Good problems

[DPV] 6.2. You are going on a long trip. You start on the road at mile post 0. Along the way there are n hotels, at mile posts $a_1 < a_2 < \cdots < a_n$, where each a_i is measured from the starting point. The only places you are allowed to stop are at these hotels, but you can choose which of the hotels you stop at. You must stop at the final hotel (at distance a_n), which is your destination.

You'd ideally like to travel 200 miles a day, but this may not be possible (depending on the spacing of the hotels). If you travel x miles during a day, the *penalty* for that day is $(200 - x)^2$. You want to plan your trip so as to minimize the total penalty—that is, the sum, over all travel days, of the daily penalties.

Give an efficient algorithm that determines the optimal sequence of hotels at which to stop.

[DPV] 6.4. You are given a string of n characters $s[1 \ldots n]$, which you believe to be a corrupted text document in which all punctuation has vanished (so that it looks something like "itwasthebestoftimes..."). You wish to reconstruct the document using a dictionary, which is available in the form of a Boolean function `dict`$(\cdot)$: for any string w,

$$\texttt{dict}(w) = \begin{cases} \texttt{true} & \text{if } w \text{ is a valid word} \\ \texttt{false} & \text{otherwise .} \end{cases}$$

(a) Give a dynamic programming algorithm that determines whether the string $s[\cdot]$ can be reconstituted as a sequence of valid words. The running time should be at most $O(n^2)$, assuming calls to `dict` take unit time.

(b) In the event that the string is valid, make your algorithm output the corresponding sequence of words.

[DPV] 6.7. A subsequence is *palindromic* if it is the same whether read left to right or right to left. For instance, the sequence

$$A, C, G, T, G, T, C, A, A, A, A, T, C, G$$

has many palindromic subsequences, including A, C, G, C, A and A, A, A, A (on the other hand, the subsequence A, C, T is *not* palindromic). Devise an algorithm that takes a sequence $x[1 \ldots n]$ and returns the (length of the) longest palindromic subsequence. Its running time should be $O(n^2)$.

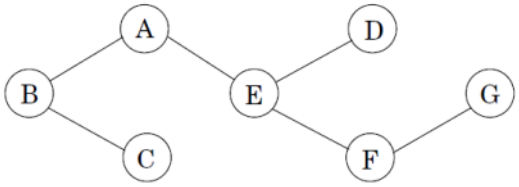
[DPV] 6.9. A certain string-processing language offers a primitive operation which splits a string into two pieces. Since this operation involves copying the original string, it takes n units of time for a string of length n, regardless of the location of the cut. Suppose, now, that you want to break a string into many pieces. The order in which the breaks are made can affect the total running time. For example, if you want to cut a 20-character string at positions 3 and 10, then making the first cut at position 3 incurs a total cost of $20 + 17 = 37$, while doing position 10 first has a better cost of $20 + 10 = 30$.

Give a dynamic programming algorithm that, given the locations of m cuts in a string of length n, finds the minimum cost of breaking the string into $m + 1$ pieces.

[DPV] 6.21. A *vertex cover* of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ that includes at least one endpoint of every edge in E. Give a linear-time algorithm for the following task.

Input: An undirected tree $T = (V, E)$.
Output: The size of the smallest vertex cover of T.

For instance, in the following tree, possible vertex covers include $\{A, B, C, D, E, F, G\}$ and $\{A, C, D, F\}$ but not $\{C, E, F\}$. The smallest vertex cover has size 3: $\{B, E, G\}$.



[DPV] 6.26. *Sequence alignment.* When a new gene is discovered, a standard approach to understanding its function is to look through a database of known genes and find close matches. The closeness of two genes is measured by the extent to which they are *aligned*. To formalize this, think of a gene as being a long string over an alphabet $\Sigma = \{A, C, G, T\}$. Consider two genes (strings) $x = ATGCC$ and $y = TACGCA$. An alignment of x and y is a way of matching up these two strings by writing them in columns, for instance:

$$\begin{array}{cccccccc} - & A & T & - & G & C & C \\ T & A & - & C & G & C & A \end{array}$$

Here the "–" indicates a "gap." The characters of each string must appear in order, and each column must contain a character from at least one of the strings. The score of an alignment is specified by a scoring matrix δ of size $(|\Sigma| + 1) \times (|\Sigma| + 1)$, where the extra row and column are to accommodate gaps. For instance the preceding alignment has the following score:

$$\delta(-, T) + \delta(A, A) + \delta(T, -) + \delta(-, C) + \delta(G, G) + \delta(C, C) + \delta(C, A).$$

Give a dynamic programming algorithm that takes as input two strings $x[1 \ldots n]$ and $y[1 \ldots m]$ and a scoring matrix δ, and returns the highest-scoring alignment. The running time should be $O(mn)$.

[CLRS] ***15-4 Printing neatly***

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \le j$, and we leave exactly one space between words, 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. We wish 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 on a printer. Analyze the running time and space requirements of your algorithm.

Interesting problems

[DPV] 6.10. *Counting heads.* Given integers n and k, along with $p_1, \ldots, p_n \in [0, 1]$, you want to determine the

[DPV] 6.10. *Counting heads.* Given integers n and k, along with $p_1, \ldots, p_n \in [0, 1]$, you want to determine the probability of obtaining exactly k heads when n biased coins are tossed independently at random, where p_i is the probability that the ith coin comes up heads. Give an $O(n^2)$ algorithm for this task.[2] Assume you can multiply and add two numbers in $[0, 1]$ in $O(1)$ time.

[2]In fact, there is also a $O(n \log^2 n)$ algorithm within your reach.

[DPV] 6.12. You are given a convex polygon P on n vertices in the plane (specified by their x and y coordinates). A *triangulation* of P is a collection of $n - 3$ diagonals of P such that no two diagonals intersect (except possibly at their endpoints). Notice that a triangulation splits the polygon's interior into $n - 2$ disjoint triangles. The cost of a triangulation is the sum of the lengths of the diagonals in it. Give an efficient algorithm for finding a triangulation of minimum cost. (*Hint:* Label the vertices of P by $1, \ldots, n$, starting from an arbitrary vertex and walking clockwise. For $1 \leq i < j \leq n$, let the subproblem $A(i, j)$ denote the minimum cost triangulation of the polygon spanned by vertices $i, i + 1, \ldots, j$.)

[DPV] 6.16. The *garage sale problem* (courtesy of Professor Lofti Zadeh). On a given Sunday morning, there are n garage sales going on, $g_1, g_2, \ldots, g_n$. For each garage sale g_j, you have an estimate of its value to you, v_j. For any two garage sales you have an estimate of the transportation cost d_{ij} of getting from g_i to g_j. You are also given the costs d_{0j} and d_{j0} of going between your home and each garage sale. You want to find a tour of a *subset* of the given garage sales, starting and ending at home, that maximizes your total benefit minus your total transportation costs.

Give an algorithm that solves this problem in time $O(n^2 2^n)$. (*Hint:* This is closely related to the traveling salesman problem.)

[DPV] 6.25. Consider the following 3-PARTITION problem. Given integers $a_1, \ldots, a_n$, we want to determine whether it is possible to partition of $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{1}{3} \sum_{i=1}^{n} a_i$$

For example, for input $(1, 2, 3, 4, 4, 5, 8)$ the answer is *yes*, because there is the partition $(1, 8)$, $(4, 5)$, $(2, 3, 4)$. On the other hand, for input $(2, 2, 3, 5)$ the answer is *no*.

Devise and analyze a dynamic programming algorithm for 3-PARTITION that runs in time polynomial in n and in $\sum_i a_i$.

[CLRS] ***15-3 Bitonic euclidean traveling-salesman problem***

In the ***euclidean traveling-salesman problem***, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. Figure 15.11(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 that we simplify the problem by restricting our attention to ***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 15.11(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.)

Challenging problems (optional)

[DPV] 6.30. *Reconstructing evolutionary trees by maximum parsimony.* Suppose we manage to sequence a particular gene across a whole bunch of different species. For concreteness, say there are n species, and the sequences are strings of length k over alphabet $\Sigma = \{A, C, G, T\}$. How can we use this information to reconstruct the evolutionary history of these species?

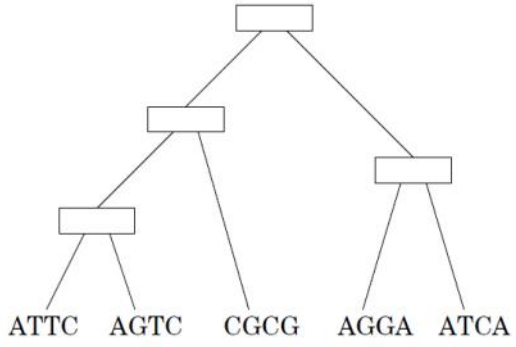
Evolutionary history is commonly represented by a tree whose leaves are the different species, whose root is their common ancestor, and whose internal branches represent speciation events (that is, moments when a new species broke off from an existing one). Thus we need to find the following:

- An evolutionary tree with the given species at the leaves.
- For each internal node, a string of length k: the gene sequence for that particular ancestor.

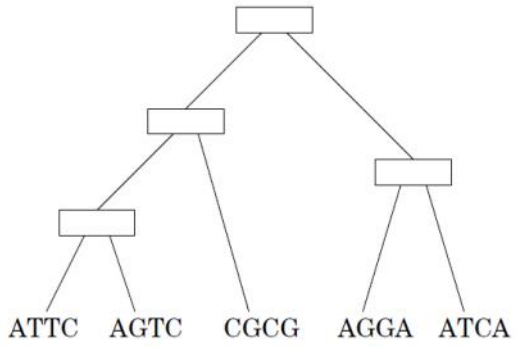
For each possible tree T, annotated with sequences $s(u) \in \Sigma^k$ at each of its nodes u, we can assign a score based on the principle of *parsimony*: fewer mutations are more likely.

$$\text{score}(T) = \sum_{(u,v)\in E(T)} (\text{number of positions on which } s(u) \text{ and } s(v) \text{ disagree}).$$

Finding the highest-score tree is a difficult problem. Here we will consider just a small part of it: suppose we know the structure of the tree, and we want to fill in the sequences $s(u)$ of the internal nodes u. Here's an example with $k = 4$ and $n = 5$:



(a) In this particular example, there are several maximum parsimony reconstructions of the internal node sequences. Find one of them.



(a) In this particular example, there are several maximum parsimony reconstructions of the internal node sequences. Find one of them.

(b) Give an efficient (in terms of n and k) algorithm for this task. (*Hint:* Even though the sequences might be long, you can do just one position at a time.)