Devoir surveillé n°2

Durée : 2H. Calculatrices interdites. **Les candidats sont invités à porter une attention particulière à la rédaction et la propreté : les copies illisibles ou mal présentées seront pénalisées.**

Le sujet comporte deux exercices, ainsi qu'une annexe comportant des rappels de syntaxe python sur les listes, chaînes et dictionnaires.

Chaque fonction écrite devra être commentée, soit dans le code, soit par des phrases d'explication.

Vous pouvez utiliser toute fonction décrite dans l'énoncé pour les question suivantes.

Exercice 1 (Un brin d'ADN)

On s'intéresse dans cette exercice à l'analyse algorithmique d'un brin d'ADN. Un tel brin est une suite de nucléotides. Il existe 4 types de nucléotides que l'on représente usuellement par 4 lettres «A», «C», «G», «T».

Informatiquement, un brin d'ADN sera donc une chaîne de caractères de longueur arbitraire et composée uniquement de ces 4 lettres (en majuscule).

Partie I : Comparaison de chaînes

Dans un brin d'ADN, les nucléotides sont regroupés par triplets (appelés codons) qui codent chacun une information génétique (un codon correspond à un acide aminé bien spécifique).

Ainsi, informatiquement, un codon est une chaîne de caractère de longueur 3 composée de caractères parmi A, C, G, T.

1. Séparation des codons.
   (a) On dispose d'une chaîne de caractères `s`. Écrire un test (`if`) permettant de savoir si la longueur de `s` est un multiple de 3.
   (b) Écrire une fonction `separe(brin)` qui prend comme argument une chaîne de caractères `brin` (encodant un brin d'ADN) et retourne une liste de chaînes de caractères (chacune de longueur 3) qui est la liste des codons contenus dans `brin`. On suppose que la longueur de `brin` est un multiple de 3 (ce qui ne sera pas vérifié dans le code).
   Par exemple `separe("ACGTCATTC")` retourne la liste `["ACG", "TCA", "TTC"]`
2. Analyse des codons.
   (a) En théorie, combien de codons différents existe-t-il ?
   (b) On souhaite, pour un brin d'ADN donné, connaître sa composition générale, c'est à dire connaître le nombre d'occurrence de chaque codon dans ce brin.
   Nous allons calculer un dictionnaire contenant ces informations. Les clés seront les codons (chaîne de longueur 3) et les valeurs le nombre d'occurrence de ce codon dans le brin.
   Écrire une fonction `compte_codons(L)` qui prend comme argument une liste de codons (telle que celles retournées par la fonction `separe`) et retourne le dictionnaire des occurrences correspondant.
   (c) Écrire une fonction `codon_majoritaire(d)` qui prend comme argument un dictionnaire d'occurrences et retourne le codon qui apparaît le plus souvent dans le brin analysé. En cas d'égalité, on retournera l'un des codons majoritaire.
3. Comparaison de deux brins d'ADN.
   Une première approche consiste à comparer les nombres d'occurrences de chaque codon dans les deux brins pour estimer leur similarité.
   (a) On dispose de `brin1` et `brin2` , des chaînes de caractères de même longueur qui est un multiple de 3. Donner les commandes python permettant d'obtenir les dictionnaires d'occurrences pour chacun de ces brins.
   (b) En notant n la longueur commune des deux brins, donner le nombre d'opérations nécessaires au calcul des deux dictionnaires précédents. On considère comme opération : l'affectation (ajouter un élément dans une liste, dans un dictionnaire ou changer la valeur d'un tel élément), la comparaison (chaque `if`).
   (c) Écrire une fonction `compare(brin1, brin2)` qui prend deux arguments `brin1` et `brin2` et retourne le rapport du nombre de codons en commun par le nombre de codons total contenu dans ces brins.
   Une manière de calculer ce rapport peut être de compter, pour chaque codon contenu dans `brin1`, le nombre d'occurrences en commun avec `brin2`, puis de sommer ces nombres.
4. Séparation des protéines.
   La méthode précédente de comparaison des occurrences est assez vite limitée. En effet, l'ordre des codons est important pour reconstruire l'information génétique.

(a) Il existe des codons particuliers dont le rôle est de séparer les informations contenus dans l'ADN en morceaux significatifs (encodant des protéines). Ces codons particuliers sont appelés codons STOP. On appellera [1] protéine une suite de codons contenus entre deux codons STOP (sauf dans le cas de la première et de la dernière protéine qui sont délimitées par une extrémité du brin et un codon STOP).

Écrire une fonction `liste_proteines(L)` qui prend comme argument une liste de codons et retourne la liste des protéines (une protéine sera une liste de codons) contenues dans `L`. Pour simplifier l'écriture, on ne considère que «TAG» et «TGA» comme codons STOP.

(b) Comment adapter la méthode de la question 3 pour mesurer la similarité des protéines contenues dans deux brins d'ADN ?

Partie II : Recherche de motifs

Le problème étudié ici est celui de la recherche des occurrences d'une sous-chaîne. Par exemple, la première occurrence de `"ACG"` dans la chaîne `"CGTACGTG"` est à l'indice 3 (la sous chaîne commence à l'indice 3). C'est d'ailleurs la seule occurrence.

1. Quels sont les indices des occurrences de `"ACA"` dans `"ACACA"` ?
2. Écrire une fonction `indices_motif(s, m)` qui prend deux arguments `s` et `m` qui sont des chaînes de caractères, et retourne la liste des entiers `i` tels que, à partir de l'indice `i`, la chaîne `s` commence par `m`.

   Autrement dit, `m` est un préfixe de `s[i:]` (`m` est le début de `s[i:]`).

   Par exemple, `indices_motif("ACTGACG", "AC")` retourne `[0, 4]` car la sous-chaîne `"AC"` apparaît deux fois.
3. On note n_s la longueur de `s` et $n_m < n_s$ la longueur de `m`. Combien de comparaison de caractères (on compte une comparaison par caractère) sont effectuées lors de l'exécution de `indices_motif(s, m)` ? On considère que pour comparer l'égalité de deux chaînes, on compare chacun de leurs caractères.
4. On donne le code suivant

```
1  def cre_tableau(m):
2      T = [0]
3      for i in range(1, len(m)):
4          l = 0
5          for j in range(1, i):
6              if m[0:j] == m[i+1-j:i+1]:
7                  l = j
8          T.append(l)
9      return T
```

   (a) Quel est la longueur de la liste `T` retournée ?

   (b) Décrire par une phrase les chaînes testées à la ligne 6. On attend le terme «sous-chaîne», ainsi que des précisions sur la longueur.

   (c) Que contient `T[i]` ? Justifier en particulier que `T[i] < i` pour $i > 0$.

   (d) Donner en fonction de n_m le nombre de comparaison s entre caractères effectuées lors de l'appel à `cre_tableau(m)`
5. Considérons maintenant la fonction

```
1  def kmp(s, m):
2      T = cre_tableau(m)
3      j = 0
4      res = []
5      for i in range(len(s)):
6          while j > 0 and s[i] != m[j]:
7              j = T[j - 1]
8          if s[i] == m[j]:
9              j = j + 1
10         if j == len(m):
11             res.append (i - (len(m) - 1))
12             j = T[j - 1]
13     return res
```

   (a) Justifier que la boucle `while` (ligne 6) termine toujours.

1. abusivement

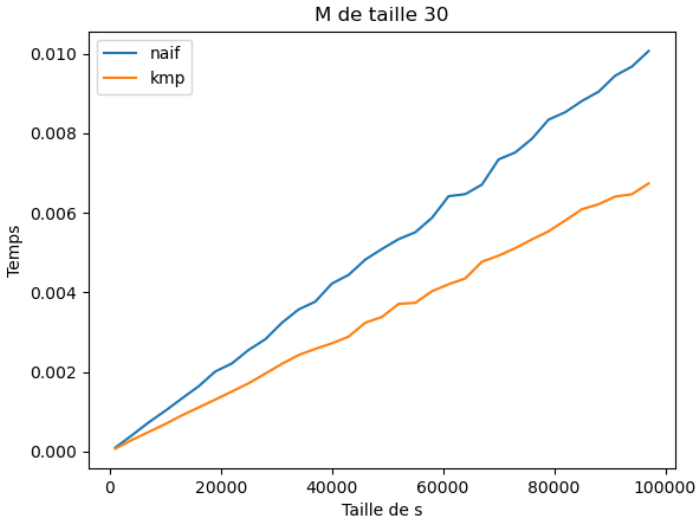

FIGURE 1 – Temps de calcul en fonction de la taille de s

(b) Montrer que si `i` est fixé, `j` vaut 0 et que `m` est un préfixe de `s[i:]` alors `i` sera ajouté à la liste `res`.
On admet que grâce à la construction de `T`, ce résultat vaut même lorsque `j` ne vaut pas 0.

(c) Quel est alors la valeur de retour de la fonction `kmp` ?

(d) On donne l'évolution du temps de calcul en fonction de la taille de `s` (pour `m` de taille fixée à 30) pour les fonction `indices_motif` (légendée «naif») et `kmp` dans la figure 1.
Que conjecturer sur la complexité de la fonction `kmp` ?

Exercice 2 (Une base de donnée)
On considère une base de données d'échantillons d'ADN qui contient deux tables :

— La tables `echantillons` qui contient les champs :
  — `id` : la clé primaire, un entier.
  — `espece` : le nom scientifique de l'espèce dont provient l'échantillon. Du texte.
  — `date` : du texte, au format AAAA-MM-JJ. La date de prélèvement.
— La table `sequences` qui contient les champs :
  — `id` : la clé primaire, un entier.
  — `echantillon_id` : un entier
  — `gene` : du texte, indiquant que ce gène a été identifié dans l'échantillon

1. Donner une requête renvoyant tous les noms d'espèce pour lesquelles un prélèvement a été effectué le 23 janvier 2023.
2. Donner une requête renvoyant le nombre total d'échantillons présents dans la base de données.
3. Donner une requête renvoyant le nom des espèces, le nombre d'échantillon par espèce, pour les espèces présente plus de 10 fois dans la base.
4. Donner une requête renvoyant les nom des espèces, le nom des gènes trouvés, pour les espèces ayant été prélevée après le 1er janvier 2023.
5. Donner une requête renvoyant les noms d'espèces et le nombre de gène trouvés pour chaque espèce, classés par nombre de gène croissant.

Annexe : rappels de syntaxe python

Listes et chaînes

`L` désigne une liste et `s` désigne une chaîne de caractères

— `[]` est la liste vide et `""` est la chaîne vide.

— `L[0]` et `s[0]` sont les premiers éléments.

— Si $i, < j$ sont des entiers naturels, `L[i:j]` et `s[i:j]` sont la liste ou la chaîne composées des éléments d'indices $i, i+1, \ldots, j-1$ de `L` et `s` respectivement.
Dans le cas où i n'est pas donné, il vaut 0 par défaut (on part du premier élément) et dans le cas où j n'est pas donné il vaut `len(L)` par défaut (on s'arrête au dernier élément).

— `L.append` est une fonction qui ajoute son argument à la fin de la liste `L`.

— La concaténation de chaînes de caractères se fait par l'opérateur `+`. Ainsi `s + s` est une chaîne contenant deux copies de `s` mises bout à bout.

Dictionnaires

`d` désigne un dictionnaire.

— `{}` est le dictionnaire vide.

— `d[c]` permet d'accéder à la valeur associée à la clé contenue dans la variable `c`. Par exemple `d["a"]` est la valeur associée à la chaîne de caractère `"a"`.

— `d[c] = v` associe la valeur contenue dans `v` à la clé `c`. Ceci ajoute une paire (clé, valeur) dans `d` ou remplace la valeur associée à `c` suivant que `c` était ou non une clé présente dans `d`

— `c in d` est un booléen indiquant si la clé `c` est présente dans `d`.

— `for c in d:` permet d'obtenir séquentiellement dans la variable `c` toutes les clés contenues dans `d`.