correction du TD5

1. Construire une sous-séquence revient à choisir un sous-ensemble parmi les indices $[\![0, n-1]\!]$ donc il y a au total $\boxed{2^n \text{ sous-séquences de } \texttt{ch1}}$

Une fois déterminées les 2^n sous-séquences de `ch1` (coût $O(2^n)$), il faut vérifier, pour chacune si elle est une sous-séquence de `ch2` (et mémoriser la sous-séquence de longueur maximale parmi celle-ci) donc 2^n recherches de coût $O(m)$. Le coût total est $\boxed{O(m2^n)}$.

On peut minimiser ce coût en testant les sous-séquences de `ch1` en commençant par les plus longues (pour éviter la recherche du maximum après) mais cela ne modifie pas le coût dans le pire des cas (aucune sous-séquence commune) qui nécessite de faire la recherche en intégralité.

2. On trouve
$$\begin{array}{|ccccc|}\hline 0&0&0&0&0\\ 0&0&0&0&0\\ 0&1&1&1&1\\ 0&1&1&2&2\\ 0&1&1&2&3\\ \hline\end{array}$$

La longueur d'une PLSSC est donc $\boxed{\ell_{4,4}=3}$ qui correspond à la sous-séquence `'psi'`.

3. a)
- Si $i=0$ ou $j=0$, une des chaine `ch1[:i]` ou `ch2[:j]` est vide donc $\ell_{i,j}=0$
- sinon, si `ch1[i-1]=ch2[j-1]`, les deux chaînes `ch1[:i]` et `ch2[:j]` terminent par le même caractère ; une PLSSC de `ch1[:i]` et `ch2[:j]` est donc constituée d'une PLSSC de `ch1[:i-1]` et `ch2[:j-1]` à laquelle on rajoute le caractère `ch1[i-1]=ch2[j-1]`
- sinon, une PLSSC `ch` de `ch1[:i]` et `ch2[:j]` est soit une PLSSC de `ch1[:i-1]` et `ch2[:j]` (si `ch1[i-1]` n'est pas le dernier caractère de `ch`), soit une PLSSC de `ch1[:i]` et `ch2[:j-1]` (si `ch2[i-1]` n'est pas le dernier caractère de `ch`).

b) La recherche de la solution optimale du problème $\ell_{n,m}$ passe par la recherche des solutions optimales des sous-problèmes $\ell_{n-1,m-1}$, $\ell_{n-1,m}$ et $\ell_{n,m-1}$ selon les cas.

c) On crée une liste de liste de bonne taille que l'on remplit ligne par ligne selon les formules précédentes :

```
def tableauPLSSC(ch1,ch2) :
    n = len(ch1)
    m = len(ch2)
    T = [[0 for j in range(m+1)] for i in range(n+1)]
    for i in range(1,n+1) :
        for j in range(1,m+1) :
            if ch1[i-1] == ch2[j-1] :
                T[i][j] = 1+T[i-1][j-1]
            else :
                T[i][j] = max([T[i-1][j],T[i][j-1]])
    return T
```

d) Il suffit de renvoyer la valeur de $\ell_{n,m}$ donc le dernier élément de la dernière ligne du tableau précédent :

```
def longPLSSC(ch1,ch2) :
    return tableauPLSSC(ch1,ch2)[-1][-1]
```

4. a) La longueur d'une PLSSC est $\ell_{4,4}=3$; on cherche ensuite à remonter dans ce tableau pour voir comment cette valeur a été obtenue :

- $\ell_{4,4}=1+\ell_{3,3}$ donc une PLSSC est obtenue en rajoutant la dernière lettre de `'mpsi'` à une PLSSC de `'mps'` et `'pcs'`.
- de même $\ell_{3,3}=1+\ell_{2,2}$ donc on ajoutera `'s'` à une PLSSC de `'mp'` et `'pc'`.
- On a enduite une ambiguïté car $\ell_{2,2}=1+\ell_{1,1}=\ell_{2,1}$: comme les dernières lettres de `'mp'` et `'pc'` ne sont pas identiques, la valeur de $\ell_{2,2}$ provient de $\ell_{2,2}=\ell_{2,1}$; une PLSSC de `'mp'` et `'pc'` est donc une PLSSC de `'mp'` et `'p'`.

  C'est en fait un problème selon l'ordre dans lequel on fait la comparaison : si on privilégie en premier la vérification de $\ell_{i,j}=\max\{\ell_{i,j-1},\ell_{i-1,j}\}$, il n'y a plus de problème. Si par exemple $\ell_{i,j}=\ell_{i-1,j}$, cela signifie qu'il existe une PLSSC de `ch1[:i]` et `ch2[:j]` de même longueur qu'une PLSSC de `ch1[:i-1]` et `ch2[:j]`.
- $\ell_{2,1}=1+\ell_{1,0}=1$ donc une PLSSC de `'mp'` et `'p'` est `'p'`

Au final une PLSSC et `'p'+'s'+'i'`

On peut également se contenter de raisonner avec la dernière pigne (ou colonne) du tableau : si $\ell_{n,j}=1+\ell_{n,j-1}$, la lettre `ch2[j-1]` fait partie d'une PLSSC alors que si $\ell_{n,j}=\ell_{n,j-1}$, la lettre `ch2[j-1]` n'est pas nécessaire à la reconstruction d'une PLSSC

b) On suit le raisonnement précédent (jusqu'à rejoindre « un bord du tableau ») :

```
def PLSSC1(ch1,ch2) :
    T = tableauPLSSC(ch1,ch2)
    ch = ''
    n,m = len(ch1),len(ch2)
    while n>0 and m>0 :
        if T[n][m] == T[n-1][m] :
            n -= 1
        elif T[n][m] == T[n][m-1] :
            m -= 1
        else :
            n -= 1
            m -= 1
            ch = ch1[n]+ch  # concaténation à gauche
    return ch
```

Ou la version à partir de la dernière ligne :

```
def PLSSC2(ch1,ch2) :
    T = tableauPLSSC(ch1,ch2)
    ch = ''
    n,m = len(ch1),len(ch2)
    while m>0 :
        if T[n][m] == 1+T[n][m-1] :
            ch = ch2[m-1]+ch  # concaténation à gauche
        m -= 1
    return ch
```

c) La construction du tableau à un coût $O(n \times m)$, la recherche de la PLSSC a ensuite un coût $O(n+m)$ donc la complexité de `PLSSC` est $\boxed{O(n \times m)}$

5. Une PLSSC de `ch1` et `ch2` se détermine, selon leur dernier caractère à partir d'une PLSSC de `ch1[:n-1]` et `ch2` ou de `ch1` et `ch2[:m-1]` ou de `ch1[:n-1]` et `ch2[:m-1]`. On mémorise les résultats des sous-problèmes dans un dictionnaire dont les clés sont les couples `(ch1,ch2)` :

```
def PLSSC1(ch1,ch2) :
    dico = {}
    def f(ch1,ch2) :
        if (ch1,ch2) not in dico :
            if len(ch1)*len(ch2) == 0 :
                r = ''
            elif ch1[-1] == ch2[-1] :  # même dernier caractère donc on le
                garde ; il est à la fin d'une PLSSC
                r = f(ch1[:-1],ch2[:-1]) + ch1[-1]
            else :
                r1 = f(ch1,ch2[:-1])    # les deux sous-problèmes à
                    envisager
                r2 = f(ch1[:-1],ch2)
                if len(r1)>len(r2) :    # on garde le meilleur
                    r = r1
                else :
                    r = r2
            dico[(ch1,ch2)] = r
        return dico[(ch1,ch2)]
    return f(ch1,ch2)
```