**Algorithms and Complexity. Exercise session 3+4**

**Dynamic Programming**

**Longest Common Substring** The string `ALGORITHM` and the string `PLÅGORIS` have the common substring `GORI`. The *longest common substring* of these strings has thus length 4. The letters in a substring must form a coherent sequence.

Construct an efficient algorithm that given two strings $a_1a_2 \ldots a_m$ and $b_1b_2 \ldots b_n$ calculates the length of the longest common substring. The algorithm uses dynamic programming and runs in time $O(nm)$.

---

**Sequences** You are given two sequences of positive integers $a_1, a_2, \ldots, a_n$ and $b_1, b_2, \ldots, b_n$, where all numbers are less than $n^2$, and a positive integer $B$, such that $B \leq n^3$. The problem is to determine if there is a sequence $c_1, c_2, \ldots, c_n$ such that $\sum_{i=1}^{n} c_i = B$ and $c_i = a_i$ or $c_i = b_i$ for $1 \leq i \leq n$.

Describe and analyze an algorithm that solves the problem using dynamic programming. Moreover, describe how to extend the algorithm to construct the solution, where $c_i = a_i$ or $c_i = b_i$ for $1 \leq i \leq n$.

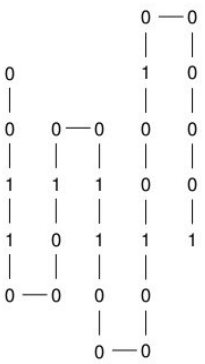
---

**Protein Folding** A protein is a long chain of aminoacids. The protein chain is not straight but it is folded in a way that minimizes the potential energy. Our goal is to calculate how the protein will fold itself. In this exercise we will therefore consider a simple model of protein folding in which aminoacids are either *hydrophobic* or *hydrophilic*. Hydrophobic aminoacids tend to clump together.

For simplicity, we can see the protein as a binary string in which ones correspond to hydrophobic aminoacids and zeros hydrophilic aminoacids. The string (protein) should then be folded into a two-dimensional square grid. The goal is to make the hydrophobic aminoacids stick together, i.e., to get as many ones as possible to be close to each other. Hence we have an optimization problem where the objective function is the number of pairs of ones that are next to each other in the grid (vertically or horizontally) without being next to each other in the string.

You should design an algorithm using dynamic programming to construct an optimal *accordion fold* of a given protein string of length $n$. An *accordion fold* is a fold where the first string goes straight down, then goes straight up, then goes straight down, and so on. In such a fold, it can be observed that the vertical pairs of adjacent ones will always result in the string, so it's only horizontal pair of ones that contribute to the objective function.

The following figure shows the string 001100010011000010010000001 of accordion fold in such a way that the objective function is 4.



Problem definition PROTEIN ACCORDION FOLD:

INPUT A binary string of $n$ characters.

PROBLEM: Find the accordion fold of input string that provides the greatest value to the objective function, ie the largest number of pairs of ones located next to each other, but not consecutive to each other in the string.

Construct and analyze the time complexity of an algorithm that solves protein accordion folding problem with dynamic programming.

Use the following algorithm which calculates the number of pairs of ones in a row (ie between two lines) lying next to each other (but not next to each other in the string). Suppose the protein is stored in an array `p[1..n]`. The parameters `a` and `b` indicate the index in the array for the first trait endpoints. The parameter `c` indicates the index for the second trait endpoint as dipicted in the figure below on the right.

```
profit(a,b,c) =
shortest←min(b-a,c-(b+1));
s←0;
for i←1 to shortest do
if p[b-i]=1 and p[b+1+i]=1 then
s←s+1;
return s;
```


Note: Protein folding is an important algorithmic problem studied in bioinformatics. Similar problems are studied in the *Algoritmisk bioinformatik* course.

---

**Analyzer for context-free grammars** A context-free grammar is usually used to describe the syntax of a particular programming language. A context-free grammar in *Chomsky Normal Form* is described by

- a set of final symbols $T$ (usually written in small letters),
- a set of non-final symbols $N$ (usually written in capital letters),
- the initial symbol $S$ (a non-final symbol in $N$),
- a set of rewrite rules of type $A \to BC$ or $A \to a$, for $A, B, C \in N$ and $a \in T$.

If $A \in N$ we define $\mathcal{L}(A)$ as

$$\mathcal{L}(A) = \{bc : b \in \mathcal{L}(B) \text{ and } c \in \mathcal{L}(C) \text{ where} A \to BC\} \cup \{a : A \to a\}.$$

*The language generated by the grammar* is then defined as $\mathcal{L}(S)$, ie. the set of all strings of the final symbols that can be formed by a rewriting chain from symbol $S$.

Example: Consider the grammar $T = \{a, b\}$, $N = \{S, A, B, R\}$, start symbol $S$ and rules $S \to AR$, $S \to AB$, $A \to a$, $B \to b$, $R \to SB$. We can see that string $aabb$ is in the language generated by the grammar using the following chain of rewritings:

$$S \to AR \to aR \to aSB \to aSb \to aABb \to aaBb \to aabb.$$

In fact, one can show that the language generated by the grammar is the set of all strings consisting of $k$ symbols $a$ followed by $k$ symbols $b$, where $k$ is a positive integer.

Your task is to *design* and *analyze* an efficient algorithm that determines if a string is in the language generated by a given grammar. The input is a context-free grammar in Chomsky Normal Form, and a string of final symbols. The output is true or false depending on whether the string can be generated by the grammar or not. Calculate the time complexity of your algorithm in terms of number of rules $m$ of the grammar and length $n$ of the string.

You can read more on grammars in the course *Artificiella språk och syntaxanalys*.

---