k_4 is the left child of k_5
k_3 is the left child of k_4
d_2 is the left child of k_3
d_3 is the right child of k_3
d_4 is the right child of k_4
d_5 is the right child of k_5

corresponding to the optimal binary search tree shown in Figure 14.9(b).

14.5-2

Determine the cost and structure of an optimal binary search tree for a set of $n = 7$ keys with the following probabilities:

i	0	1	2	3	4	5	6	7
p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

14.5-3

Suppose that instead of maintaining the table $w[i, j]$, you computed the value of $w(i, j)$ directly from equation (14.12) in line 9 of OPTIMAL-BST and used this computed value in line 11. How would this change affect the asymptotic running time of OPTIMAL-BST?

★ 14.5-4

Knuth [264] has shown that there are always roots of optimal subtrees such that $root[i, j - 1] \le root[i, j] \le root[i + 1, j]$ for all $1 \le i < j \le n$. Use this fact to modify the OPTIMAL-BST procedure to run in $\Theta(n^2)$ time.

Problems

14-1 Longest simple path in a directed acyclic graph

You are given a directed acyclic graph $G = (V, E)$ with real-valued edge weights and two distinguished vertices s and t. The ***weight*** of a path is the sum of the weights of the edges in the path. Describe a dynamic-

programming approach for finding a longest weighted simple path from s to t. What is the running time of your algorithm?

14-2 Longest palindrome subsequence

A ***palindrome*** is a nonempty string over some alphabet that reads the same forward and backward. Examples of palindromes are all strings of length 1, `civic`, `racecar`, and `aibohphobia` (fear of palindromes).

Give an efficient algorithm to find the longest palindrome that is a subsequence of a given input string. For example, given the input `character`, your algorithm should return `carac`. What is the running time of your algorithm?

14-3 Bitonic euclidean traveling-salesperson problem

In the ***euclidean traveling-salesperson problem***, you are given a set of n points in the plane, and your goal is to find the shortest closed tour that connects all n points.

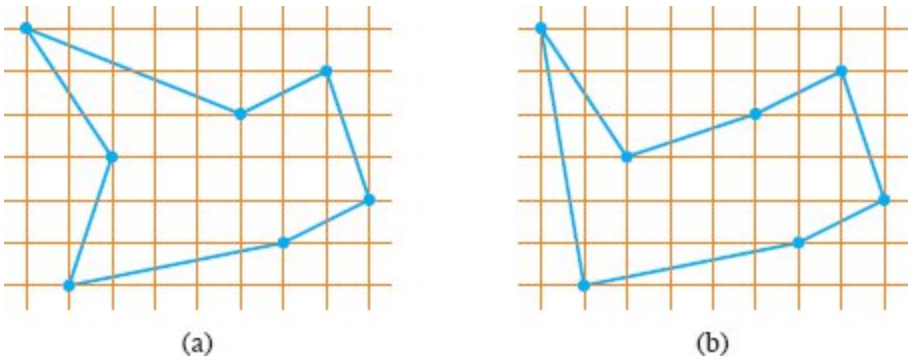

Figure 14.11 Seven points in the plane, shown on a unit grid. **(a)** The shortest closed tour, with length approximately 24.89. This tour is not bitonic. **(b)** The shortest bitonic tour for the same set of points. Its length is approximately 25.58.

Figure 14.11(a) shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time (see Chapter 34).

J. L. Bentley has suggested simplifying the problem by considering only ***bitonic tours***, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. Figure 14.11(b) shows the shortest bitonic

tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (*Hint:* Scan left to right, maintaining optimal possibilities for the two parts of the tour.)

14-4 Printing neatly

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width). The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters, which are to be printed neatly on a number of lines that hold a maximum of M characters each. No word exceeds the line length, so that $l_i \le M$ for $i = 1, 2, \ldots, n$. The criterion of "neatness" is as follows. If a given line contains words i through j, where $i \le j$, and exactly one space appears between words, then the number of extra space characters at the end of the line is $M - j + i - \sum_{k=i}^{j} l_k$, which must be nonnegative so that the words fit on the line. The goal is to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly. Analyze the running time and space requirements of your algorithm.

14-5 Edit distance

In order to transform a source string of text $x[1 : m]$ to a target string $y[1 : n]$, you can perform various transformation operations. The goal is, given x and y, to produce a series of transformations that changes x to y. An array z—assumed to be large enough to hold all the characters it needs—holds the intermediate results. Initially, z is empty, and at termination, you should have $z[j] = y[j]$ for $j = 1, 2, \ldots, n$. The procedure for solving this problem maintains current indices i into x and j into z, and the operations are allowed to alter z and these indices. Initially, $i = j = 1$. Every character in x must be examined during the transformation,

which means that at the end of the sequence of transformation operations, $i = m + 1$.

You may choose from among six transformation operations, each of which has a constant cost that depends on the operation:

Copy a character from x to z by setting $z[j] = x[i]$ and then incrementing both i and j. This operation examines $x[i]$ and has cost Q_C.

Replace a character from x by another character c, by setting $z[j] = c$, and then incrementing both i and j. This operation examines $x[i]$ and has cost Q_R.

Delete a character from x by incrementing i but leaving j alone. This operation examines $x[i]$ and has cost Q_D.

Insert the character c into z by setting $z[j] = c$ and then incrementing j, but leaving i alone. This operation examines no characters of x and has cost Q_I.

Twiddle (i.e., exchange) the next two characters by copying them from x to z but in the opposite order: setting $z[j] = x[i + 1]$ and $z[j + 1] = x[i]$, and then setting $i = i + 2$ and $j = j + 2$. This operation examines $x[i]$ and $x[i + 1]$ and has cost Q_T.

Kill the remainder of x by setting $i = m + 1$. This operation examines all characters in x that have not yet been examined. This operation, if performed, must be the final operation. It has cost Q_K.

Figure 14.12 gives one way to transform the source string `algorithm` to the target string `altruistic`. Several other sequences of transformation operations can transform `algorithm` to `altruistic`.

Assume that $Q_C < Q_D + Q_I$ and $Q_R < Q_D + Q_I$, since otherwise, the copy and replace operations would not be used. The cost of a given sequence of transformation operations is the sum of the costs of the individual operations in the sequence. For the sequence above, the cost of transforming `algorithm` to `altruistic` is $3Q_C + Q_R + Q_D + 4Q_I + Q_T + Q_K$.

a. Given two sequences $x[1 : m]$ and $y[1 : n]$ and the costs of the transformation operations, the ***edit distance*** from x to y is the cost of the least expensive operation sequence that transforms x to y. Describe a dynamic-programming algorithm that finds the edit distance from $x[1 : m]$ to $y[1 : n]$ and prints an optimal operation sequence. Analyze the running time and space requirements of your algorithm.

Operation	x	z
initial strings	a`lgorithm`	_
copy	a`lgorithm`	a_
copy	a`lgorithm`	al_
replace by `t`	a`lgorithm`	alt_
delete	a`lgorithm`	alt_
copy	a`lgorithm`	altr_
insert `u`	a`lgorithm`	altru_
insert `i`	a`lgorithm`	altrui_
insert `s`	a`lgorithm`	altruis_
twiddle	a`lgorithm`	altruisti_
insert `c`	a`lgorithm`	altruistic_
kill	a`lgorithm_`	altruistic_

Figure 14.12 A sequence of operations that transforms the source `algorithm` to the target string `altruistic`. The underlined characters are $x[i]$ and $z[j]$ after the operation.

The edit-distance problem generalizes the problem of aligning two DNA sequences (see, for example, Setubal and Meidanis [405, Section 3.2]). There are several methods for measuring the similarity of two DNA sequences by aligning them. One such method to align two sequences x and y consists of inserting spaces at arbitrary locations in the two sequences (including at either end) so that the resulting sequences x' and y' have the same length but do not have a space in the same position (i.e., for no position j are both $x'[j]$ and $y'[j]$ a space). Then we assign a "score" to each position. Position j receives a score as follows:

- $+1$ if $x'[j] = y'[j]$ and neither is a space,
- -1 if $x'[j] \neq y'[j]$ and neither is a space,
- -2 if either $x'[j]$ or $y'[j]$ is a space.

The score for the alignment is the sum of the scores of the individual positions. For example, given the sequences x = `GATCGGCAT` and y = `CAATGTGAATC`, one alignment is

```
G ATCG GCAT
CAAT GTGAATC
-*++*+*+-++*
```

A + under a position indicates a score of +1 for that position, a − indicates a score of −1, and a * indicates a score of −2, so that this alignment has a total score of $6 \cdot 1 - 2 \cdot 1 - 4 \cdot 2 = -4$.

b. Explain how to cast the problem of finding an optimal alignment as an edit-distance problem using a subset of the transformation operations copy, replace, delete, insert, twiddle, and kill.

14-6 Planning a company party

Professor Blutarsky is consulting for the president of a corporation that is planning a company party. The company has a hierarchical structure, that is, the supervisor relation forms a tree rooted at the president. The human resources department has ranked each employee with a conviviality rating, which is a real number. In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

Professor Blutarsky is given the tree that describes the structure of the corporation, using the left-child, right-sibling representation described in Section 10.3. Each node of the tree holds, in addition to the pointers, the name of an employee and that employee's conviviality ranking. Describe an algorithm to make up a guest list that maximizes the sum of the conviviality ratings of the guests. Analyze the running time of your algorithm.

14-7 Viterbi algorithm

Dynamic programming on a directed graph can play a part in speech recognition. A directed graph $G = (V, E)$ with labeled edges forms a formal model of a person speaking a restricted language. Each edge $(u, v) \in E$ is labeled with a sound $\sigma(u, v)$ from a finite set Σ of sounds. Each

directed path in the graph starting from a distinguished vertex $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model, with the label of a path being the concatenation of the labels of the edges on that path.

a. Describe an efficient algorithm that, given an edge-labeled directed graph G with distinguished vertex v_0 and a sequence $s = \langle \sigma_1, \sigma_2, \ldots, \sigma_k \rangle$ of sounds from Σ, returns a path in G that begins at v_0 and has s as its label, if any such path exists. Otherwise, the algorithm should return NO-SUCH-PATH. Analyze the running time of your algorithm. (*Hint:* You may find concepts from Chapter 20 useful.)

Now suppose that every edge $(u, v) \in E$ has an associated nonnegative probability $p(u, v)$ of being traversed, so that the corresponding sound is produced. The sum of the probabilities of the edges leaving any vertex equals 1. The probability of a path is defined to be the product of the probabilities of its edges. Think of the probability of a path beginning at vertex v_0 as the probability that a "random walk" beginning at v_0 follows the specified path, where the edge leaving a vertex u is taken randomly, according to the probabilities of the available edges leaving u.

b. Extend your answer to part (a) so that if a path is returned, it is a *most probable path* starting at vertex v_0 and having label s. Analyze the running time of your algorithm.

14-8 Image compression by seam carving

Suppose that you are given a color picture consisting of an $m \times n$ array $A[1 : m, 1 : n]$ of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. You want to compress this picture slightly, by removing one pixel from each of the m rows, so that the whole picture becomes one pixel narrower. To avoid incongruous visual effects, however, the pixels removed in two adjacent rows must lie in either the same column or adjacent columns. In this way, the pixels removed form a "seam" from the top row to the bottom row, where successive pixels in the seam are adjacent vertically or diagonally.

a. Show that the number of such possible seams grows at least exponentially in m, assuming that $n > 1$.

b. Suppose now that along with each pixel $A[i, j]$, you are given a real-valued disruption measure $d[i, j]$, indicating how disruptive it would be to remove pixel $A[i, j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Define the disruption measure of a seam as the sum of the disruption measures of its pixels.

Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

14-9 Breaking a string

A certain string-processing programming language allows you to break a string into two pieces. Because this operation copies the string, it costs n time units to break a string of n characters into two pieces. Suppose that you want to break a string into many pieces. The order in which the breaks occur can affect the total amount of time used. For example, suppose that you want to break a 20-character string after characters 2, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If you program the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units (breaking the string from characters 3 to 20 at character 8), and the third break costs 12 time units, totaling 50 time units. If you program the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break costs 10 time units, and the third break costs 8 time units, totaling 38 time units. In yet another order, you could break first at 8 (costing 20), then break the left piece at 2 (costing another 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

Design an algorithm that, given the numbers of characters after which to break, determines a least-cost way to sequence those breaks. More formally, given an array $L[1 : m]$ containing the break points for a string of n characters, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this cost.

14-10 Planning an investment strategy

Your knowledge of algorithms helps you obtain an exciting job with a hot startup, along with a \$10,000 signing bonus. You decide to invest this money with the goal of maximizing your return at the end of 10 years. You decide to use your investment manager, G. I. Luvcache, to manage your signing bonus. The company that Luvcache works with requires you to observe the following rules. It offers n different investments, numbered 1 through n. In each year j, investment i provides a return rate of r_{ij}. In other words, if you invest d dollars in investment i in year j, then at the end of year j, you have dr_{ij} dollars. The return rates are guaranteed, that is, you are given all the return rates for the next 10 years for each investment. You make investment decisions only once per year. At the end of each year, you can leave the money made in the previous year in the same investments, or you can shift money to other investments, by either shifting money between existing investments or moving money to a new investment. If you do not move your money between two consecutive years, you pay a fee of f_1 dollars, whereas if you switch your money, you pay a fee of f_2 dollars, where $f_2 > f_1$. You pay the fee once per year at the end of the year, and it is the same amount, f_2, whether you move money in and out of only one investment, or in and out of many investments.

a. The problem, as stated, allows you to invest your money in multiple investments in each year. Prove that there exists an optimal investment strategy that, in each year, puts all the money into a single investment. (Recall that an optimal investment strategy maximizes the amount of money after 10 years and is not concerned with any other objectives, such as minimizing risk.)

b. Prove that the problem of planning your optimal investment strategy exhibits optimal substructure.

c. Design an algorithm that plans your optimal investment strategy. What is the running time of your algorithm?

d. Suppose that Luvcache's company imposes the additional restriction that, at any point, you can have no more than \$15,000 in any one investment. Show that the problem of maximizing your income at the end of 10 years no longer exhibits optimal substructure.

14-11 Inventory planning

The Rinky Dink Company makes machines that resurface ice rinks. The demand for such products varies from month to month, and so the company needs to develop a strategy to plan its manufacturing given the fluctuating, but predictable, demand. The company wishes to design a plan for the next n months. For each month i, the company knows the demand d_i, that is, the number of machines that it will sell. Let $D = \sum_{i=1}^{n}$ be the total demand over the next n months. The company keeps a full-time staff who provide labor to manufacture up to m machines per month. If the company needs to make more than m machines in a given month, it can hire additional, part-time labor, at a cost that works out to c dollars per machine. Furthermore, if the company is holding any unsold machines at the end of a month, it must pay inventory costs. The company can hold up to D machines, with the cost for holding j machines given as a function $h(j)$ for $j = 1, 2, \ldots, D$ that monotonically increases with j.

Give an algorithm that calculates a plan for the company that minimizes its costs while fulfilling all the demand. The running time should be polynomial in n and D.

14-12 Signing free-agent baseball players

Suppose that you are the general manager for a major-league baseball team. During the off-season, you need to sign some free-agent players for your team. The team owner has given you a budget of \$$X$ to spend on free agents. You are allowed to spend less than \$$X$, but the owner will fire you if you spend any more than \$$X$.

You are considering N different positions, and for each position, P free-agent players who play that position are available.[10] Because you do not want to overload your roster with too many players at any position, for each position you may sign at most one free agent who

plays that position. (If you do not sign any players at a particular position, then you plan to stick with the players you already have at that position.)

To determine how valuable a player is going to be, you decide to use a sabermetric statistic[11] known as "WAR," or "wins above replacement." A player with a higher WAR is more valuable than a player with a lower WAR. It is not necessarily more expensive to sign a player with a higher WAR than a player with a lower WAR, because factors other than a player's value determine how much it costs to sign them.

For each available free-agent player p, you have three pieces of information:

- the player's position,
- $p.cost$, the amount of money it costs to sign the player, and
- $p.war$, the player's WAR.

Devise an algorithm that maximizes the total WAR of the players you sign while spending no more than $\$X$. You may assume that each player signs for a multiple of \$100,000. Your algorithm should output the total WAR of the players you sign, the total amount of money you spend, and a list of which players you sign. Analyze the running time and space requirement of your algorithm.

Chapter notes

Bellman [44] began the systematic study of dynamic programming in 1955, publishing a book about it in 1957. The word "programming," both here and in linear programming, refers to using a tabular solution method. Although optimization techniques incorporating elements of dynamic programming were known earlier, Bellman provided the area with a solid mathematical basis.

Galil and Park [172] classify dynamic-programming algorithms according to the size of the table and the number of other table entries each entry depends on. They call a dynamic-programming algorithm