Algorithm 8.6  

**HUFFMAN**  

**Input:** A set $C = \{c_1, c_2, \ldots, c_n\}$ of $n$ characters and their frequencies  
$\{f(c_1), f(c_2), \ldots, f(c_n)\}$.  

**Output:** A Huffman tree $(V, T)$ for $C$.  

1. Insert all characters into a min-heap $H$ according to their frequencies.  
2. $V \leftarrow C$; $T \leftarrow \{\}$  
3. for $j \leftarrow 1$ to $n - 1$  
4. $c \leftarrow \text{DELETEMIN}(H)$  
5. $c' \leftarrow \text{DELETEMIN}(H)$  
6. $f(v) \leftarrow f(c) + f(c')$ \{$v$ is a new node$\}$  
7. \text{INSERT}(H, v)$  
8. $V = V \cup \{v\}$ \{Add $v$ to $V$\}  
9. $T = T \cup \{(v, c), (v, c')\}$ \{Make $c$ and $c'$ children of $v$ in $T$\}  
10. end while  

8.6  

**Exercises**  

8.1. Is Algorithm LINEARESEARCH described in Sec. 1.3 a greedy algorithm? Explain.  

8.2. Is Algorithm MAJORITY described in Sec. 5.7 a greedy algorithm? Explain.  

8.3. This exercise is about the money change problem stated in Exercise 7.30. Consider a currency system that has the following coins and their values: dollar (100 cents), quarter (25 cents), dime (10 cents), nickel (5 cents) and 1-cent coins. (A unit-value coin is always required). Suppose we want to give a change of value $n$ cents in such a way that the total number of coins $n$ is minimized. Give a greedy algorithm to solve this problem.  

8.4. Give a counterexample to show that the greedy algorithm obtained in Exercise 8.3 does not always work if we instead use coins of values 1 cent, 5 cents, 7 cents and 11 cents. Note that in this case dynamic programming can be used to find the minimum number of coins. (See Exercises 7.30 and 7.31).  

8.5. Suppose in the money change problem of Exercise 8.3 the coin values are: $1, 2, 4, 8, 16, \ldots, 2^k$, for some positive integer $k$. Give an $O(\log n)$ algorithm to solve the problem if the value to be paid is $n < 2^{k+1}$?  

8.6. For what denominations $\{v_1, v_2, \ldots, v_k\}, k \geq 2$, does the greedy algorithm for the money change problem stated in Exercise 7.30 always give the minimum number of coins? Prove your answer.  

8.7. Let $G = (V, E)$ be an undirected graph. A vertex cover for $G$ is a subset
Consider the following algorithm for finding a vertex cover for $G$. First, order the vertices in $V$ by decreasing order of degree. Next, execute the following step until all edges are covered. Pick a vertex of highest degree that is incident to at least one edge in the remaining graph, add it to the cover, and delete all edges incident to that vertex. Show that this greedy approach does not always result in a vertex cover of minimum size.

8.8. Let $G = (V, E)$ be an undirected graph. A clique $C$ in $G$ is a subgraph of $G$ that is a complete graph by itself. A clique $C$ is maximum if there is no other clique $C'$ in $G$ such that the size of $C'$ is greater than the size of $C$. Consider the following method that attempts to find a maximum clique in $G$. Initially, let $C = G$. Repeat the following step until $C$ is a clique. Delete from $C$ a vertex that is not connected to every other vertex in $C$. Show that this greedy approach does not always result in a maximum clique.

8.9. Let $G = (V, E)$ be an undirected graph. A coloring of $G$ is an assignment of colors to the vertices in $V$ such that no two adjacent vertices have the same color. The coloring problem is to determine the minimum number of colors needed to color $G$. Consider the following greedy method that attempts to solve the coloring problem. Let the colors be 1, 2, 3, … First, color as many vertices as possible using color 1. Next, color as many vertices as possible using color 2, and so on. Show that this greedy approach does not always color the graph using the minimum number of colors.

8.10. Let $A_1, A_2, \ldots, A_m$ be $m$ arrays of integers each sorted in nondecreasing order. Each array $A_j$ is of size $n_j$. Suppose we want to merge all arrays into one array $A$ using an algorithm similar to Algorithm merge described in Sec. 1.4. Give a greedy strategy for the order in which these arrays should be merged so that the overall number of comparisons is minimized. For example, if $m = 3$, we may merge $A_1$ with $A_2$ to obtain $A_4$ and then merge $A_3$ with $A_4$ to obtain $A$. Another alternative is to merge $A_2$ with $A_3$ to obtain $A_4$ and then merge $A_1$ with $A_4$ to obtain $A$. Yet another alternative is to merge $A_1$ with $A_3$ to obtain $A_4$ and then merge $A_2$ with $A_4$ to obtain $A$. (Hint: Give an algorithm similar to Algorithm Huffman).

8.11. Analyze the time complexity of the algorithm in Exercise 8.10.

8.12. Consider the following greedy algorithm which attempts to find the distance from vertex $s$ to vertex $t$ in a directed graph $G$ with positive lengths on its edges. Starting from vertex $s$, go to the nearest vertex, say $x$. From vertex $x$, go to the nearest vertex, say $y$. Continue in this manner until you arrive at vertex $t$. Give a graph with the fewest number of vertices to show that this heuristic does not always produce the distance from $s$
to $t$. (Recall that the distance from vertex $u$ to vertex $v$ is the length of a shortest path from $u$ to $v$).

8.13. Apply Algorithm Dijkstra on the directed graph shown in Fig. 8.7. Assume that vertex 1 is the start vertex.


8.15. What are the merits and demerits of using the adjacency matrix representation instead of the adjacency lists in the input to Algorithm Dijkstra?

8.16. Modify Algorithm Dijkstra so that it finds the shortest paths in addition to their lengths.

8.17. Prove that the subgraph defined by the paths obtained from the modified shortest path algorithm as described in Exercise 8.16 is a tree. This tree is called the shortest path tree.

8.18. Can a directed graph have two distinct shortest path trees (see Exercise 8.17)? Prove your answer.

8.19. Give an example of a directed graph to show that Algorithm Dijkstra does not always work if some of the edges have negative weights.

8.20. Show that the proof of correctness of Algorithm Dijkstra (Lemma 8.1) does not work if some of the edges in the input graph have negative weights.

8.21. Let $G = (V, E)$ be a directed graph such that removing the directions from its edges results in a planar graph. What is the running time of Algorithm SHORTESTPATH when applied to $G$? Compare that to the running time when using Algorithm Dijkstra.

8.22. Let $G = (V, E)$ be a directed graph such that $m = O(n^{1.5})$, where $n = |V|$ and $m = |E|$. What changes should be made to Algorithm SHORTESTPATH so that it will run in time $O(m)$?

8.23. Show the result of applying Algorithm Kruskal to find a minimum cost spanning tree for the undirected graph shown in Fig. 8.8.
8.24. Show the result of applying Algorithm PRIM to find a minimum cost spanning tree for the undirected graph shown in Fig. 8.8.

8.25. Let $G = (V, E)$ be an undirected graph such that $m = O(n^{1.99})$, where $n = |V|$ and $m = |E|$. Suppose you want to find a minimum cost spanning tree for $G$. Which algorithm would you choose: Algorithm PRIM or Algorithm KRUSKAL? Explain.

8.26. Let $e$ be an edge of minimum weight in an undirected graph $G$. Show that $e$ belongs to some minimum cost spanning tree of $G$.

8.27. Does Algorithm PRIM work correctly if the graph has negative weights? Prove your answer.

8.28. Let $G$ be an undirected weighted graph such that no two edges have the same weight. Prove that $G$ has a unique minimum cost spanning tree.

8.29. What is the number of spanning trees of a complete undirected graph $G$ with $n$ vertices? For example, the number of spanning trees of $K_3$, the complete graph with three vertices, is 3.

8.30. Let $G$ be a directed weighted graph such that no two edges have the same weight. Let $T$ be a shortest path tree for $G$ (see Exercise 8.17). Let $G'$ be the undirected graph obtained by removing the directions from the edges of $G$. Let $T'$ be a minimum spanning tree for $G'$. Prove or disprove that $T = T'$.

8.31. Use Algorithm HUFFMAN to find an optimal code for the characters a, b, c, d, e and f whose frequencies in a given text are respectively 7, 5, 3, 2, 12, 9.

8.32. Prove that the graph obtained in Algorithm HUFFMAN is a tree.

8.33. Algorithm HUFFMAN constructs the code tree in a bottom-up fashion. Is it a dynamic programming algorithm?

8.34. Let $B = \{b_1, b_2, \ldots, b_n\}$ and $W = \{w_1, w_2, \ldots, w_n\}$ be two sets of black and white points in the plane. Each point is represented by the pair $(x, y)$ of $x$ and $y$ coordinates. A black point $b_i = (x_i, y_i)$ dominates a white point $w_j = (x_j, y_j)$ if and only if $x_i \geq x_j$ and $y_i \geq y_j$. A matching between a black point $b_i$ and a white point $w_j$ is possible if
$b_i$ dominates $w_j$. A matching $M = \{(b_{i_1}, w_{j_1}), (b_{i_2}, w_{j_2}), \ldots, (b_{i_k}, w_{j_k})\}$ between the black and white points is maximum if $k$, the number of matched pairs in $M$, is maximum. Design a greedy algorithm to find a maximum matching in $O(n \log n)$ time. (Hint: Sort the black points in increasing $x$-coordinates and use a heap for the white points).

### 8.7 Bibliographic notes

The greedy graph algorithms are discussed in most books on algorithms (see the bibliographic notes of Chapter 1).

Algorithm Dijkstra for the single source shortest path problem is from Dijkstra (1959). The implementation using a heap is due to Johnson (1977); see also Tarjan (1983). The best known asymptotic running time for this problem is $O(m + n \log n)$, which is due to Fredman and Tarjan (1987).

Graham and Hell (1985) discuss the long history of the minimum cost spanning tree problem, which has been extensively studied. Algorithm Kruskal comes from Kruskal (1956). Algorithm Prim is due to Prim (1957). The improvement using heaps can be found in Johnson (1975). More sophisticated algorithms can be found in Yao (1975), Cheriton and Tarjan (1976) and Tarjan (1983). Algorithm Huffman for file compression is due to Huffman (1952) (see also Knuth (1968)).