left as an exercise (Exercise 9.26).

9.6 Exercises

9.1. Show the result of running depth-first search on the undirected graph shown in Fig. 9.8(a) starting at vertex $a$. Give the classification of edges as tree edges or back edges.

9.2. Show the result of running depth-first search on the directed graph shown in Fig. 9.8(b) starting at vertex $a$. Give the classification of edges as tree edge, back edges, forward edges or cross edges.

9.3. Show the result of running depth-first search on the undirected graph of Fig. 9.9 starting at vertex $f$. Give the classification of edges.

9.4. Show the result of running depth-first search on the directed graph of Fig. 9.10 starting at vertex $e$. Give the classification of edges.

9.5. Give an iterative version of Algorithm DFS that uses a stack to store unvisited vertices.
9.6. What will be the time complexity of the depth-first search algorithm if the input graph is represented by an adjacency matrix (see Sec. 3.3.1 for graph representation).

9.7. Show that when depth-first search is applied to an undirected graph $G$, the edges of $G$ will be classified as either tree edges or back edges. That is, there are no forward edges or cross edges.

9.8. Suppose that Algorithm dfs is applied to an undirected graph $G$. Give an algorithm that classifies the edges of $G$ as either tree edges or back edges.

9.9. Suppose that Algorithm dfs is applied to a directed graph $G$. Give an algorithm that classifies the edges of $G$ as either tree edges, back edges, forward edges or cross edges.

9.10. Give an algorithm that counts the number of connected components in an undirected graph using depth-first search or breadth-first search.

9.11. Given an undirected graph $G$, design an algorithm to list the vertices in each connected component of $G$ separately.

9.12. Give an $O(n)$ time algorithm to determine whether a connected undirected graph with $n$ vertices contains a cycle.

9.13. Apply the articulation points algorithm to obtain the articulation points of the undirected graph shown in Fig. 9.9.

9.14. Let $T$ be the depth-first search tree resulting from a depth-first search traversal on a connected undirected graph. Show that the root of $T$ is an articulation point if and only if it has two or more children. (See Sec. 9.3.3)

9.15. Let $T$ be the depth-first search tree resulting from a depth-first search traversal on a connected undirected graph. Show that a vertex $v$ other than the root is an articulation point if and only if $v$ has a child $w$ with $\beta[w] \geq \alpha[v]$. (See Sec. 9.3.3)

9.16. Apply the strongly connected components algorithm on the directed graph shown in Fig. 9.10.

9.17. Show that in the strongly connected components algorithm, any choice
of the first vertex to carry out the depth-first search traversal leads to
the same solution.

9.18. An edge of a connected undirected graph \( G \) is called a bridge if its deletion disconnects \( G \). Modify the algorithm for finding articulation points so that it detects bridges instead of articulation points.

9.19. Show the result of running breadth-first search on the undirected graph shown in Fig. 9.8(a) starting at vertex \( a \).

9.20. Show the result of running breadth-first search on the directed graph shown in Fig. 9.8(b) starting at vertex \( a \).

9.21. Show the result of running breadth-first search on the undirected graph of Fig. 9.1 starting at vertex \( a \).

9.22. Show the result of running breadth-first search on the undirected graph of Fig. 9.9 starting at vertex \( f \).

9.23. Show the result of running breadth-first search on the directed graph of Fig. 9.10 starting at vertex \( e \).

9.24. Show that when breadth-first search is applied to an undirected graph \( G \), the edges of \( G \) will be classified as either tree edges or cross edges. That is, there are no back edges or forward edges.

9.25. Show that when breadth-first search is applied to a directed graph \( G \), the edges of \( G \) will be classified as tree edges, back edges or cross edges. That is, unlike the case of depth-first search, the search does not result in forward edges.

9.26. Let \( G \) be a graph (directed or undirected), and let \( s \) be a vertex in \( G \). Modify Algorithm \texttt{bfs} so that it outputs the shortest path measured in the number of edges from \( s \) to every other vertex.

9.27. Use depth-first search to find a spanning tree for the complete bipartite graph \( K_{3,3} \). (See Sec. 3.3 for the definition of \( K_{3,3} \)).

9.28. Use breadth-first search to find a spanning tree for the complete bipartite graph \( K_{3,3} \). Compare this tree with the tree obtained in Exercise 9.27.

9.29. Suppose that Algorithm \texttt{bfs} is applied to an undirected graph \( G \). Give an algorithm that classifies the edges of \( G \) as either tree edges or cross edges.

9.30. Suppose that Algorithm \texttt{bfs} is applied to a directed graph \( G \). Give an algorithm that classifies the edges of \( G \) as either tree edges, back edges or cross edges.

9.31. Show that the time complexity of breadth-first search when applied on a graph with \( n \) vertices and \( m \) edges is \( \Theta(n + m) \).

9.32. Design an efficient algorithm to determine whether a given graph is bipartite (see Sec. 3.3 for the definition of a bipartite graph).
9.33. Design an algorithm to find a cycle of shortest length in a directed graph. Here the length of a cycle is measured in terms of its number of edges.

9.34. Let $G$ be a connected undirected graph, and $T$ the spanning tree resulting from applying breadth-first search on $G$ starting at vertex $r$. Prove or disprove that the height of $T$ is minimum among all spanning trees with root $r$.

9.7 Bibliographic notes

Graph traversals are discussed in several books on algorithms, either separately or intermixed with other graph algorithms (see the bibliographic notes of Chapter 1). Hopcroft and Tarjan (1973) were the first to recognize the algorithmic importance of depth-first search. Several applications of depth-first search can be found in this paper and in Tarjan (1972). Algorithm STRONGCONNECTCOMP for the strongly connected components is similar to the one by Sharir (1981). Tarjan (1972) contains an algorithm for finding the strongly connected components that needs only one depth-first search traversal. Breadth-first search was discovered independently by Moore (1959) and Lee (1961).