TP N°3 : Plus longue sous-suite commune

Pour l'exercice suivant, nous allons utiliser le type dictionnaire de python, la page suivante vous donne tout ce qui est utile pour l'exercice suivant :
https ://courspython.com/dictionnaire.html
(taper : "cours python dictionnaire" dans un moteur de recherche)

De plus, la méthode `get()` permet de renvoyer la valeur d'une clé donnée si elle est présente dans le dictionnaire, sinon elle retourne une valeur donnée (None par défaut si on l'utilise avec un argument). Par exemple, si `dico={A:2}`, alors dico.get("A",0) renvoie 2, alors que dico.get("C",0) renvoie 0.

Exercice 1. *Créer une fonction `ocu(L)` qui prend en argument une liste `L` et qui renvoie un dictionnaire contenant pour clés les éléments trouvées dans la liste L et pour valeurs le nombre d'occurrence de ces éléments.*

Correction :

```
def ocu(L):
    dico={}
    for k in L:
        dico[str(k)]=dico.get(str(k),0)+1
    return dico
```

Exercice 2 : Recherche d'une plus grande sous-suite commune

Définition 1.1: Sous-suite d'une suite finie

Soit $S = (s_0, s_1, \ldots, s_{n-1})$ une suite (de caractères, de mots etc...). Une sous-suite de S (ou une suite extraite selon une terminologie fréquente dans les cours de maths) est une suite $(z_0, z_1, \ldots, z_{p-1})$ (éventuellement vide) de termes de S construite de la façon suivante :
- on se donne une suite strictement croissante d'indices $0 \leq i_0 < i_1 < \ldots < i_{p-1} \leq n-1$
- on pose $z_k = s_{i_k}$ pour $0 \leq k \leq p-1$. En d'autres termes, $Z = (z_k)_k$ est obtenue en prélevant dans l'ordre certains éléments de S.

Problème :

Soient $S = (s_i)_{0 \leq i < m}$ et $T = (t_j)_{0 \leq j < n}$ deux suites. On appelle sous-suite commune à S et T toute suite $Z = (z0, \ldots, z_{p-1})$ pour laquelle il existe un couple $(I, J) \subset [\![0, m-1]\!] \times [\![0, n-1]\!]$ tel que :
-$|I| = |J| = p$
- en numérotant dans l'ordre croissant les éléments de I et de J : $z_k = s_{i_k} = t_{j_k}$, pour $0 \leq k < p$
Parmi ces sous-suites, une plus grande sous-suite est un couple (I, J) pour lequel p est maximal, pour simplifier, on notera PLSC au lieu de plus grand sous-suite commune.

Ce problème des applications importantes dans la comparaison de textes, la recherche de plagiats, la comparaison de génomes.

1 Si on a que `S="numérisation"` et `T="mémoïsation"` que vaut p, I, et J ?

> **Correction :** $p = 8$, $I = (2,3,6,7,8,9,10,11)$ et $J = (0,1,5,6,7,8,9,10)$, la sous-suite donne le mot `"mésation"`.

2 Combien existe-t-il de sous-suites à une suite finie de p éléments ? Est-il possible, selon vous, d'effectuer une recherche par force brute (en comparant toutes les sous-suites de chaque suite) ?

> **Correction :** Il existe 2^p sous-suites à une suite à p éléments (autant que de parties à p éléments). Un algorithme de recherche par force brute impliquerait de comparer les 2^m sous-suites de S aux 2^n sous-suites de T ce qui donne 2^{n+m} comparaisons à faire : c'est d'une complexité exponentielle.

Nous allons voir qu'une propriété permet de réduire efficacement la complexité du problème. En effet, si on note maintenant Z une plus longue suite commune, on a la propriété suivante :

- Soit $s_{m-1} = t_{n-1}$ et dans ce cas : $z_p = s_{m-1} = t_{n-1}$ et $(z_0, \dots z_{p-2})$ est une plus longue suite commune à $S' = (s_i)_{0 \leq i < m-1}$ et $T' = (t_j)_{0 \leq j < n-1}$
- Soit $s_{m-1} \neq t_{n-1}$ et dans ce cas : Z est une plus longue suite commune à $S' = (s_i)_{0 \leq i < m-1}$ et T ou à S et $T' = (t_j)_{0 \leq j < n-1}$

Nous remarquons donc qu'ici la construction d'une structure optimale au problème se fait en construisant une sous-structure optimale : une solution optimale se construit à l'aide d'une solution optimale pour des suites de plus petite taille, c'est le principe de la programmation dynamique.
Le terme programmation dynamique désigne une façon de traiter des problèmes d'optimisation combinatoire qui peuvent se ramener à la résolution de sous-problèmes (et en programmant de façon tantôt itérative et ascendante, tantôt récursive). Nous revisitons ainsi une problématique déjà abordée avec les algorithmes gloutons. Ce sera également l'occasion de revenir sur certaines notions abordées avec l'étude de la récursivité..

Nous allons voir deux manières de résoudre le problème, l'une itérative, l'autre récursive.

Méthode récursive

On considère deux chaînes S et T de longueurs m et n, on note $c(m,n)$ la longueur d'une PLSC de S et de T, et d'une façon générale, lorsque $0 \leq p < m$ et $0 \leq q < n$, $c(p,q)$ désignera la longueur d'une PLSC de $S[0:p]$ et $T[0:q]$ (notation python).

3 On suppose $p = 0$ ou $q = 0$, que vaut $c(p,q)$?

> **Correction :** L'une des suites est vide, la seule sous-suite commune est la suite vide, d'où $c(p,q) = 0$.

4 Le théorème précédent permet d'exprimer $c(p,q)$ en fonction de $c(p',q')$ avec $p'+q' < p+q$. Compléter ce qui suit en fonction de l'énoncé de ce théorème :

$$\begin{cases} \text{Si } s_{p-1} = t_{p-1}, \text{ alors, } c(p,q) = ??? \\ \text{Si } s_{p-1} \neq t_{p-1}, \text{ alors, } c(p,q) = ??? \end{cases}$$

Correction :

$$\begin{cases} \text{Si } s_{p-1} = t_{p-1}, \text{ alors, } c(p,q) = c(p-1,q-1)+1 \\ \text{Si } s_{p-1} \neq t_{p-1}, \text{ alors, } c(p,q) = max(c(p-1,q), c(p,q-1)) \end{cases}$$

5 Écrire une fonction récursive PLSC1 (S, T, D) dont l'appel principal se fera avec D = {} (dictionnaire vide) et qui construit et renvoie le dictionnaire (de dictionnaires) avec $D[p][q] = c(p,q)$ pour $0 \leq p < m, 0 \leq q < n$.

Correction :

```
def PLSC1(S,T,D):
    m, n = len(S), len(T)
    if m not in D :
        D[m]={}
    if m==0 or n==0:
        D[m][n]=0
        return D[m][n]

    elif S[-1] == T[-1] :
        if m-1 not in D or n-1 not in D[m-1]:
            r=PLSC1(S[0:m-1],T[0:n-1],D)
        else :
            r=D[m-1][n-1][0]
        D[m][n]= r+1
        return D[m][n]

    else:
        if m-1 not in D or n not in D[m-1]:
            r1=PLSC1(S[0:m-1], T, D)
        else :
            r1=D[m-1][n]

        if n-1 not in D[m]:
            r2=PLSC1(S,T[0:n-1],D)
        else:
            r2=D[m][n-1]

        if r1>r2:
            D[m][n]= r1
        else:
            D[m][n] = r2

        return D[m][n][0]
```

6 Donner une majoration de la complexité en mémoire et en temps de cette fonction.

Correction : Le dictionnaire de dictionnaires occupera au plus $(m+1) \times (n+1)$ entrées et chaque valeur prend la place mémoire d'un entier long et de deux entiers dans $\{-1, 0, 1\}$. Le nombre d'appels récursifs avec mémoïsation est, lui aussi, majoré par $(m+1) \times (n+1)$

7 Modifier PLSC1(S, T, D) pour obtenir PLSC2(S, T, D) qui remplit le dictionnaire avec : $D[p][q] = c(p,q), (-1,-1)$, dans le premier cas de figure du théorème du théorème, $D[p][q] = c(p,q), (-1,0)$ ou $D[p][q] = c(p,q), (0,-1)$ dans le second. Ces informations supplémentaires permettront de reconstituer une plus longue liste commune.

Correction :

```
def PLSC2(S,T,D):
    m, n = len(S), len(T)
    if m not in D :
        D[m]={}
    if m==0 or n==0:
        D[m][n]=0, (0,0)
        return D[m][n][0]

    elif S[-1] == T[-1] :
        if m-1 not in D or n-1 not in D[m-1]:
            r=PLSC1(S[0:m-1],T[0:n-1],D)
        else :
            r=D[m-1][n-1][0]
        D[m][n]= r+1, (-1,-1)
        return D[m][n][0]

    else:
        if m-1 not in D or n not in D[m-1]:
            r1=PLSC1(S[0:m-1], T, D)
        else :
            r1=D[m-1][n][0]

        if n-1 not in D[m]:
            r2=PLSC1(S,T[0:n-1],D)
        else:
            r2=D[m][n-1][0]

        if r1>r2:
            D[m][n]= r1, (-1,0)
        else:
            D[m][n] = r2, (0,-1)

        return D[m][n][0]
```

Version itérative

8 Écrire une fonction itérative PLSC 3(S, T) qui renvoie un dictionnaire D tel que : $D[p][q] = c(p,q),(-1,-1)$, dans le premier cas de figure du théorème, $D[p][q] = c(p,q),(-1,0)$ ou $D[p][q] = c(p,q),(0,-1)$ dans le second. Cette nouvelle version doit être itérative, en voici un début de script :

```
def PLC3(S, T) :
        m, n = len(S), len(T)
        D={p: {} for p in range(m+1) }
        for q in range(n+1):
                D[0][q] = 0, (0,0)
        for p in range(n+1):
                D[p][0] = 0, (0,0)

        for p in range(1,m+1):
                for q in range(1,n+1):
                        ???????
        return D
```

Correction :

```
def PLSC3(S, T) :
        '''
        S,T : str;
        renvoie un dictionnaire
        '''
        m, n = len(S), len(T)
        D={p: {} for p in range(m+1) }
        for q in range(n+1):
                D[0][q] = 0, (0,0)
        for p in range(m+1):
                D[p][0] = 0, (0,0)

        for p in range(1,m+1):
                for q in range(1,n+1):
                        if S[p-1] == T[q-1]:
                                D[p][q] = D[p-1][q-1][0]+1 , (-1,-1)
                        else :
                                r1 = D[p-1][q][0]
                                r2 = D[p][q-1][0]
                                if r1 > r2 :
                                        D[p][q] = r1, (-1,0)
                                else :
                                        D[p][q] = r2, (0,-1)
        return D
```

9 Donner la complexité en mémoire et en temps de cet algorithme.

Correction : Le dictionnaire contient au plus $(m+1)\times(n+1)$ paires et le nombre d'itérations est $(m+1)\times(n+1)+m+n+2 = O(m\times n)$.

Construire une plus longue sous-suite commune

On se propose de déterminer explicitement une PLSC à S et T à partir des informations contenues dans les dictionnaires construits par dans la procédure PLSC2(S,T, D) ou par PLSC3(S,T). On rappelle que, dans les deux cas, D est un dictionnaire tel que $D[p][q] = c(p,q), (\alpha,\beta)$ où :
$c(p,q)$ est la longueur d'une PLSC à $S[0:p], T[0:q]$;
$(\alpha,\beta) = (-1,-1)$ si $S[p-1] = T[q-1]$
$(\alpha,\beta) = (-1,0)$ si $S[p-1] \neq T[q-1]$ et $c(p-1,q) > c(p,q-1)$;
$(\alpha,\beta) = (0,-1)$ si $S[p-1] \neq T[q-1]$ et $c(p-1,q) \leq c(p,q-1)$;

10 Écrire une fonction PLSC(S,T) qui prend deux listes en arguments et en renvoie une PLSC. Cette fonction appelle PLSC2 ou PLSC3 qui construit ou retourne D, elle dispose d'une sous-procédure récursive reconstruire PLSC (p, q) qui renvoie une PLSC à $S[0:p]$ et $T[0,q]$:

```
def PLSC(S,T):
    #sous-fonction
    def reconstruirePLSC(p,q):
        if p==0 or q==0 :
            return ???
        else :
            ????
    #fin de la sous-fonction
    m,n=len(S), len(T)
    D=PLSC3(S,T)
    return reconstruirePLSC(m,n)
```

Correction :

```
def PLSC(S,T):
    #sous-fonction
    def reconstruirePLSC(p,q):
        if p==0 or q==0 :
            return ''
        else :
            R=D[p][q][1]
            if R== (-1,-1):
                return reconstruirePLSC(p-1,q-1) + T[q-1]
            elif R==(-1,0):
                return reconstruirePLSC(p-1,q)
            elif R==(0,-1):
                return reconstruirePLSC(p,q-1)
    #fin de la sous-fonction
    m,n=len(S), len(T)
    D=PLSC3(S,T)
    return reconstruirePLSC(m,n)
```