WERKCOLLEGE 9

You are allowed to answer in Dutch. Whenever an algorithm is required, it can be given in pseudocode or plain English (or Dutch), and its running time and correctness must always be justified (even informally, but in a clear way!).

9.1. A company wants to open a series of shops along a highway. The n possible locations are along a straight line, and the distances of these locations from the start of the highway are, in kilometers and in increasing order $m_1, m_2, \ldots, m_n$. The constraints are as follows:

- At any location the company may open at most one shop. The expected profit from opening a shop at location i is p_i, where $p_i > 0$ and $i = 1, 2, \ldots, n$.
- Any two shops should be at least k kilometers apart, where k is a positive integer.

Give an efficient algorithm to compute the maximum expected total profit subject to the given constraints.

9.2. Let M be a $m \times n$ matrix of natural numbers and let C be a natural number. A C-path of M is a sequence of the form $M[i_1, j_1], M[i_2, j_2], \ldots, M[i_l, j_l]$, with $l \geq 1$, such that:

a) the sum of its elements is C;
b) for any two consecutive elements $M[i_k, j_k]$ and $M[i_{k+1}, j_{k+1}]$, either $i_{k+1} = i_k + 1$ and $j_{k+1} = j_k$; or $i_{k+1} = i_k$ and $j_{k+1} = j_k + 1$.

In words, a C-path is a non-empty path of sum C through the matrix: at each step it either goes one cell down, or one cell to the right.

We want to write a dynamic programming algorithm that computes the number of C-paths from $M[0, 0]$ to $M[m, n]$. For example, for

$$M = \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 6 & 5 \\ \hline 3 & 2 & 1 \\ \hline \end{array} \qquad C = 12$$

we have two such C-paths: $1, 2, 6, 2, 1$ and $1, 2, 3, 5, 1$.

a) Let $P[i, j, k]$ be the number of k-paths from $M[0, 0]$ to $M[i, j]$. Explain how $P[i, j, k]$ can be recursively computed.
b) Give a bottom-up, non-recursive algorithm that returns the number of C-paths from $M[0, 0]$ to $M[m, n]$ in $\mathcal{O}(nmC)$ time. The algorithm should rely on the bottom-up computation of $P[i, j, k]$ in *a*).

9.3. A palindrome is a non-empty 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). A *subsequence* of a string s is a string that can be derived from s by deleting some elements without changing the order of the remaining elements. E.g., `acd` is a subsequence of `abcde`. We want to solve the following problem: given a string s of length n, find the **length** of the longest palindrome that is a subsequence of s. For example, given the input `character`, the output should be 5, i.e., the length of `carac`.

a) Let $L[i, j]$ be the length of the longest palindromic subsequence of $s[i, \ldots, j]$. Explain how $L[i, j]$ can be recursively computed. (Hint: if $s[i] = s[j]$, we have found a palindrome subsequence of length 2; then we have to look for palindromes in $s[i + 1, \ldots, j - 1]$).

b) We want to use bottom-up dynamic programming to solve the problem in $\mathcal{O}(n^2)$ time. Give a non-recursive, bottom-up algorithm that returns the length of the longest palindrome subsequence of a given string s. The algorithm should rely on the bottom-up computation of $L[i, j]$ of point *b*).