DS1 PCSI Python (durée = 1h15)

3 pts **Question de cours**

Ecrire une fonction d'en-tête `dicho(L,x)` prenant en arguments une liste triée L et un élément x, qui renvoie `True` si x est dans L et `False` sinon. La recherche doit se faire par dichotomie

Problème

Une séquence d'ADN est constituée de deux brins complémentaires, qui sont un long enchaînement de nucléotides de quatre types différents désignés par les lettres `A` (Adénine), `T` (Thymine), `C` (Cytosine) et `G` (Guanine).
Pour simplifier le sujet, on suppose qu'une séquence d'ADN est une liste non vide dont les éléments sont des caractères de l'alphabet `{A,C,G,T}` (on s'intéresse donc seulement à l'un des deux brins).

Partie I

1 pt 1) On considère la séquence d'ADN :

```
seq=['A','T','C','G','T','A','C','G']
```

Que renvoient respectivement les commandes `seq[4]` ? `seq[-1]` ? `seq[2:6]` ?

2 pts 2) Écrire une fonction `generation(n)` qui renvoie une séquence d'ADN aléatoire de longueur n, construite en ajoutant successivement à une liste initialement vide, des nucléotides tirés au hasard dans `{A,C,G,T}`.

On suppose que la fonction `randint` du module `random` a déjà été importée. La commande `r=randint(1,4)` tire un entier aléatoire r entre 1 et 4 ; on associera respectivement les valeurs 1, 2, 3 et 4 aux nucléotides `A`, `C`, `G` et `T`.

3) La fonction ci-dessous prend en argument une séquence d'ADN.

```
def pourcentage(seq):
    a,b,c,d=0,0,0,0
    n=len(seq)
    i=0
    while i<n:
        if seq[i]=='A':
            a=a+1
        elif seq[i]=='C':
            b=b+1
        elif seq[i]=='G':
            c=c+1
        else:
            d=d+1
        i=i+1
    return [.........,.........,.........,.........]
```

1 pt a) Compléter la fonction pour qu'elle renvoie la liste des pourcentages de `A`, `C`, `G` et `T` dans la séquence d'ADN.

0,5 pt b) Calculer, en fonction de n, le nombre total d'additions effectuées lors de l'exécution de la boucle `while` (en comptant toutes les itérations).

Partie II

Soit S une séquence d'ADN de longueur n.
On appelle *motif* de S toute liste M pour laquelle il existe deux entiers i et j avec $0 \leq i < j \leq n$ tels que $M = S[i : j]$ (c'est-à-dire les éléments de S d'indices i, $i+1$, ..., $j-1$).
On dit alors que M *apparaît* dans S à l'indice i.

Exemples :
Prenons `S=['A','C','T','G','G','T','C','A','C','T']`.
`M=['T','G','G']` est un motif de S (il apparaît à l'indice 2).
En revanche, `M=['T','A','G']` n'en est pas un.
`M=['A','T']` n'est pas non plus un motif de S : les lettres `'A'` puis `'T'` apparaissent bien dans cet ordre dans S, mais jamais côte à côte.

II.1 Algorithme naïf :

La fonction `recherche` ci-dessous prend en arguments deux séquences d'ADN : le motif M (de longueur m) et la séquence S (de longueur n).
On teste successivement les positions possibles de M dans S ; pour chacune, on regarde si M apparaît à cet indice.

```
def recherche(M,S):
    m=len(M)
    n=len(S)
    for i in range(......):
        c,j=0,0
        while j<m:
            if S[i+j]==M[j]:
                c=......
                j=j+1
            else:
                j=m
        if c==m:
            return ......
    return ......
```

2 pts 4)a) Compléter la fonction `recherche` afin qu'elle renvoie le plus petit indice où M apparaît dans S si M est un motif de S, et -1 sinon.

Exemple :
Si `S=['A','C','T','G','G','T','C']` et `M=['T','G','G']`, la fonction doit renvoyer 2.

1 pt b) Que se passerait-il si on supprimait le bloc `else` ?

1 pt 5) On admet que, dans le pire des cas, le nombre d'opérations élémentaires effectuées par cette fonction est de l'ordre de $3mn$, lorsque $m \ll n$.
Sachant qu'une séquence d'ADN humain est composée de 3 milliards de caractères, combien de secondes, dans le pire des cas, met un ordinateur réalisant 10^{10} opérations élémentaires par seconde, pour chercher un motif de 50 caractères dans cette séquence d'ADN ?

II.2 Bordures

Soit S une séquence d'ADN de longueur n.
Un *préfixe* de S est un motif de la forme $S[0:j]$ avec $0 < j < n$ (c'est-à-dire, un début de S, mais différent de S).
Par exemple, `['C','A']` et `['C']` sont des préfixes de `['C','A','T']`, mais pas `['A']` (ce n'est pas un début), ni `['C','A','T']` car il est égal à S.

Un *suffixe* de S est un motif de la forme $S[i:n]$ avec $0 < i < n$ (c'est-à-dire une fin de S, mais différente de S).
Par exemple, `['A','T']` et `['T']` sont des suffixes de `['C','A','T']`, mais pas `['C']` (ce n'est pas une fin), ni `['C','A','T']`.

Lorsqu'un motif de S est à la fois un préfixe et un suffixe de S, on dit que c'est une *bordure* de S (une séquence S peut posséder plusieurs bordures, une bordure ou aucune).

1 pt 6) Donner tous les préfixes et les suffixes de `['A','C','G','A']`.

1,5 pt 7) Déterminer les bordures respectives de `['A','C','G','T','A','C']` et `['A','C','A','C','A']`.

On considère par la suite la fonction `bordures` qui prend comme argument une séquence S de longueur n.

```
def bordures(S):
    F=[0]
    i=1
    j=0
    while i<len(S):
        if S[j]==S[i]:
            F.append(j+1)
            i=i+1
            j=j+1
        else:
            if j>0:
                j=F[j-1]
            else:
                F.append(0)
                i=i+1
    return F
```

On admet que, pour tout indice k de la liste `F` renvoyée par `bordures(S)`, `F[k]` est la longueur de la plus longue bordure de `S[0:k+1]` (avec `F[k]=0` si ce motif n'a pas de bordure).

2 pts 8) On appelle la fonction avec `S=['A','C','A','A','C','A']`.
Compléter le tableau en précisant le contenu des variables `i`, `j` et `F` à la fin de chacune des six itérations de la boucle `while`.

boucle	`i`	`j`	`F`
$\times\times\times$	1	0	[0]
1			
2			
3			
4			
5			
6			

1 pt 9) Avec la valeur de `F` obtenue à la question précédente, vérifier que `F[5]` est bien la longueur de la plus longue bordure de S, puis donner cette bordure.

2 pts 10) Montrer que la boucle `while` de la fonction `bordures` se termine.
Indication : on pourra étudier l'évolution de la quantité $2i - j$ au cours des itérations.

1 pt 11) Justifier que la liste `F` renvoyée a la même longueur que S.