The \(i\)th entry of column 9, that is, \(V[i, 9]\) contains the maximum value we can get by filling the knapsack using the first \(i\) items. Thus, an optimal packing is found in the last entry of the last column and is achieved by packing items 3 and 4. There is also another optimal solution, which is packing items 1, 2 and 3. This packing corresponds to entry \(V[3, 9]\) in the table, which is the optimal packing before the fourth item was considered.

### 7.7 Exercises

**7.1.** We have defined the dynamic programming paradigm in such a way that it encompasses all algorithms that solve a problem by breaking it down into smaller subproblems, saving the solution to each subproblem and using these solutions to compute an optimal solution to the main problem. Which of the following algorithms can be classified as dynamic programming algorithms?

(a) Algorithm `linearsearch`.
(b) Algorithm `insertionSort`.
(c) Algorithm `bottomUpSort`.
(d) Algorithm `mergesort`.

**7.2.** Give an efficient algorithm to compute \(f(n)\), the \(n\)th number in the Fibonacci sequence (see Example 7.1). What is the time complexity of your algorithm? Is it an exponential algorithm? Explain.

**7.3.** Give an efficient algorithm to compute the binomial coefficient \(\binom{n}{k}\) (see Example 7.2). What is the time complexity of your algorithm? Is it an exponential algorithm? Explain.

**7.4.** Prove Observation 7.1.

**7.5.** Use Algorithm `lcs` to find the length of a longest common subsequence of the two strings \(A = "xzyzyx"\) and \(B = "xxyyxxz"\). Give one longest common subsequence.

**7.6.** Show how to modify Algorithm `lcs` so that it outputs a longest common subsequence as well.

**7.7.** Show how to modify Algorithm `lcs` so that it requires only \(\Theta(\min\{m, n\})\) space.

**7.8.** In Sec. 7.3, it was shown that the number of ways to fully parenthesis
Exercises

221

The summation

\[ f(n) = \sum_{k=1}^{n-1} f(k)f(n-k). \]

Show that the solution to this recurrence is

\[ f(n) = \frac{1}{n} \left( \frac{2n-2}{n-1} \right). \]

7.9. Consider using Algorithm \textit{matchain} to multiply the following five matrices:

\[ M_1 : 4 \times 5, \ M_2 : 5 \times 3, \ M_3 : 3 \times 6, \ M_4 : 6 \times 4, \ M_5 : 4 \times 5. \]

Assume the intermediate results shown in Fig. 7.6 for obtaining the multiplication \( M_i \times \ldots \times M_j, 1 \leq i \leq j \leq 5 \). Also shown in the figure parenthesized expressions showing the optimal sequence for carrying out the multiplication \( M_i \times \ldots \times M_j \). Find \( C[1,5] \) and the optimal parenthesized expressions for carrying out the multiplication \( M_1 \times \ldots \times M_5 \).

\[
\begin{array}{|c|c|c|c|}
\hline
& C[1, 1] = 0 & C[1, 2] = 60 & C[1, 3] = 132 \\
\hline
M_1 & (M_1 M_2)M_3 & (M_1 M_2)(M_3 M_4) & C[1, 4] = 180 \\
\hline
M_2 & (M_2 M_3)M_4 & M_2(M_3 M_4) & C[2, 5] = 207 \\
\hline
M_3 & M_3 M_4 & M_3 M_4 & C[4, 5] = 120 \\
\hline
& C[4, 4] = 0 & C[4, 5] = 120 \\
M_4 & M_4 & M_4 & C[5, 5] = 0 \\
\hline
\end{array}
\]

Fig. 7.6 An incomplete table for the matrix chain multiplication problem.

7.10. Give a parenthesized expression for the optimal order of multiplying the five matrices in Example 7.4.

7.11. Consider applying Algorithm \textit{matchain} on the following five matrices:

\[ M_1 : 2 \times 3, \ M_2 : 3 \times 6, \ M_3 : 6 \times 4, \ M_4 : 4 \times 2, \ M_5 : 2 \times 7. \]
(a) Find the minimum number of scalar multiplications needed to multiply the five matrices, (that is $C[1,5]$).
(b) Give a parenthesized expression for the order in which this optimal number of multiplications is achieved.

7.12. Give an example of three matrices in which one order of their multiplication costs at least 100 times the other order.

7.13. Show how to modify the matrix chain multiplication algorithm so that it also produces the order of multiplications as well.

7.14. Let $G = (V,E)$ be a weighted directed graph, and let $s,t \in V$. Assume that there is at least one path from $s$ to $t$;
(a) Let $\pi$ be a path of shortest length from $s$ to $t$ that passes by another vertex $x$. Show that the portion of the path from $s$ to $x$ is a shortest path from $s$ to $x$.
(b) Let $\pi'$ be a longest simple path from $s$ to $t$ that passes by another vertex $y$. Show that the portion of the path from $s$ to $y$ is not necessarily a longest path from $s$ to $y$.

7.15. Run the all-pairs shortest path algorithm on the weighted directed graph shown in Fig. 7.7.

7.16. Use the all-pairs shortest path algorithm to compute the distance matrix for the directed graph with the lengths of the edges between all pairs of vertices are as given by the matrix

\[
\begin{bmatrix}
0 & 1 & \infty & 2 \\
2 & 0 & \infty & 2 \\
\infty & 9 & 0 & 4 \\
8 & 2 & 3 & 0
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 & 2 & 4 & 6 \\
2 & 0 & 1 & 2 \\
5 & 9 & 0 & 1 \\
9 & \infty & 2 & 0
\end{bmatrix}
\]

7.17. Give an example of a directed graph that contains some edges with negative costs and yet the all-pairs shortest path algorithm gives the correct distances.
7.18. Give an example of a directed graph that contains some edges with negative costs such that the all-pairs shortest path algorithm fails to give the correct distances.

7.19. Show how to modify the all-pairs shortest path algorithm so that it detects negative-weight cycles (A negative-weight cycle is a cycle whose total length is negative).

7.20. Prove Observation 7.2.

7.21. Solve the following instance of the knapsack problem. There are four items of sizes 2, 3, 5, and 6 and values 3, 4, 5, and 7, and the knapsack capacity is 11.

7.22. Solve the following instance of the knapsack problem. There are five items of sizes 3, 5, 7, 8 and 9 and values 4, 6, 7, 9 and 10, and the knapsack capacity is 22.

7.23. Explain what would happen when running the knapsack algorithm on an input in which one item has negative size.

7.24. Show how to modify Algorithm \textsc{knapsack} so that it requires only $\Theta(C)$ space, where $C$ is the knapsack capacity.

7.25. Show how to modify Algorithm \textsc{knapsack} so that it outputs the items packed in the knapsack as well.

7.26. In order to lower the prohibitive running time of the knapsack problem, which is $\Theta(nC)$, we may divide $C$ and all the $s_i$’s by a large number $K$ and take the floor. That is, we may transform the given instance into a new instance with capacity $\lfloor C/K \rfloor$ and item sizes $\lfloor s_i/K \rfloor$, $1 \leq i \leq n$. Now, we apply the algorithm for the knapsack discussed in Sec. 7.6. This technique is called scaling and rounding (see Sec. 15.6). What will be the running time of the algorithm when applied to the new instance? Give a counterexample to show that scaling and rounding does not always result in an optimal solution to the original instance.

7.27. Another version of the knapsack problem is to let the set $U$ contain a set of types of items, and the objective is to fill the knapsack with any number of items of each type in order to maximize the total value without exceeding the knapsack capacity. Assume that there is an unlimited number of items of each type. More formally, let $T = \{t_1, t_2, \ldots, t_n\}$ be a set of $n$ types of items, and $C$ the knapsack capacity. For $1 \leq j \leq n$, let $s_j$ and $v_j$ be, respectively, the size and value of the items of type $j$. Find a set of nonnegative integers $x_1, x_2, \ldots, x_n$ such that

$$\sum_{i=1}^{n} x_i v_i$$
is maximized subject to the constraint
\[ \sum_{i=1}^{n} x_i s_i \leq C. \]

Note that \( x_j = 0 \) means that no item of the \( j \)th type is packed in the knapsack. Rewrite the dynamic programming algorithm for this version of the knapsack problem.

7.28. Solve the following instance of the version of the knapsack problem described in Exercise 7.27. There are five types of items with sizes 2, 3, 5 and 6 and values 4, 7, 9 and 11, and the knapsack capacity is 8.

7.29. Show how to modify the knapsack algorithm discussed in Exercise 7.27 so that it computes the number of items packed from each type.

7.30. Consider the money change problem. We have a currency system that has \( n \) coins with values \( v_1, v_2, \ldots, v_n \), where \( v_1 = 1 \), and we want to pay change of value \( y \) in such a way that the total number of coins is minimized. More formally, we want to minimize the quantity
\[ \sum_{i=1}^{n} x_i \]
subject to the constraint
\[ \sum_{i=1}^{n} x_i v_i = y. \]

Here, \( x_1, x_2, \ldots, x_n \) are nonnegative integers (so \( x_i \) may be zero).

(a) Give a dynamic programming algorithm to solve this problem.
(b) What are the time and space complexities of your algorithm?
(c) Can you see the resemblance of this problem to the version of the knapsack problem discussed in Exercise 7.27? Explain.

7.31. Apply the algorithm in Exercise 7.30 to the instance \( v_1 = 1, v_2 = 5, v_3 = 7, v_4 = 11 \) and \( y = 20 \).

7.32. Let \( G = (V, E) \) be a directed graph with \( n \) vertices. \( G \) induces a relation \( R \) on the set of vertices \( V \) defined by: \( u R v \) if and only if there is a directed edge from \( u \) to \( v \), i.e., if and only if \( (u, v) \in E \). Let \( M_R \) be the adjacency matrix of \( G \), i.e., \( M_R \) is an \( n \times n \) matrix satisfying \( M_R[u, v] = 1 \) if \( (u, v) \in E \) and 0 otherwise. The reflexive and transitive closure of \( M_R \), denoted by \( M_R^* \), is defined as follows. For \( u, v \in V \), if \( u = v \) or there is a path in \( G \) from \( u \) to \( v \), then \( M_R^*[u, v] = 1 \) and 0 otherwise. Give a dynamic programming algorithm to compute \( M_R^* \) for a given directed
graph. (Hint: You only need a slight modification of Floyd’s algorithm for the all-pairs shortest path problem).

**7.33.** Let $G = (V, E)$ be a directed graph with $n$ vertices. Define the $n \times n$ distance matrix $D$ as follows. For $u, v \in V$, $D[u, v] = d$ if and only if the length of the shortest path from $u$ to $v$ measured in the number of edges is exactly $d$. For example, for any $v \in V$, $D[v, v] = 0$ and for any $u, v \in V$ $D[u, v] = 1$ if and only if $(u, v) \in E$. Give a dynamic programming algorithm to compute the distance matrix $D$ for a given directed graph. (Hint: Again, you only need a slight modification of Floyd’s algorithm for the all-pairs shortest path problem).

**7.34.** Let $G = (V, E)$ be a directed acyclic graph (dag) with $n$ vertices. Let $s$ and $t$ be two vertices in $V$ such that the indegree of $s$ is 0 and the outdegree of $t$ is 0. Give a dynamic programming algorithm to compute a longest path in $G$ from $s$ to $t$. What is the time complexity of your algorithm?

**7.35.** Give a dynamic programming algorithm for the traveling salesman problem: Given a set of $n$ cities with their intercity distances, find a tour of minimum length. Here, a tour is a cycle that visits each city exactly once. What is the time complexity of your algorithm? This problem can be solved using dynamic programming in time $O(n^2 2^n)$ (see the bibliographic notes).

**7.36.** Let $P$ be a convex polygon with $n$ vertices (see Sec. 18.2). A chord in $P$ is a line segment that connects two nonadjacent vertices in $P$. The problem of triangulating a convex polygon is to partition the polygon into $n - 2$ triangles by drawing $n - 3$ chords inside $P$. Figure 7.8 shows two possible triangulations of the same convex polygon.

![Fig. 7.8 Two triangulations of the same convex polygon.](image)

(a) Show that the number of ways to triangulate a convex polygon with $n$ vertices is the same as the number of ways to multiply $n - 1$ matrices.

(b) A minimum weight triangulation is a triangulation in which the sum of the lengths of the $n - 3$ chords is minimum. Give a dynamic programming algorithm for finding a minimum weight triangulation of a convex polygon with $n$ vertices. (Hint: This problem is very similar to the matrix chain multiplication covered in Sec. 7.3).
Dynamic programming was first popularized in the book by Bellman (1957). Other books in this area include Bellman and Dreyfus (1962), Dreyfus (1977) and Nemhauser (1966). Two general survey papers by Brown (1979) and Held and Karp (1967) are highly recommended. The all-pairs shortest paths algorithm is due to Floyd (1962). Matrix chain multiplication is described in Godbole (1973). An $O(n \log n)$ algorithm to solve this problem can be found in Hu and Shing (1980, 1982, 1984). The one and two dimensional knapsack problems have been studied extensively; see for example Gilmore (1977), Gilmore and Gomory (1966) and Hu (1969). Held and Karp (1962) gave an $O(n^2 2^n)$ dynamic programming algorithm for the traveling salesman problem. This algorithm also appears in Horowitz and Sahni (1978).