NSI Terminale	TD 2 : Programmation Dynamique – L'alignement de séquences	Algorithme Séquence 3

On cherche dans ce TP à calculer la plus longue sous-séquence ($PLSS$) de deux chaînes de caractères. Par cela on entend : PLSS = la plus longue chaîne de caractères que l'on peut obtenir à partir des deux chaînes en supprimant certains de leurs caractères (pas obligatoirement les mêmes).

Notez bien que : l'on peut supprimer des caractères mais que **l'on ne change pas l'ordre des caractères restants.** La longueur de la $PLSS$ de deux chaînes est unique mais il peut exister plusieurs chaînes de cette longueur.

chaîne1	A	**B**	C	B	**D**	--	**A**	**B**	--
chaîne2	--	**B**	--	--	**D**	C	**A**	**B**	A

Par exemple : pour les chaînes ABCBDAB et BDCABA, la longueur maximale des $PLSS$ est 4 et l'une d'entre elles est BDAB. En effet :

~~A~~**B**~~C~~~~B~~**DAB** □ BDAB et **BD**~~C~~**AB**~~A~~ □ BDAB

Partie A : Approche « Force-Brute »

L'approche « Force Brute » consiste à énumérer toutes les sous-séquences de chacune des deux chaînes et à les comparer entre elles.

Sous séquence de longueur 0	Sous séquences de longueur 1	Sous séquences de longueur 2	Sous séquences de longueur 3	Sous séquences de longueur 4
∅	B A C B	BA BC BB AC AB CB	BAC BAB BCB ACB	BACB

Considérons les chaînes BACB et BCB. Les sous séquences de la première chaîne sont :

Il y en a donc 16 différentes (y compris la vide).

1) Lister toutes les sous-séquences de la seconde chaîne BCB.

Sous séquence de longueur 0	Sous séquences de longueur 1	Sous séquences de longueur 2	Sous séquences de longueur 3
∅	C B B	BC BB CB	BCB

2) Quelle est la PLSS entre ces 2 chaînes ?

3) Combien de sous-séquences compte une chaîne de n caractères tous différents ? On ne demande pas de justification exacte mais l'on pourra s'appuyer sur les tableaux.

4) Pourquoi une telle méthode est-elle infaisable en pratique lorsque les chaînes de caractères sont de grande taille ?

Partie B : Programmation dynamique

Considérons deux chaînes X et Y de longueurs respectives m et n.

Soit *PLSS* la fonction qui prend en argument deux chaînes X, Y et qui renvoie une $PLSS$ de X et Y.
Par exemple PLSS("ABCBDAB", "BDAB") renvoie "BDAB" (voir l'exemple du début de l'exercice).

La résolution du problème à l'aide de la programmation dynamique envisage deux cas de figures :

⇨ Le dernier caractère des deux chaînes est le même (X[-1] == Y[-1]) :

Dans ce cas, comme les derniers caractères coïncident, on peut étudier les sous chaînes sans ce dernier caractère et ajouter le dernier caractère au résultat.

La $PLSS$ est égale à : PLSS(X[:-1], Y[:-1]) + PLSS[-1]

Remarque : on rappelle qu'en python, X[-1] est le dernier caractère d'une chaîne et l'appel X[:-1] renvoie la chaîne X privée de son dernier caractère

On a donc par exemple : PLSS("ABCD","CFD") = PLSS("ABC","CF") + X[-1]

⇨ Les deux derniers caractères ne correspondent pas (X[-1] != Y[-1]) :

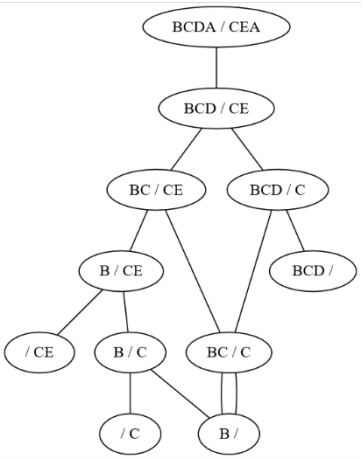
Dans ce cas il faut étudier deux sous-problèmes :

- celui dans lequel on a ôté le dernier caractère de X sans changer Y
- celui dans lequel on a ôté le dernier caractère de Y sans changer X

On retient le meilleur des deux cas en comparant les longueurs des résultats.

On a donc par exemple pour PLSS("ABC","CF") :

```
sol_x = PLSS("AB","CF")  #Si on enlève le C de ABC :
        sol_y = PLSS("ABC", "C") # Si on enlève le F de CF
if len(sol_x) >= len(sol_y) :
        PLSS("ABC","CF") = sol_x
        else :
                PLSS("ABC","CF") = sol_y
```



Les appels récursifs ont un cas de base qui est atteint lorsque l'une des deux chaînes est de longueur 0. Dans ce cas la longueur de la $PLSS$ vaut 0 et la fonction renvoie une chaîne vide " ".

Cette approche a toutefois un inconvénient : certains des cas de figure sont étudiés plusieurs fois…

Afin de pallier ce soucis on crée un dictionnaire M qui à chaque couple (chaîne1, chaîne2) associe la $PLSS$. Ce dictionnaire sera une variable « globale » qui pourra donc être lue et modifiée par chacun des appels récursifs. On utilise alors la programmation dynamique.

Remarque : on pourrait aussi comparer les premiers caractères de la chaîne au lieu des derniers.

1) On fournit le pseudo code incomplet suivant de la fonction PLSS. Compléter cet algorithme.

```
mémoire est un dictionnaire vide
Fonction PLSS(X : chaîne, Y : chaîne) -> chaîne:
        Si longueur(……) == 0 OU …… :
                Renvoyer ……
        Sinon :
                Si (X,Y) ne fait pas partie des clés de …… :
                        Si ……[-1] == …… :
                                mémoire[(X,Y)] = ……
                        Sinon :
                                sol_x = ……
                                ……
                                Si…… :
                                        mémoire[……] = ……
                                Sinon :
                                        ……
        Renvoyer mémoire[(X,Y)]
```

2) Coder cet algorithme en python. On pourra effectuer les tests suivants :

X = …	Y = …	PLSS(X, Y) = …
ABCBDAB	BDCABA	BDAB
∅	BDCABA	" "
ABCBDABCBDAB	BDCABA	BDCBA
ABCBDABCBDAB	ABCBDABCBDAB	ABCBDABCBDAB

3) Coder la fonction *longueur_PLSS(X, Y)* qui affiche la longueur des *plss* entre X et Y. Pour cela vous implémenterez un dictionnaire vide nommé *memoire_longueur* qui sera ensuite modifié à chaque appel récursif. On pourra se servir du code de la fonction PLSS et le modifier pour qu'il affiche non plus la *plss* mais sa longueur.

Source : Sur une idée d'un TP de Nicolas REVERET professeur de NSI