Plus longue sous-suite commune à deux suites de caractères **T.D.2.**

Étant donnée une suite de caractères $X = x_1, x_2, \ldots, x_n$, une autre suite de caractères $Y = y_1, y_2, \ldots, y_p$ est une sous-suite de X s'il existe des indices $i_1, \ldots, i_p$ tels que $1 \leqslant i_1 < i_2 < \ldots < i_p \leqslant n$ et tels que $\forall\, k \in [\![1, p]\!]$, $x_{i_k} = y_k$.
Par exemple Y =a,g,i,e,s est une sous-suite de X =a,l,g,o,r,i,t,h,m,e,s

1. Écrire deux versions, une itérative, l'autre récursive de la fonction booléenne `est_sous_suite` qui prend comme argument deux suites de caractères X et Y et qui renvoie la valeur `vrai` si Y est une sous-suite de X et `false` sinon.

   ```
   est_sous_suite : string -> string -> bool = <fun>
   ```

   Préciser le coût en comparaisons, dans le pire des cas de ces fonctions.

   On s'intéresse maintenant à la détermination d'une plus longue sous-suite commune (`plssc`) à deux suites X et Y données.

2. (a) Proposer en quelques mots un algorithme naïf, qui utilise la fonction booléenne `est_sous_suite` permettant de déterminer une plus longue sous-suite commune à deux suites X et Y.

   (b) Quel est l'ordre de grandeur du temps d'exécution d'un tel algorithme.

3. Si Z est une plus longue sous-suite commune à deux suites X et Y, on démontre que Z possède les propriétés suivantes :
   — Si $x_1 = y_1$, alors la liste Z privée de son premier caractère est une plus longue sous-suite commune à X et Y privées toutes deux de leur premier caractère
   — Si $x_1 \neq y_1$ et $x_1 \neq z_1$, alors Z est une plus longue sous-suite commune à X privée de son premier caractère et Y
   — Si $x_1 \neq y_1$ et $y_1 \neq z_1$, alors Z est une plus longue sous-suite commune à X et Y privée de son premier caractère

   (a) Déduire de ces propriétés l'écriture de la fonction récursive `plssc` qui prend comme arguments deux suites X et Y et qui renvoie pour résultat une plus longue sous-suite commune à X et Y.

   ```
   plssc : string -> string -> string = <fun>
   ```

   (b) Quelle est l'ordre de grandeur du nombre de comparaisons de caractères requis par cette fonction ?

   (c) Comment améliorer le résultat précédent ?

Rappel de quelques fonctions de la bibliothèque CAML
`Char.escaped : char -> string` renvoie une chaîne composée du caractère donné
`String.length : string -> int` renvoie la longueur de la chaîne donnée
`s.[n]` renvoie le caractère numéro `n` de la chaine `s` .
`String.sub : string -> int -> int -> string` . `String.sub s d l` renvoie une nouvelle chaine de caractères de longueur `l` contenant les caractères de la chaine `s` à partir du numéro `d`
`^ : string -> string -> string` . `s1^s2` renvoie une nouvelle chaîne concaténée des chaînes `s1` et `s2`
`@ : 'a list -> 'a list -> 'a list` . `l1@l2` renvoie une nouvelle liste concaténée des listes `l1` et `l2`
`List.length : list -> int` renvoie le nombre d'éléments de la liste donnée