Algorithms and programming
École normale supérieure - Département d'informatique
TD 3: Dynamic programming and greedy algorithms

---

1. **(Binomial coefficients)** Let k be a fixed integer. Show that $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$. Using this recursion, develop two algorithms to compute $\binom{n}{k}$ and show their complexities :
   (a) a recursive algorithm ;
   (b) a dynamic programming algorithm.
2. **(Scheduling)** Suppose we have a set of ENS sport activities that wish to use a gym. The gym can be used by only one activity at a time. Each activity i starts at time s_i and finishes at time f_i, where $s_i < f_i$. Activities i and j are *compatible* if the intervals $[s_i, f_i)$ and $[s_j, f_j)$ do not overlap. The scheduling problem is to select a maximum-size set of mutually compatible activities. We assume that $f_1 \leq f_2 \leq \cdots \leq f_n$.
   (a) Show correctness and complexity of the following algorithm.

Algorithm 1 Greedy algorithm for scheduling

```
A ← {1}
j ← 1
for i ← 2 to n do
    if s_i ≥ f_j then
        A ← A ∪ {i}
        j ← i
    end if
end for
return A
```

   Assume now that each activity i has a weight w_i and that we want to select a maximum weighted set of mutually compatible activities where the *weight of a set* is the sum of the weight of each selected activity.
   (b) Give example(s) showing that greedy algorithm doesn't work in this setting.
   (c) Develop a dynamic programming algorithm for this problem. The algorithm must run in $O(n^2)$ time and $O(n)$ space.
3. **(Matrix-chain multiplication)** We are given a sequence (chain) $A_1, A_2, \ldots, A_n$ of n matrices to be multiplied, and we wish to compute the product $A_1 A_2 \ldots A_n$. We can evaluate the expression $A_1 A_2 \ldots A_n$ using the standard algorithm for multiplying two matrices as a procedure once we have parenthesized it.
   (a) Show that the way we parenthesize a chain of matrices can change the running time of the algorithm.
   (b) Show that the way we parenthesize a chain of matrices does not change the value of the resulting expression.

From now on, our task is to find a parenthesization that gives the best running time. A naive way would be to try all possible parenthesizations. Let us show that their number is too large.

(c) Professor Dumbledore believes that there are $n!$ different ways to parenthesize a chain of n matrices. His argument is as follows : We can choose the first product to perform in n ways, the second in $n-1$ ways, and so on, therefore there are $n!$ different parenthesizations. Show that he is wrong by a counterexample.

(d) Give a recursive formula for the number of different parenthesizations. Use the formula to show that a chain of n matrices can be parenthesized in $b_n = \frac{1}{n+1}\binom{2n}{n}$ different ways (b_n is the n^{th} Cathalan number). Show that b_n grows exponentially.

(e) As we have proved, the naive algorithm has exponential running time. Use the dynamic programming approach to develop an algorithm with $O(n^3)$ running time.

4. **(Huffman codes)** A *binary character code* replaces each character with a binary string. We call a code in which no codeword is a prefix of some other codeword *prefix code*. Huffman codes are binary character, prefix codes. The Huffman code of a string S depends on the set of characters occurring in it and their frequencies, and is defined by the following process. The process first constructs a binary tree. The leaf nodes of the tree correspond to the characters of the alphabet. The weight of each node is defined as the frequency of the character is represents. Then, the process takes the two nodes with smallest weights, and creates a new node having these two nodes as children. The weight of the new node is set to the sum of the weight of the children. We then apply the process again, on the new node and on the remaining nodes. We repeat this process until only one node remains, which will be the root of the tree. Consider a character e and the path from the root to the leaf node representing e. The codeword for e is defined by this path : each time we turn left, we write "0", and each time we turn right, we write "1".

(a) What is an optimal Huffman code for the following set of frequencies : a :1 b :1 c :2 d :3 e :5 f :8 g :13 h :21

We will now prove optimality of Huffman codes.

(b) Prove that a binary tree that is not full cannot correspond to an optimal prefix code.

(c) Let C be an alphabet in which each character $e \in C$ has frequency $f[e]$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

(d) Prove : Let T be a full binary tree representing an optimal prefix code over an alphabet C, where frequency $f[c]$ is defined for each character $c \in C$. Consider any two characters x and y that appear as sibling leaves in T, and let z be their parent. Then, considering Z as a character with frequency $f[z] = f[x] + f[y]$, the

tree $T' = T - \{x, y\}$ represents an optimal prefix code for the alphabet $C' = C - \{x, y\} \cup \{z\}$.

(e) Show that no compression scheme can expect to compress a file of randomly chosen 8-bit characters by even a single bit.

5. **(Longest common subsequence)** A subsequence of a given sequence is just the given sequence with some elements (possibly none) left out. Formally, given a sequence $X = (x_1, x_2, \ldots, x_m)$, another sequence $Z = (z_1, z_2, \ldots, z_k)$ is a subsequence of X if there exists a strictly increasing sequence $(i_1, i_2, ..., i_k)$ of indices of X such that for all $j = 1, 2, \ldots, k$, we have $x_{i_j} = z_j$. Given two sequences X and Y, we say that a sequence Z is a common subsequence of X and Y if Z is a subsequence of both X and Y. In the longest-common-subsequence problem, we are given two sequences $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_n)$ and wish to find a maximum length common subsequence of X and Y.

   (a) Let us start with an easier problem. Suppose that we wish to compute the length of such subsequence (but not the subsequence itself). Develop a dynamic programming solution for this problem with running time $O(mn)$ and space $O(mn)$.

   (b) Can you augment the dynamic programming table with additional information to compute the longest common subsequence as well ?

   (c) Can you improve the space of the algorithm in (a) to $O(\min(m, n))$?

6. **(Edit distance)** The edit distance between two strings S, T is defined as the minimum number of edit operations that are required in order to transform S to T. The edit operations are : substitution (replace one letter of S by another letter), deletion (delete one letter of S), or insertion (insert one letter). Develop a dynamic programming algorithm that computes the edit distance between S and T in $O(mn)$ time, where m, n are lengths of S and T respectively.

7. **(RNA secondary structure)** Ribonucleic acid molecule (RNA) can be represented as a string $\mathcal{R} \in \{A, C, G, U\}^n$. To predict its interaction with other molecules, bioinformaticians are interested in so-called secondary structure of RNA. A secondary structure on $\mathcal{R}$ is a set S of pairs $(\mathcal{R}[i], \mathcal{R}[j])$, $1 \leq i, j \leq n$, satisfying the following conditions :

   (a) (Watson-Crick) The elements in each pair in S consist of either $\{A, U\}$ or $\{C, G\}$ (in either order) ;

   (b) S is a matching : no index appears in more than one pair ;

   (c) (no knots) If (i, j) and (k, l) are two pairs in S, then we cannot have $i < k < j < l$.

   A secondary structure is optimal if it contains the largest possible number of pairs. Develop a dynamic programming algorithm to compute an optimal secondary structure on $\mathcal{R}$.