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

1. Version itérative 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.

```
let est_sous_suite x y =
   let n= String.length x and p = String.length y
       and i = ref 0 and j =ref 0 in
   while (!i < n) && (!j < p) do
      if x.[!i] = y.[!j] then
           begin i := !i + 1; j := !j + 1 end
      else i := !i + 1
   done;
   !j = p  ;;
```

Voici une première version récursive de cette fonction `est_sous_suite` :

```
let rec est_sous_suite x y = match (x,y) with
  |_ , "" -> true
  |"" , _ -> false
  |x , y when (x.[0]= y.[0] ->
              est_sous_suite (String.sub x 1 (String.length x - 1))
                             (String.sub y 1 (String.length y - 1))
  |_ , _  -> est_sous_suite (String.sub x 1 (String.length x - 1)) y;;
```

En voici une seconde version qui n'est pas récursive mais utilise une fonction locale récursive et qui a l'avantage de ne pas utiliser `String.sub` :

```
let est_sous_suite x y =
  let n = String.length x and p = String.length y in
  let rec aux i j =
    if j=p then true else
      if i=n then false else
        if x.[i]=y.[j] then aux (i+1) (j+1) else aux (i+1) j
  in aux 0 0  ;;
```

Dans le pire des cas, chacune de ces fonctions sera amenée à comparer tous les caractères de `x` à certains caractères de `y` : le nombre de comparaisons de caractères dans le pire des cas est donc pour chacune de ces trois fonctions égal à la longueur de la chaîne `x`

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

2. (a) On peut supposer que la longueur de Y est inférieure ou égale à celle de X. On construit alors une liste L constituée de toutes les sous-suites de Y classées par ordre décroissant de longueur (par exemple si Y est `"agi"` , L pourrait être `["agi";"gi";"ai";"ag";"i";"g";"a";""]`). On examine ensuite grâce à `est_sous_suite` si les différentes chaines figurant dans L sont également des sous-suites de X en s'arrêtant dès qu'on en trouve une qui est alors nécessairement une plus longue sous-suite commune à X et Y. L'algorithme s'arrête dans le pire des cas avec la sous-suite vide qui termine la liste L.

(b) Si n est la longueur de X et p celle de Y, L comporte 2^p éléments et chaque appel de l'instruction `est_sous_suite` utilise au pire n comparaisons donc au pire il y a $n.2^p$ comparaisons ce qui est bien sûr déraisonnable d'autant qu'on n'a pas tenu compte du coût pour fabriquer la liste L.

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) Si X est une chaîne de caractères non vide on convient de noter X_1 la sous-chaîne obtenue à partir de X en supprimant son premier caractère. Une première idée pour écrire `plssc` est, une fois traités les cas simples où l'une des chaînes est vide ainsi que le cas où les deux chaînes commencent par le même caractère, de prendre comme plus grande sous-chaîne commune à X et Y la plus longue des deux chaînes `plssc(` $X1, Y$ `)` et `plssc(` $X, Y1$ `)` ce qui donne la version suivante :

```
let plus_longue x y =
  if String.length x > String.length y then x else y;;
let rec plssc x y =
  if x = "" or y = "" then "" else
    let x1 = String.sub x 1 (String.length x - 1) and
        y1 = String.sub y 1 (String.length y - 1) in
      if x.[0] = y.[0]
      then
        (Char.escaped (x.[0]))^(plssc x1 y1)
      else
        plus_longue (plssc x1 y) (plssc x y1);;
```

   (b) Si n est la longueur de X et p celle de Y, si on note $C(n,p)$ le nombre de comparaisons de caractères effectuées par la fonction précédente dans le pire des cas, on a : $C(n,p) = C(n-1,p) + C(n,p-1)$ ce qui conduit par une récurrence facile sur $n+p$ à $C(n,p) = O(2^{n+p})$ ce qui est encore moins bien que pour l'algorithme naïf envisagé plus haut.

   (c) On notera à nouveau ici X_1 la sous-chaîne de X obtenue en lui retirant son premier caractère noté pour sa part x_1. Plaçons-nous à nouveau dans le cas où X et Y ne commencent pas par le même caractère et notons Z_1 une `plssc` de X_1 et Y_1. Il résulte alors facilement des propriétés données par l'énoncé qu'il existe une `plssc` de X et Y qui est soit de la forme $x_1 Z_1$ (c'est le cas si cette chaîne est une sous-suite de Y) soit de la forme $y_1 Z_1$ (si cette chaîne est une sous-suite de X) soit de la forme Z_1. On en déduit une nouvelle version de `plssc`

```
let rec plsc x y =
  if x = "" or y = "" then ""
  else
    let x1 = String.sub x 1 (String.length x - 1) and
        y1 = String.sub y 1 (String.length y - 1) in
    let z1 = plssc x1 y1 in
    if x.[0] = y.[0] then (Char.escaped x.[0])^z1
    else let x1z1 = (Char.escaped x.[0])^z1 in
      if est_sous_suite y x1z1 then x1z1
      else let y1z1 = (Char.escaped y.[0])^z1 in
        if est_sous_suite x y1z1 then y1z1 else z1;;
```

Si on note $C(n,p)$ le nombre de comparaisons de caractères requis dans le pire des cas pour cette fonction appliquée à des listes de longueurs n et p on a cette fois : $C(n,p) = C(n-1,p-1)+1+n+p$. Supposant $n \geqslant p$, on écrit cette relation pour les couples (n,p), $(n-1,p-1),\ldots,(n-p+1,1)$ et on remarque que $C(n-p,0) = 0$. En sommant ces relations, il vient : $C(n,p) = p+p.(n+p) - \sum_{k=1}^{p} 2k = p+p.(n+p) - p(p+1) = np$. Le nombre de comparaisons de caractères est cette fois en $O(np)$ ce qui est beaucoup plus satisfaisant.