A depth-first or breadth-first search can be used to find the connected components of a graph. We pick an arbitrary node as a source node and then conduct a search. When the algorithm terminates, all visited nodes belong to one component. We then find an unvisited node in the graph to serve as a source for another search, which will produce a second component. We continue this process until there are no unvisited nodes in the graph.

Although we defined reachability only for directed graphs, the concept also makes sense for undirected, unconnected graphs. Let us consider only simple undirected, unconnected graphs but impose the convention that, even though there are no loops, each node is reachable from itself. Reachability then becomes an equivalence relation on the set of nodes of the graph; our convention imposes the reflexive property, and symmetry and transitivity follow because the graph is undirected. This equivalence relation partitions the nodes of the graph into equivalence classes, and each class consists of the nodes in one connected component of the graph. Warshall's algorithm can be applied to undirected graphs as well as directed graphs. Using Warshall's algorithm results in a matrix from which the nodes making up various components of the graph can be determined, but this requires more work than using the depth-first search.

As a final remark about depth-first search, we saw in Section 1.5 that the programming language Prolog, when processing a query based on a recursive definition, pursues a depth-first search strategy (Example 40).

SECTION 7.4 REVIEW

TECHNIQUES

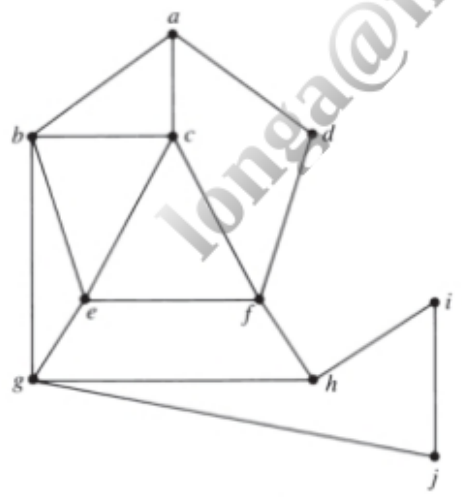
- Conduct a depth-first search of a graph.
- Conduct a breadth-first search of a graph.

MAIN IDEAS

- Algorithms exist to visit the nodes of a graph systematically.
- Depth-first and breadth-first searches can serve as a basis for other tasks.

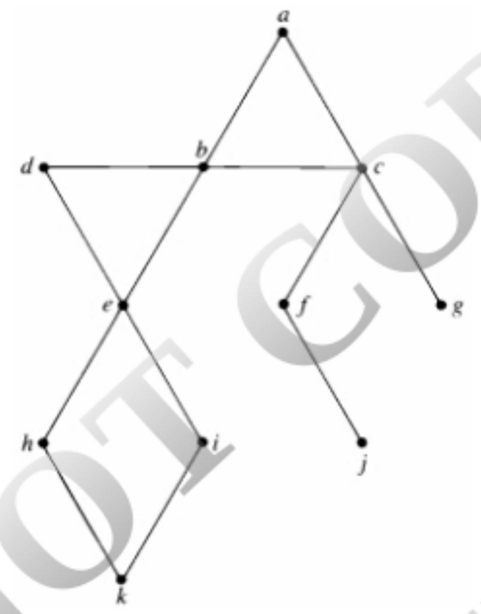
EXERCISES 7.4

For Exercises 1–6, write the nodes in a depth-first search of the following graph, beginning with the node specified.



1. *a* 2. *c* 3. *d* 4. *g* 5. *e* 6. *h*

For Exercises 7–10, write the nodes in a depth-first search of the following graph, beginning with the node specified.



7. *a* 8. *e* 9. *f* 10. *h*

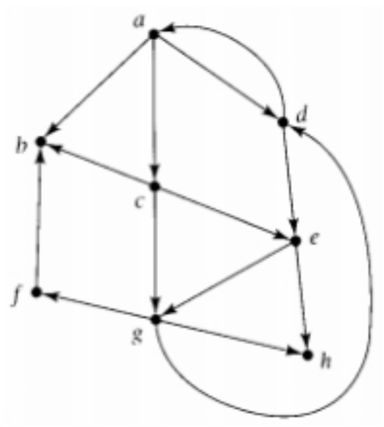
For Exercises 11–16, write the nodes in a breadth-first search of the graph for Exercises 1–6, beginning with the node specified.

11. *a* 12. *c* 13. *d* 14. *g* 15. *e* 16. *h*

For Exercises 17–20, write the nodes in a breadth-first search of the graph for Exercises 7–10, beginning with the node specified.

17. *a* 18. *e* 19. *f* 20. *h*

For Exercises 21–24, write the nodes in a depth-first search of the following graph, beginning with the node specified.

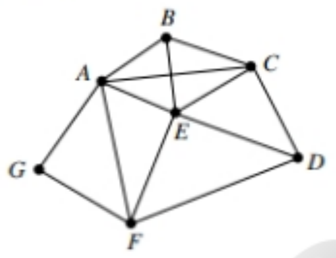


21. *a* 22. *g* 23. *f* 24. *e*

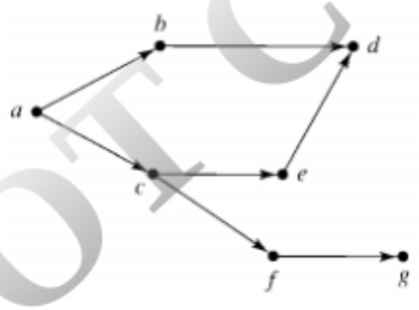
For Exercises 25–28, write the nodes in a breadth-first search of the graph for Exercises 21–24, beginning with the node specified.

25. *a* 26. *g* 27. *f* 28. *e*

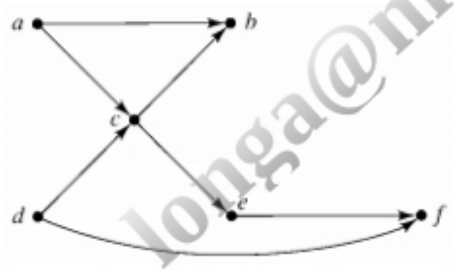
29. In the computer network in the accompanying figure, the same message is to be broadcast from node *C* to nodes *A*, *E*, *F*, and *G*. One way to do this is to find the shortest path from *C* to each of these nodes and send out multiple copies of the same message. A more efficient approach is to send one copy out from *C* along a spanning tree for the subgraph containing the nodes involved. Use the depth-first search algorithm to find a spanning tree for the subgraph.



30. Using the graph for Exercise 29, use the breadth-first search algorithm to find a spanning tree for the subgraph.
31. Use the depth-first search algorithm to do a topological sort on the following graph. Indicate the counting numbers on the graph. Also state the starting node or nodes for the search.



32. Use the depth-first search algorithm to do a topological sort on the following graph. Indicate the counting numbers on the graph. Also state the starting node or nodes for the search.



33. The data structure used to implement a breadth-first search is a queue. What is the appropriate data structure to implement a depth-first search?
34. Find a way to traverse a tree in level order, that is, so that all nodes at the same depth are listed from left to right for increasing depth. (*Hint*: We already have a way to do this.)
35. Describe how the depth-first search algorithm can be used in a connected (undirected) graph to detect the presence of cycles in the graph. (While it is simple to look at Figure 7.13 and see that *a*–*b*–*c*–*e*–*a*, for example, is a cycle, in a huge graph with thousands of node and arcs, a cycle may be less easy to spot, in addition to which you might not even have a visual representation.)

36. a. Describe the order in which nodes are visited in a breadth-first search of the bipartite complete graph $K_{m,n}$.
   b. Describe the order in which nodes are visited in a depth-first search of the bipartite complete graph $K_{m,n}$.

SECTION 7.5 ARTICULATION POINTS AND COMPUTER NETWORKS

The Problem Statement

In a graph that represents a computer network, the nodes denote the communicating entities (end-user computers, servers, routers, and so on) and the arcs denote the communications medium (coaxial cable, fiber optic, and so on). Such a graph should be a connected graph, so that there is a path between every pair of nodes. To minimize the length of cable or wire required, we would choose a minimum spanning tree. However, if an arc in a minimum spanning tree is removed (for example, that section of cable or wire is damaged or broken), then the graph is no longer connected. Each arc becomes a single point of failure for the network. That is why such a network usually contains more arcs than just those of a minimal spanning tree. However, even in a graph sufficiently rich in arcs to withstand the loss of a single arc, a node may be a single point of failure. If the node fails (and thus is logically removed), the arcs of which that node is an endpoint are disabled and the result may be a disconnected graph.

DEFINITION ARTICULATION POINT
A node in a simple, connected graph is an **articulation point** if its removal (along with its attached arcs) causes the remaining graph to be disconnected.

EXAMPLE 14 Node *d* in the graph of Figure 7.20a is an articulation point. Removing *d* results in the disconnected graph of Figure 7.20b.

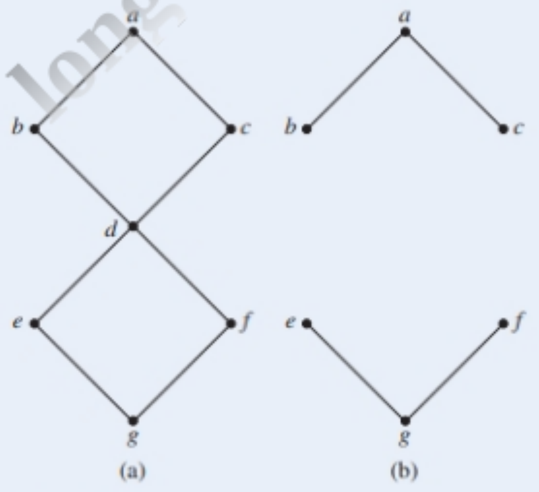


Figure 7.20

DEFINITION BICONNECTED GRAPH
A simple, connected graph is **biconnected** if it has no articulation points.