CONCOURS BLANC INFORMATIQUE
AUTOUR DU SÉQUENÇAGE DU GÉNOME

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

La réponse ne doit pas se cantonner à la rédaction de l'algorithme sans explication, **les programmes doivent être expliqués et commentés.**

Dans ce sujet, on s'intéresse à la recherche d'un motif dans une molécule d'ADN. Une molécule d'ADN est constituée de deux brins complémentaires, qui sont un long enchaînement de nucléotides de quatre types différents désignés par les lettres A, T, C et G. Les deux brins sont complémentaires : "en face" d'un 'A', il y a toujours un 'T'et "en face" d'un 'C', il y a toujours un 'G'. Pour simplifier le sujet, on va considérer qu'une molécule d'ADN est une chaîne de caractères sur l'alphabet {A,C,G,T} (on s'intéresse donc seulement à un des deux brins). On parlera de séquence d'ADN.

I. Génération d'une séquence d'ADN

On considère la chaîne de caractère `seq='ATCGTACGTACG'`.

Q1. Que renvoie la commande `seq[3]` ? Que renvoie la commande `seq[2:6]` ?

Les fonctions que nous allons construire par la suite devront prendre en paramètre une chaîne de caractères ne contenant que des 'A','C','G'et'T' (ceci correspond à une séquence d'ADN). Nous allons commencer par construire aléatoirement une séquence d'ADN. Pour générer aléatoirement une séquence d'ADN composée de *n* caractères, on utilisera le principe suivant.

- On commence par créer une chaîne de caractères vide.

- Puis on tire aléatoirement n chiffres compris entre 1 et 4 et

  - si on obtient un 1, alors on ajoute un 'A' à notre chaîne de caractères ;
  - si on obtient un 2, alors on ajoute un 'C' à notre chaîne de caractères ;
  - si on obtient un 3 ,alors on ajoute un 'G' à notre chaîne de caractères ;
  - si on obtient un 4, alors on ajoute un 'T' à notre chaîne de caractères.

- On renvoie la chaîne de caractères ainsi construite.

*Q2. Écrire une fonction `generation()` qui prend en paramètre un entier `n` et qui renvoie une chaîne de caractères aléatoires de longueur `n` ne contenant que des 'A','C','G'et'T'. On pourra utiliser les fonctions `random()` ou `randint()` détaillées en **annexe**.*

*Q3. Ecrire une fonction `pourcentage()` qui prend en argument une séquence d'ADN et renvoie dans une liste le pourcentage de 'A', 'C', 'G' et 'T' présents. **On utilisera forcément un while.***

Q4. Combien y-a-t'il d'itérations lors de l'exécution de votre fonction `pourcentage()`? Donner le nom de la variable assurant que l'algorithme n'a qu'un nombre fini d'étapes.

II. Recherche d'un motif

Soit une chaîne de caractères `S='ACTGGTCACT'`, on appelle sous-chaîne de caractères de `S` une suite de caractères incluse dans `S`. Par exemple, `'TGG'` est une sous-chaîne de `S` mais `'TAG'` n'est pas une sous-chaîne de `S`. **L'objectif est ici de rechercher une sous-chaîne de caractères `M` de longueur `m` appelée motif dans une chaîne de caractères `S` de longueur `n`.**

Il s'agit d'une problématique classique en informatique, qui répond aux besoins de nombreuses applications.
On trouve plus de 100 algorithmes différents pour cette même tâche, les plus célèbres datant des années 1970, mais plus de la moitié ont moins de 10 ans.
Dans cette partie, nous allons d'abord nous intéresser à l'algorithme naïf (**sous-partie II.1**), puis à deux autres algorithmes : l'algorithme de Knuth-Morris-Pratt (**sous-partie II.2**), et un algorithme utilisant une structure de liste (**sous-partie II.3**) et enfin, aux fonctions de hachage (**sous-partie II.4**).

Les différentes sous-parties sont indépendantes.

II.1. Algorithme naïf

Principe de l'algorithme naïf : On parcourt la chaîne. À chaque étape, on regarde si on a trouvé le bon motif. Si ce n'est pas le cas, on recommence avec l'élément suivant de la chaîne de caractères.

Cet algorithme a une complexité en *O(nm)* avec *n*, la taille de la chaîne de caractère et *m*, la taille du motif.

Q5. Écrire une fonction `recherche()` qui à une sous-chaîne de caractères `M` et une chaîne de caractères `S` renvoie -1 si `M` n'est pas dans `S`, et la position de la première lettre de la chaîne de caractères `M` si `M` est présente dans `S`.

Cet algorithme doit correspondre à l'algorithme naïf.

Q6. Combien faut-il d'opérations pour chercher un motif de 50 caractères dans une séquence d'ADN en utilisant l'algorithme naïf ? On supposera qu'une séquence d'ADN est composée de $3 .10^9$ caractères. En combien de temps un ordinateur réalisant 10^{12} opérations par seconde fait-il ce calcul ?

En génétique, on utilise des algorithmes de recherche pour identifier les similarités entre deux séquences d'ADN.
Pour cela, on procède de la manière suivante :

- découper la première séquence d'ADN en morceaux de taille 50 ;

- rechercher chaque morceau dans la deuxième séquence d'ADN.

Q7. En utilisant les calculs précédents, combien de temps faut-il pour un ordinateur réalisant 10^{12} opérations par seconde pour comparer deux séquences d'ADN avec l'algorithme naïf ? Vous semble-t-il intéressant d'utiliser l'algorithme de recherche naïf ?

II.2. Algorithme de Knuth-Morris-Pratt (1970)

Lorsqu'un échec a lieu dans l'algorithme naïf, c'est-à-dire lorsqu'un caractère du motif est différent du caractère correspondant dans la séquence d'ADN, la recherche reprend à la position suivante en repartant au début du motif. Si le caractère qui a provoqué l'échec n'est pas au début du motif, cette recherche commence par comparer une partie du motif avec une partie de la séquence d'ADN qui a déjà été comparée avec le motif. L'idée de départ de l'algorithme de Knuth-Morris-Pratt est d'éviter ces comparaisons inutiles. Pour cela, une fonction annexe qui recherche le plus long préfixe d'un motif qui soit aussi un suffixe de ce motif est utilisée.

Avant d'étudier l'algorithme de Knuth-Morris-Pratt (sous-partie II.2.b) nous allons définir les notions de préfixe et suffixe (sous-partie II.2.a).

II.2.a Préfixe et suffixe

Un préfixe d'un motif M est un motif u, différent de M, qui est un début de M. Par exemple, 'mo' et 'm' sont des préfixes de 'mot', mais 'o' n'est pas un préfixe de 'mot'. Un suffixe d'un motif M est un motif u, différent de M, qui est une fin de M. Par exemple, 'ot' et 't' sont des suffixes de 'mot', mais 'mot' n'est pas un suffixe de 'mot'.

Q8. Donner tous les préfixes et les suffixes du motif 'ACGTAC'.

Q9. Quel est le plus grand préfixe de 'ACGTAC' qui soit aussi un suffixe ? Quel est le plus grand préfixe de 'ACAACA' qui soit aussi un suffixe ?

II.2.b Algorithme de Knuth-Morris-Pratt

Nous rappelons que l'algorithme de Knuth-Morris-Pratt (KMP) est une fonction de recherche qui utilise une fonction annexe prenant en argument une chaîne de caractères `M` dont on notera la longueur `m`. Cette fonction annexe, appelée `fonctionannexe()`, doit permettre, pour chaque lettre à la position `i`, de trouver la longueur du plus grand mot qui est à la fois préfixe et suffixe de `M[0:i+1]`. Le code de `fonctionannexe()` est le suivant :

```
1  def fonctionannexe(M):
2      F=[1]
3      i=1
4      j=0
5      while i < m :
6          if M[i]=M[j] :
7              F.append(j+1)
8              i=i+1
9              j=j+1
10         else
11             if j>0 :
12                 j=F[j-1]
13             else:
14                 F.append(0)
15                 i=i+1
16     return F
```

Q10. Quel est le type de la sortie de la fonction `fonctionannexe()` ?

Q11. Une ou des erreurs de syntaxe s'est (se sont) glissée(s) dans la fonction `fonctionannexe()`. Identifier la ou les erreur(s) et corriger la fonction pour qu'il n'y ait plus de message d'erreur quand on exécute la fonction. Réécrire ***la fonction corrigée sur votre copie.***

Q12. Décrire l'exécution de la fonction `fonctionannexe()` lorsque `M='ACAACA'` en précisant pour les six premiers tours dans la boucle `while`, à la sortie de la boucle, le contenu des variables : `i`, `j` et `F`.

Initialisation : i=1; j=0; F=[1]
Fin du premier passage : i ? j ? F ?
Fin du 2éme passage : i ? j ? F ?
...

Fin du 6ème passage : i ? j ? F ?

II.3. Algorithme utilisant la structure de liste

Une autre possibilité pour chercher un motif dans une chaîne de caractères (ou séquence d'ADN) est de construire une liste contenant tous les sous-motifs de notre chaîne, triés par ordre alphabétique, puis de faire la recherche dans cette liste. Par exemple, à la chaîne `'CATCG'`, on peut associer la liste :
`['C','A','T','G','CA','AT','TC','CG','CAT','ATC','TCG','CATC','ATCG','CATCG']`
que l'on peut ensuite trier pour obtenir la liste:
`['A','AT','ATC','ATCG','C','CA','CAT','CATC','CATCG','CG','G','T','TC','TCG']`.
La première étape de cette méthode est donc de trier une liste.

On ne demande pas ici d'algorithme de tri, on considère que la liste sera déjà triée pour la suite de l'étude. On peut alors faire une recherche dichotomique dans cette nouvelle liste.

*Q13. Écrire une fonction `recherchedichotomique()` de recherche dichotomique d'un caractère `a` dans **une liste L triée** (vous pouvez écrire le code comme a un entier et L une liste de nombre triée, c'est finalement la même chose qu'une liste de chaîne de caractère par ordre alphabétique !) On renverra la position (indice) du premier caractère trouvé et **on ne gérera pas le cas où a n'est pas dans la liste.***

Quel est l'intérêt de ce type d'algorithme (on parlera de complexité) ?

II.4 - Fonction de hachage et évaluation de polynôme

II.4.a Fonction de hachage, algorithme de Karp-Rabin

Certains algorithmes, comme l'algorithme de Karp-Rabin (1987), utilisent une fonction de hachage `h` qui à un motif renvoie une valeur numérique.

Voici un exemple de fonction de hachage :

- à chaque caractère de l'alphabet, on associe une valeur. Ici, on va associer à 'A' la valeur 0, à 'C' la valeur 1, à 'G' la valeur 2 et à 'T' la valeur 3. Pour un motif de taille *n*, on obtient donc une suite de chiffre $a_{n-1}\dots a_1 a_0$. Par exemple, à la chaîne 'TAGC', on lui associe la suite de chiffre 3021 ;

- cette suite de chiffre est considérée comme l'écriture d'un entier en base *b*, où *b* est le nombre de caractères présents dans l'alphabet. On a donc ici $b = 4$;

- on calcule ensuite cet entier en base 10 (on calcule donc $a_{n-1}b^{n-1}+\dots+a_1 b^1+a_0 b^0$) ;

- puis on calcule le reste de la division euclidienne de ce nombre par 13.

Q14. On ne considère que des motifs de taille 3. Que renvoie la fonction de hachage avec les motifs 'CCC', 'ACG', 'GAG'? On détaillera les calculs.

Dans cette fonction de hachage, nous avons besoin de transformer un entier en base *b* en un entier en base 10. On remarque que l'on peut éventuellement faire ce calcul en évaluant un polynôme.

II.4.b Évaluation de polynôme, Algorithme de Hörner

Dans cette sous-partie, nous allons nous intéresser à l'évaluation d'un polynôme et de son coût lorsque l'on compte les multiplications, les additions et les affectations comme des opérations unitaires.

Soit

$$P=\sum_{k=0}^{n} a_k K^k$$

un polynôme représenté par la liste `[a_n , …, a_0 ]`

> *Q15. Écrire une fonction* `eval()` *ayant pour paramètre un polynôme* `P` *(donc une liste de nombres* `[a_n , …, a_0 ]`*) et un nombre* `b`*. Cette fonction doit renvoyer la valeur de* `P` *en* `b`*, c'est-à-dire calculant :*

$$P(b)=\sum_{k=0}^{n} a_k b^k$$

En admettant que le calcul de b^k utilise $k-1$ multiplications, on trouve une complexité quadratique.

On peut être plus astucieux en utilisant l'algorithme de Hörner qui se base sur l'égalité suivante :

$$P(X)=\Big(\big(\ldots\big(\big(a_n X+a_{n-1}\big)X+a_{n-2}\big)X+\ldots\big)X+a_1\Big)X+a_0$$

Plus précisément, pour évaluer P en b, on commence par calculer $a_n \times b + a_{n-1}$, puis on multiplie le résultat par b et on ajoute a_{n-2}, etc. On trouvera alors une complexité linéaire.

> *Q16. Écrire une fonction itérative* `hornerit()` *ayant pour paramètres un polynôme* `P`*, sous forme de liste, ainsi qu'un réel* `b`*, et renvoyant* `P(b)` *en utilisant l'algorithme de Hörner.*

III. Collection Française de Bactéries Phytopathogènes

La Collection Française de Bactéries Phytopathogènes (CFBP) possède deux bases de données :

- l'une, appelée Echantillon, qui permet de stocker les différents échantillons d'ADN ;

- l'autre, appelée Sequence, qui permet de mémoriser quelle personne est responsable de l'obtention de la séquence (i.e. de l'extraction d'un gène particulier dans les différents échantillons d'ADN). Des extraits des tables Echantillon et Sequence sont données par les tableaux 1 et 2.

On considère que le chemin du dossier d'écriture a été renseigné au préalable dans Python par `import os ; os.chdir('\chemin')`

On souhaite tout d'abord écrire un fichier texte pour la base Echantillon.

La première ligne du fichier doit être : `'ADN \t Genre \t \t Espèce \t Sous-espèce \n'`

Le reste du fichier va être rempli à l'aide d'une liste de liste `tab` contenant les informations suivantes :
`tab=[['...'],[309, 'Pseudomonas','syringae','vinorum'], ['...'], [3589,'Pseudomonas','syringae','vignae'], ['...']]`

Chaque sous-liste de `tab` contient donc le numéro de l'ADN (int) le nom du genre, de l'espèce puis de la sous-espèce (str).

> *Q17. Écrire une procédure `echantillon()` ayant pour paramètre le tableau `tab`, et qui permet d'écrire le fichier texte `'echantillon.txt'`. Les lignes du fichier seront les sous-listes de `tab` et chaque élément sera séparé par une tabulation, comme la première ligne. On pensera à convertir les nombres en chaîne de caractère pour la bonne écriture dans le fichier et à sauter des lignes entre chaque sous-liste !*

Le fichier texte aura alors la forme suivante (les 3 points permettent juste ici de ne pas tout montrer) :

```
ADN      Genre          Espèce          Sous-espèce

...

309      Pseudomonas    syringae        vinorum

...

3589     Pseudomonas    syringae        vignae

...
```

On souhaite maintenant récupérer les informations du fichier texte `'sequence.txt'` afin de recouper les informations entre les échantillons et les séquences (non traitées dans ce sujet). Voici un exemple du fichier texte :

```
Code     Date           ADN     Gène   Protocole              Employé

A        '01-03-2018'   309     gyrB   Spilker                Dupont

B        '01-03-2018'   2028    recA   Cesbron and Manceau    Martin
...

AGZ      '10-03-2018'   2028    leuS   Deletoile              Martin

...
```

Chaque ligne du fichier contient donc les éléments séparés par une tabulation comme suit :

```
'Code \t Date \t ADN \t Gène\t Protocole \t Employé \n'
```

> *Q18. Ouvrir le fichier en lecture et récupérer toutes les informations dans une liste de listes de la même forme que tab de la question précédente SANS la première ligne. Les méthodes strip et split sont explicitées en annexe.*

Pour l'exemple illustré ci-dessus le tableau aura la forme suivante :

```
[['A', "'01-03-2018' ", ' 309', 'gyrB', 'Spilker', 'Dupont'], ['B', "'01-03-
2018' ", ' 2028', 'recA', 'Cesbron and Manceau', 'Martin'], ['...'], ['AGZ',
"'10-03-2018' ", ' 2028', 'leuS', 'Deletoile', 'Martin'], ['...']]
```

L'annexe se trouve page suivante.

Annexes

`random.random()` renvoie un nombre flottant tiré aléatoirement dans [0, 1[suivant une distribution uniforme

`random.randint(a,b)` génère un nombre entier tel que $a \leq randint(a,b) < b$

La méthode `strip()` permet d'enlever le dernier caractère d'une chaîne de caractère

La méthode `S.split('sep')` permet de créer, à partir de la chaîne de caractères `S`, une liste contenant les sous-chaines qui se situent entre chaque occurrence du séparateur `'sep'`. Par exemple si `S='cafouillage'` alors `S.split('a')` retourne `['c','fouill','ge']`.