Projet : Autour de l'alignement local

Dans ce projet, nous vous proposons de développer un programme en Python qui réalise différentes fonctionnalités liées à l'alignement local de deux séquences. Les parties 1 et 2 sont obligatoires. La partie 3 est en bonus.

1 Partie 1: Algorithme de Smith et Waterman

L'objectif de cette première partie est de construire l'alignement local entre deux séquences d'ADN, U et V, suivant l'algorithme de Smith et Waterman vu en cours. Cet algorithme fonctionne avec des pénalités de scores linéaires pour les indels. Les coûts pour les opérations d'identité, de substitution et d'indel sont donnés dans un fichier de configuration `score.txt` de la forme

```
# costs values

identity : +2
substitution : -1
indel : -2
```

Question 1 ⋆ *Ecrivez une fonction* `ParseFasta(Fi)` *qui prend en entrée un fichier* `Fi` *contenant deux séquences d'ADN au format Fasta et en extrait les séquences et leurs identifiants. Cette fonction devra gérer correctement les erreurs de syntaxe, c'est-à-dire le cas où le fichier ne contient pas les informations au format attendu.*

```
sequence au format Fasta:

>identifiant_seq1
ACGGTGCGTGGACGTAACGTTGGCACGTGACGCGCT |
CGCAGACTTGATGATATAGCTGCGTGCGCGCGGATT |---> séquence seq1
CGGCATATATTGCCGATGATGATGCACTATTACGGG |
>identifiant_seq2
CAGTGAGAGATATTATATTGCAGTCGTACTGTGACT
CCGGGAGTATGATGTGATGTGCTGTGCATGCTGGGC
```

Question 2 ⋆ *Ecrivez une fonction* `ParseConfig(Fi)` *qui prend en entrée un fichier de configuration pour les coûts d'opération, tel que décrit pour le fichier* `score.txt`*. Cette fonction devra retourner un dictionnaire* `scores` *(voir ci-dessous) avec en clé les noms et en valeurs les entiers correspondants à chacun des trois coûts. Comme précédemment, cette fonction devra gérer correctement le format attendu (attention aux lignes commentées ou vides).*

```
retour de la fonction ParseConfig(Fi)
{'identity ': 2, 'substitution ': -1, 'indel ': -2}
```

Question 3 ⋆ ⋆ *Ecrivez une classe* `Cost` *qui contiendra tous les outils nécessaires au calcul du coût d'une opération dans la matrice de programmation dynamique.*

Un objet de la classe `Cost` s'initialise avec deux paramètres: un dictionnaire contenant les coûts des opérations (identité, substitution, indel) et un alphabet des caractères autorisés (ex: `A,T,C,G`).

Parmi les méthodes et attributs demandés dans cette classe, se trouvent:

- un attribut pour **chaque** coût d'opération,
- un alphabet contenant les caractères autorisés,
- une méthode **privée** intitulée `compare()` permettant de comparer deux caractères et de retourner le coût selon l'opération. Par exemple: `compare("A","")` doit renvoyer le coût d'un indel, et `compare("G","G")` celui d'une identité,
- une redéfinition de `__call__()` qui prendra deux caractères en entrée. Celle-ci permettra d'appeler une instance de la classe `Cost`, et sera en charge de vérifier que les deux caractères fournis sont bien valides et présents dans l'alphabet autorisé. Les deux caractères devront ensuite être transmis à `compare()`, et le résultat retourné.

L'objectif est de pouvoir comparer deux nucléotides de la sorte:

```
scores = {'identity ': 2, 'substitution ': -1, 'indel ': -2}
cost = Cost( scores , "ATCG")
c = cost("A","T")
print(c)
>>> -1
c = cost("A","9")
>>> ValueError: Opération invalide pour ces caractères.
```

Question 4 ⋆ *Ecrivez une fonction* `DPmatrix(U,V,cost)` *qui calcule la matrice de programmation dynamique de l'algorithme de Smith et Waterman pour les deux séquences U et V, en utilisant une instance de Cost pour le calcul du coût de chaque opération.*

Question 5 ⋆ *Ecrivez une fonction* `SimilarityScore` *qui détermine le score de l'alignement local optimal à partir de la matrice précédemment calculée.*

Question 6 ⋆ ⋆ *Ecrivez une fonction* `Align` *qui construit et affiche l'alignement local correspondant au score optimal à partir de deux fichiers Fasta et d'un objet* `Cost`. *Cela se fera par trace back à partir de la matrice de programmation dynamique. Il pourrait s'avérer nécessaire de développer de nouveaux objets ou fonctions pour vous assister.*

Exemple d'affichage pour cette partie

```
>seq_1
TGTTACGG
>seq_2
GGTTGACTA

Alignement score: 8

Alignment:

seq_1   2  GTT-AC  6
           ||| ||
seq_2   2  GTTGAC  7
```

2 Partie 2: Significativité du score d'alignement local

Dans la deuxième partie, vous allez enrichir votre code, pour ajouter une nouvelle fonctionnalité: déterminer la *significativité statistique* d'un score d'alignement. Le calcul de cette significativité est obtenu en construisant des séquences mélangées. Plus précisément:

1. Soient a, c, g, t le nombre de nucléotides `A`,`C`, `G` et `T` contenus dans la séquence U.
2. On génère un grand nombre n de séquences aléatoires contenant chacune a occurrences de `A`, c occurrences de `C`, g occurrences de `G` et t occurrences de `T` (par exemple $n = 200$).
3. Pour chacune de ces séquences, $S_1, \ldots, S_n$, on calcule le score d'alignement entre V et S_i $(1 \leq i \leq n)$. Cela donne une distribution de n scores: $s_1, \ldots, s_n$.
4. On calcule le pourcentage de valeurs parmi les n scores $s_1, \ldots, s_n$, qui sont supérieures ou égales au score initial s. Cela donne la significativité. Plus ce nombre est faible, proche de 0, plus l'alignement initial entre U et V est jugé significatif.

Question 7 ⋆ ⋆ *Ecrivez une fonction* `Significance(V,U,s,n,cost)`*, qui construit n séquences aléatoires de même composition que U, calcule les scores d'alignement local entre ces séquences et la séquence V et indique le pourcentage de séquences dont le score est supérieur ou égal à s. Cette fonction utilisera la fonction* `SimilarityScore` *précédemment définie. Pour la construction des séquences aléatoires, vous pouvez utiliser la méthode* `shuffle()` *du module* `random`*: https://www.w3schools.com/python/module_random.asp .*

Question 8 ⋆ ⋆ *Écrivez une fonction* `DisplayDist` *qui affiche la distribution des scores calculée par la fonction* `Significance` *(voir exemple à suivre). La fonction devra avoir comme paramètres la distribution des scores aléatoires, le score d'alignement original, le nombre de séquences générées et le nombre de modes.*

Question 9 ⋆ ⋆ *Lors du calcul de la significance, une matrice de programmation dynamique est reconstruite à chaque fois avant d'appeler* `SimiliarityScore()`*. Cette méthode est plutôt coûteuse en mémoire, car il n'est pourtant pas nécessaire de stocker entièrement la matrice pour accéder au score d'alignement.*
Écrivez une fonction `DPmatrix_v2()` *qui permettra de renvoyer directement le score d'alignement au lieu d'une matrice, et utilisez la dans votre fonction* `Significance()`*. Cette fonction ne devra utiliser que deux listes correspondant à deux lignes successives dans la matrice, mais ne stockera pas la matrice entière.*

Ce découpage en fonctions du projet est donné à titre indicatif, pour vous aider. Vous pouvez créer d'autre fonctions intermédiaires. Au final, l'exécution de votre programme devra ressembler à ceci:

```
>seq_1
TGTTACGG
>seq_2
GGTTGACTA

Alignement score: 8

Alignment:

seq_1   2  GTT-AC  6
           ||| ||
seq_2   2  GTTGAC  7

Significance of the score: 35.5% (200 sequences)

  score      #
      0      0:
      1      0:
      2      0:
      3      0:
      4      6:==
      5     16:======
      6     77:==============================
      7     30:============
*     8     42:==================
      9     11:====
     10     12:=====
     11      2:=
     12      4:==
```

3 Partie 3: Questions subsidiaires

Code IUPAC. Il peut arriver que les séquences d'ADN données en entrée contiennent des caractères indéterminés, issus du code IUPAC. Par exemple, la présence d'un `N` dans la séquence signifie que le nucléotide présent à cette position est inconnu. Cela peut être `A`, `C`, `G` ou `T`. Le code IUPAC, donné ci-dessous, est ainsi un alphabet à 15 lettres qui permet de représenter n'importe quelle combinaison de nucléotides.

IUPAC	Nucléotide
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
R	`A G`
Y	`C T`
S	`G C`
W	`A T`
K	`G T`
M	`A C`
B	`C G T`
D	`A G T`
H	`A C T`
V	`A C G`
N	`A C G T`

Question 10 ⋆ ⋆ *Modifiez votre programme de manière à permettre que les séquences U et V contiennent des caractères IUPAC. Les valeurs des coûts pour les identités et les substitutions sont définies par les règles suivantes:*

- *si les deux caractères correspondent à des nucléotides (`A`, `C`, `G` ou `T`), les valeurs pour l'identité et la substitution sont les scores habituels.*
- *si l'un des deux caractères au moins est un caractère ambigu, on regarde si l'intersection entre les deux symbôles à aligner est vide. Si elle est vide, le score est celui d'une substitution. Si elle n'est pas vide, le score est 0.*

Les règles pour les indels ne changent pas.

Pénalités de gaps affines. En cours, vous avez vu que les programmes d'alignement de séquences utilisaient en fait un système de score affine pour les indels, avec des pénalités d'*ouverture* et d'*extension de gaps*. L'algorithme pour ce système de scores utilise trois matrices de programmation dynamique.

Question 11 ⋆ ⋆ ⋆ *Reprenez votre code et ajoutez des nouvelles fonctions pour permettre de prendre en compte ce système de score: calcul du score et construction de l'alignement.*