

Devoir surveillé n°1

Durée : 45 minutes

L'utilisation d'une calculatrice ou de tout document est interdit.

Toute sortie anticipée est interdite.

Le sujet d'intéresse au problème de détermination de la longueur de la plus longue sous-séquence commune à deux séquences d'ADN. Cette longueur est un indicateur de proximité d'espèces permettant de les comparer lors d'une étude phylogénétique.

Les parties 1, 2 et 3 sont indépendantes. On pourra utiliser les fonctions de la partie 1 dans la partie 3.

1. Questions préliminaires

- Écrire une fonction `maximum` qui renvoie plus grand des deux nombres passés en argument.
Par exemple, `maximum(2, 4)` devra renvoyer 4.
- Écrire une fonction `zeros` qui prend en argument deux entiers naturels n et p (supposés non nuls) et qui renvoie une liste de n listes contenant chacune p zéros, représentant ainsi la matrice nulle à n lignes et p colonnes.
Par exemple, `zeros(2, 3)` devra renvoyer `[[0,0,0], [0,0,0]]`.

2. Plus longue sous-chaîne commune

On considère $A = a_1 \dots a_n$ et $B = b_1 \dots b_p$ deux chaînes de caractères non vides.

On appelle **sous-chaîne** de A toute chaîne de caractères $a_{i_1} \dots a_{i_k}$ où $1 \leq i_1 < \dots < i_k \leq n$ (**ces caractères ne sont pas nécessairement consécutifs** dans A).

On appelle **plus longue sous-chaîne commune** à A et B toute sous-chaîne commune à A et B de longueur maximale. Si l'une des chaînes A ou B est vide, ou si A et B n'ont aucune sous-chaîne commune, on convient que la chaîne vide est l'unique plus longue sous-chaîne commune à A et B .

On s'intéresse alors au problème ci-dessous.

Étant donné deux chaînes de caractères A et B de longueurs respectives n et p , quelle est la longueur d'une plus longue sous-chaîne commune à A et B ?

Par exemple, les chaînes de caractères "AAA" et "TAA" sont des plus longues sous-chaînes communes aux chaînes de caractères `chaine1 = "ATAGA"` et `chaine2 = "TAACA"`.

La chaîne de caractères `sschaine = "ATGC"` est une plus longue sous-chaîne commune aux chaînes de caractères `chaine1 = "AATGCG"` et `chaine2 = "TATTAGC"`.

- Quelle est la longueur d'une plus longue sous-chaîne commune aux chaînes "AATGCG" et "TATTAGC" ? Justifier.
- Déterminer une plus longue sous-chaîne commune aux chaînes "AATGCG" et "TATTAGC" autre que "ATGC".

- c. Parmi les chaînes de caractères ci-dessous, indiquez sur votre copie quelle est la seule plus longue sous-chaîne commune aux chaînes "TCGTA" et "CTG"? *On ne demande pas de justifier la réponse.*

"CTG" "TGCT" "CGT" "CG"

- d. Déterminer toutes les plus longues sous-chaînes communes aux chaînes "TCGTA" et "CTG". *On ne demande pas de justifier la réponse.*
- e. Parmi les fonctions ci-dessous, déterminer les deux seules permettant de déterminer si une chaîne de caractères `ssch` est une sous-chaîne d'une chaîne de caractères `ch`. *On ne demande pas de justifier la réponse.*

```
def estSousChaine1(ch, ssch):
    n = len(ch)
    p = len(ssch)
    i, j = 0, 0
    while i < n and j < p:
        if ch[i] == ssch[j]:
            j += 1
            i += 1
    return j == p
```

```
def estSousChaine2(ch, ssch):
    n = len(ch)
    p = len(ssch)
    i, j = 0, 0
    while i < n and j < p:
        if ch[i] == ssch[j]:
            j += 1
            i += 1
    return i == j
```

```
def estSousChaine3(ch, ssch):
    n, p = len(ch), len(ssch)
    for i in range(n):
        j = 0
        while j < p and ch[i+j] == ssch[j]:
            j += 1
        if j == p:
            return True
    return False
```

```
def estSousChaine4(ch, ssch):
    j = 0
    for i in range(len(ch)):
        if ch[i] == ssch[j]:
            j += 1
        if j == len(ssch):
            return True
    return False
```

- f. Écrire alors une fonction booléenne `sousChaineCommune` qui prend en argument trois chaînes de caractères `chaine1`, `chaine2` et `sschaine` qui renvoie `True` si `sschaine` est une sous-chaîne commune à `chaine1` et `chaine2`, et `False` dans le cas contraire.

3. Recherche d'une solution par programmation dynamique

Si $A = a_1 \dots a_n$ et $B = b_1 \dots b_p$ sont deux chaînes de caractères non vides, on note $\ell_{i,j}$ la longueur d'une plus longue sous-chaîne commune aux chaînes $a_1 a_2 \dots a_i$ et $b_1 b_2 \dots b_j$ (respectivement composées des i premiers et j premiers caractères de A et B).

On peut montrer que la suite double $(\ell_{i,j})_{(i,j) \in \llbracket 0, n \rrbracket \times \llbracket 0, p \rrbracket}$ vérifie l'initialisation et la relation de récurrence :

$$\forall (i, j) \in \llbracket 0, n \rrbracket \times \llbracket 0, p \rrbracket, \ell_{i,j} = \begin{cases} 0 & \text{si } i = 0 \text{ ou } j = 0 \\ 1 + \ell_{i-1, j-1} & \text{sinon si } a_i = b_j \\ \max(\ell_{i-1, j}, \ell_{i, j-1}) & \text{sinon si } a_i \neq b_j \end{cases}$$

On cherche donc à calculer la longueur d'une plus longue sous-chaîne commune à $A = a_1 \dots a_n$ et $B = b_1 \dots b_p$, c'est-à-dire le coefficient $\ell_{n,p}$.

Le principe est de calculer de proche en proche chaque coefficient $\ell_{i,j}$ (en respectant l'ordre défini par la relation de récurrence) en les mémorisant dans une matrice.

- a. Quelle est la taille (nombres de lignes et de colonnes) de la matrice où on mémorisera les coefficients $\ell_{i,j}$?
- b. Recopier et compléter le code de la fonction `lplsch` calculant la longueur d'une plus longue sous-chaine commune à deux chaînes de caractères `chaine1` et `chaine2` passées en arguments.

```
def lplsch(chaine1, chaine2):
    n, p = len(chaine1), len(chaine2)
    l = zeros(....., .....)
    for i in range(....., n+1):
        for j in range(1, .....):
            if chaine1[i-1] == chaine2[j-1]:
                l[i][j] = .....
            else:
                l[i][j] = maximum(....., .....)
    return l[.....][.....]
```

- c. Donner la matrice des coefficients ($\ell_{i,j}$) dans le cas où `A="CTG"` et `B="TCGT"`.
- d. Expliquer (sans l'implémenter) comment adapter l'algorithme donné par la fonction `lplsch` pour construire une plus longue sous-chaine commune.
- e. On sait qu'une chaîne de n caractères admet au plus 2^n sous-chaines distinctes (en comptant la chaîne vide). Expliquer pourquoi la méthode programmée à la question précédente est plus efficace qu'en comparant chaque sous-chaine de A avec chaque sous-chaine de B .