asked to choose the underlying representation of a graph for a particular application, what factors would you consider in deciding whether to use adjacency lists or adjacency matrices in the implementation?

**This decision is a classic time‑space tradeoff. Adjacency lists use much less memory for sparse graphs but are likely to use more memory if the graphs are dense. Adjacency matrices are more efficient in terms of checking connections.**

15. Why is it unnecessary to implement a separate iterator facility for the graph package?

**Graphs are built on top of sets, which have their own iterator.**

16. Why do the sets used in either version of the **graph.h** interface use pointers to arcs and nodes as their element types?

**Nodes and arcs are shared in the data structure. For example, if two arcs connect to the same node, there must be a single structure for that node, rather than two copies of the same information. Using pointers facilitates this sharing.**

17. What are the two fundamental traversal strategies for graphs?

**Depth‑first and breadth-first traversals.**

18. Write down both the depth-first and the breadth-first traversal of the airline graph in Figure 18-1, starting from Atlanta. Assume that iteration over nodes and arcs always occurs in alphabetical order.

**Depth-first: Atlanta, Chicago, Denver, Dallas, Los Angeles, San Francisco, Portland, Seattle, Boston, New York**

**Breadth-first: Atlanta, Chicago, Dallas, New York, Denver, Los Angeles, San Francisco, Boston, Portland, Seattle**

19. What problem does this chapter cite as the most significant problem with including class definitions for **Node** and **Arc** in the **graph.h** interface?

**The fundamental problem is that dynamic memory allocation and inheritance do not always work together in C++ as seamlessly as they do in other languages, which makes it difficult to adopt an inheritance‑based approach.**

20. What rules does the **graph.h** interface impose on the client-defined types used to represent nodes and arcs?

**The node structure must contain a name and a set of arc pointers.**

**The arc structure must contain pointers to the start and finish nodes.**

21. What is a greedy algorithm?

**A *greedy algorithm* is one that operates by making the optimal choice at each local decision point, as opposed to using a backtracking technique to find a best set of choices for the entire problem.**

22. Explain the operation of Dijkstra’s algorithm for finding minimum-cost paths.

**Dijkstra’s algorithm operates by exploring all paths from the starting node in order of increasing total path cost until if finds the destination. This path must be the optimal one, because the algorithm has already explored all paths beginning at the starting node that have a lower cost.**

23. Show the contents of the priority queue at each step of the trace of Dijkstra’s algorithm shown in Figure 18‑11.

**Enqueue San Francisco→Portland (550)**

**[ San Francisco→Portland (550) ]**

**Enqueue San Francisco→Dallas (1468)**

**[ San Francisco→Portland (550), San Francisco→Dallas (1468) ]**

**Enqueue San Francisco→Denver (954)**

**[ San Francisco→Portland (550), San Francisco→Denver (954), San Francisco→Dallas (1468) ]**

**Dequeue San Francisco→Portland (550)**

**[ San Francisco→Denver (954), San Francisco→Dallas (1468) ]**

**Enqueue San Francisco→Portland→Seattle (680)**

**[ San Francisco→Portland→Seattle (680), San Francisco→Denver (954), San Francisco→Dallas (1468) ]**

**Dequeue San Francisco→Portland→Seattle (680)**

**[ San Francisco→Denver (954), San Francisco→Dallas (1468) ]**

**Enqueue San Francisco→Portland→Seattle→Boston (3169)**

**[ San Francisco→Denver (954), San Francisco→Dallas (1468), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Denver (954)**

**[ San Francisco→Dallas (1468), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Denver→Chicago (1861)**

**[ San Francisco→Dallas (1468), San Francisco→Denver→Chicago (1861), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Denver→Dallas (1604)**

**[ San Francisco→Dallas (1468), San Francisco→Denver→Dallas (1604), San Francisco→Denver→Chicago (1861), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Dallas (1468)**

**[ San Francisco→Denver→Dallas (1604), San Francisco→Denver→Chicago (1861), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Dallas→Atlanta (2193)**

**[ San Francisco→Denver→Dallas (1604), San Francisco→Denver→Chicago (1861), San Francisco→Dallas→Atlanta (2193), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Dallas→Los Angeles (2708)**

**[ San Francisco→Denver→Dallas (1604), San Francisco→Denver→Chicago (1861), San Francisco→Dallas→Atlanta (2193), San Francisco→Dallas→Los Angeles (2708), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Denver→Dallas (1604)**

**[ San Francisco→Denver→Chicago (1861), San Francisco→Dallas→Atlanta (2193), San Francisco→Dallas→Los Angeles (2708), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Denver→Chicago (1861)**

**[ San Francisco→Dallas→Atlanta (2193), San Francisco→Dallas→Los Angeles (2708), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Denver→Chicago→Atlanta (2460)**

**[ San Francisco→Dallas→Atlanta (2193), San Francisco→Denver→Chicago→Atlanta (2460), San Francisco→Dallas→Los Angeles (2708), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Dallas→Atlanta (2193)**

**[ San Francisco→Denver→Chicago→Atlanta (2460), San Francisco→Dallas→Los Angeles (2708), San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Dallas→Atlanta→New York (2949)**

**[ San Francisco→Denver→Chicago→Atlanta (2460), San Francisco→Dallas→Los Angeles (2708), San Francisco→Dallas→Atlanta→New York (2949), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Denver→Chicago→Atlanta (2460)**

**[ San Francisco→Dallas→Los Angeles (2708), San Francisco→Dallas→Atlanta→New York (2949), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Dallas→Los Angeles (2708)**

**[ San Francisco→Dallas→Atlanta→New York (2949), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Dallas→Atlanta→New York (2949)**

**[ San Francisco→Portland→Seattle→Boston (3169) ]**

**Enqueue San Francisco→Dallas→Atlanta→New York→Boston (3140)**

**[ San Francisco→Dallas→Atlanta→New York→Boston (3140), San Francisco→Portland→Seattle→Boston (3169) ]**

**Dequeue San Francisco→Dallas→Atlanta→New York→Boston (3140)**

**[ San Francisco→Portland→Seattle→Boston (3169) ]**

24. Using Figure 18‑12 as a model, trace the execution of Dijkstra’s algorithm to find the shortest path from Portland to Atlanta.

**Fix distance to Portland at 0**

**Process the arcs out of Portland (Seattle, San Francisco)**

**Enqueue the path: Portland→Seattle (130)**

**Enqueue the path: Portland→San Francisco (550)**

**Dequeue the shortest path: Portland→Seattle (130)**

**Fix distance to Seattle at 130**

**Process the arcs out of Seattle (Boston, Portland)**

**Enqueue the path: Portland→Seattle→Boston (2619)**

**Ignore Portland because its distance is fixed**

**Dequeue the shortest path: Portland→San Francisco (550)**

**Fix distance to San Francisco at 550**

**Process the arcs out of San Francisco (Portland, Dallas, Denver)**

**Ignore Portland because its distance is fixed**

**Enqueue the path: Portland→San Francisco→Dallas (2018)**

**Enqueue the path: Portland→San Francisco→Denver (1504)**

**Dequeue the shortest path: Portland→San Francisco→Denver (1504)**

**Fix distance to Denver at 1504**

**Process the arcs out of Denver (Chicago, Dallas, San Francisco)**

**Enqueue the path: Portland→San Francisco→Denver→Chicago (2411)**

**Enqueue the path: Portland→San Francisco→Denver→Dallas (2154)**

**Ignore San Francisco because its distance is fixed**

**Dequeue the shortest path: Portland→San Francisco→Dallas (2018)**

**Fix distance to Dallas at 2018**

**Process the arcs out of Dallas (Atlanta, Denver, Los Angeles, San Francisco)**

**Enqueue the path: Portland→San Francisco→Dallas→Atlanta (2743)**

**Ignore Denver because its distance is fixed**

**Enqueue the path: Portland→San Francisco→Dallas→Los Angeles (3258)**

**Ignore San Francisco because its distance is fixed**

**Dequeue the shortest path: Portland→San Francisco→Denver→Dallas (2154)**

**Ignore this path because the distance to Dallas is fixed**

**Dequeue the shortest path: Portland→San Francisco→Denver→Chicago (2411)**

**Fix distance to Chicago at 2411**

**Process the arcs out of Chicago (Atlanta, Denver)**

**Enqueue the path: Portland→San Francisco→Denver→Chicago→Atlanta (3010)**

**Ignore Denver because its distance is fixed**

**Dequeue the shortest path: Portland→Seattle→Boston (2619)**

**Fix distance to Boston at 2619**

**Process the arcs out of Boston (New York, Seattle)**

**Enqueue the path: Portland→Seattle→Boston→New York (2810)**

**Ignore Seattle because its distance is fixed**

**Dequeue the shortest path: Portland→San Francisco→Dallas→Atlanta (2743)**