DS1 - INFORMATIQUE

AUTOUR DU SEQUENÇAGE DU GENOME

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 séqeunce 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 sequence d'ADN et renvoie dans une liste le pourcentage de 'A', 'C', 'G' et 'T' présents. **On utilisera forcément un while.***

Q4. Quelle est la complexité de votre fonction `pourcentage()` ? Donner le nom de la variable permettant de montrer la terminaison de l'algorithme (on justifiera le raisonnement).

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 1012 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-partieII.2.b) nous allons définir les notions de préfixe et suffixe (sous-partieII.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'. Uns uffixe 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 le plus grand sous-mot de `M` qui finit par la lettre `M[i]` (c'est donc le plus grand suffixe de `M[:i+1])` qui soit aussi un préfixe de `M`. Le code de `fonctionannexe()` est le suivant :

```
1 def fonctionannexe(M):
2     F=[0]
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. ***Récécrire 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=[0]

Fin du premier passage : i ? j ? F ?
Fin du 2éme passage : i ? j ? F ?
…
Fin du 6ème passage : i ? j ? F ?

Q13. Ecrire une foncion `KMP()` *qui prend en argument* `M` *et* `T`*, 2 chaines de caractères et qui réalise* ***l'agorithme de Knuth-Morris-Pratt, cad, trouver si*** `M` ***est contenu dans*** `T`*. On utlisera bien sûr la* `fonctionannexe()` *afin d'optimiser la recherche (elle permet de savoir où repartir dans la recherche de mot sans faire des comparaisons inutiles) ; si le mot* `M` *est bien contenu dans* `T`*, on renvoie la position du début du mot (l'indice) et si le mot n'est pas trouver, on retourne -1.*

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 lui 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.

Q14. Donner un alogortihme de tri simple (par exemple par sélection) pour trier cette liste (on rappelle qu'il est possible de comparer des chaines de caractères par ordre alphabétique sur Python)

Après avoir obtenu une liste triée, on peut faire une recherche dichotomique dans cette nouvelle liste.

Q15. *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 triées , c'est finalement la même chose qu'une liste de chaine de caractère par ordre alphabétique !) On renverra la position (indice) du premier caractère trouvé et* ***on ne gèrera pas le ca 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.

Q16. 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 ]`

Q17. 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 :

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

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.

Q18. 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.

*Q19. Ecrire une fonction `hornerrec()` pour avoir une fonction **récursive** qui évalue un polynôme 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'obention 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.

Echantillon			
ADN	Genre	Espèce	Sous-espèce
309	Pseudomonas	syringae	morsprunorum
…			
3589	Pseudomonas	syringae	vignae
…			

Tableau 1 - Table recensant toutes les séquences d'ADN dont dispose la CFBP

Sequence					
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
…	…	…	…	…	…

Tableau 2 - Table recensant tous les travaux réalisés à la CFBP en mars 2018

Q20. Définir le but et de la requête(1), écrite en SQL : `SELECT count(*) FROM Sequence WHERE Date='01-03-2018'`

Q21. Écrire en SQL la requête(2) qui donne les ADN de la table Sequence pour les gènes 'IeuS'.

Q22. Écrire en SQL la requête(3) qui permet d'obtenir la liste des espèces étudiées par M.Martin le 10 mars 2018.

Q23. Écrire en SQL la requête(4) permettant d'obtenir le nombre d'échantillons prélevés par chaque employé.

Q24. Ecrire la requête(5) qui permet de connaitre l'employé qui a prélevé le plus d'échantillons.

IV. Exercice supplémentaire !

Soient y une fonction de classe C^2 sur $\mathbb{R}$ et $t_{\min}$ et $t_{\max}$ deux réels tels que $t_{\min} < t_{\max}$. On note I l'intervalle $[t_{\min}, t_{\max}]$. On s'intéresse à une équation différentielle du second ordre de la forme :

$$\forall t \in I, \quad y''(t) = f(y(t)) \tag{II.1}$$

où $f : \mathbb{R} \to \mathbb{R}$, est une fonction continue.

De nombreux systèmes physiques peuvent être décrits par une équation de ce type.

On suppose connues les valeurs $y_0 = y(t_{\min})$ et $z_0 = y'(t_{\min})$. On suppose également que le système physique étudié est conservatif. Ce qui entraine l'existence d'une quantité indépendante de t (temps, énergie, quantité de mouvement, …), notée E, qui vérifie l'équation (II.2) où $g' = -f$.

$$\forall t \in I, \quad \frac{1}{2}y'(t)^2 + g\big(y(t)\big) = E \tag{II.2}$$

A – Mise en forme du problème

Pour résoudre numériquement l'équation différentielle (II.1), on introduit la fonction $z : I \to \mathbb{R}$ définie par $\forall t \in I, z(t) = y'(t)$.

A.1) Montrer que l'équation (II.1) peut se mettre sous la forme d'un système différentiel du premier ordre en (t) et (t), noté (S).

A.2) Soit n un entier strictement supérieur à 1 et $J_n = [\![0, n-1]\!]$. On pose $h = \frac{t_{\max} - t_{\min}}{n-1}$ et $\forall i \in J_n$, $t_i = t_{\min} + ih$ Montrer que, pour tout entier $i \in [\![0, n-2]\!]$,

$$y(t_{i+1}) = y(t_i) + \int_{t_i}^{t_{i+1}} z(t)\,dt \quad \text{et} \quad z(t_{i+1}) = z(t_i) + \int_{t_i}^{t_{i+1}} f(y(t))\,dt \qquad \text{(II.3)}$$

La suite du problème exploite les notations introduites dans cette partie et présente deux méthodes numériques dans lesquelles les intégrales précédentes sont remplacées par une valeur approchée.

B – Schéma d'Euler explicite

Dans le schéma d'Euler explicite, chaque terme sous le signe intégrale est remplacé par sa valeur prise en la borne inférieure.

B.1) Dans ce schéma, montrer que les équations (II.3) permettent de définir deux suites $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$, où y_i et z_i sont des valeurs approchées de $y(t_i)$ et $z(t_i)$. Donner les relations de récurrence permettant de déterminer les valeurs de y_{i+1} et z_{i+1} connaissant y_i et z_i.

B.2) Écrire une fonction `euler` qui reçoit en argument les paramètres `(f,y0,z0,tmin,n,h)` et qui renvoie deux listes contenant les valeurs $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$.

C – Schéma de Verlet

Le physicien français Loup Verlet a proposé en 1967 un schéma numérique d'intégration d'une équation de la forme (II.1) dans lequel, en notant $f_i = f(y_i)$ et $f_{i+1} = f(y_{i+1})$, les relations de récurrence s'écrivent

$$y_{i+1} = y_i + h\,z_i + \frac{h^2}{2} f_i \quad \text{et} \quad z_{i+1} = z_i + \frac{h}{2}(f_i + f_{i+1})$$

C.1) Écrire une fonction `verlet` qui reçoit les mêmes arguments que la fonction `euler` et qui renvoie deux listes contenant les valeurs $(y_i)_{i\in J_n}$ et $(z_i)_{i\in J_n}$.

D – Comparaison Euler vs Verlet

On souhaite ici comparer la qualité de l'approximation d'Euler et de Verlet sur un exemple. On admet que le système physique est conservatif.

$$y''(t) + \omega^2 y = 0 \ , \quad avec\ \omega \in \mathbb{R} \qquad \text{(II.4)}$$

D.1)

a) Donner l'expression de $f(x)$ permettant de réécrire l'équation (II.4) sous la forme de (II.1) et (II.2) puis démontrer que $g(x) = \omega^2 \frac{x^2}{2}$ convient pour l'équation (II.2).

b) Schéma d'Euler : on note E_i la valeur approchée de E à l'instant $t_i, i \in J_n$, calculée en utilisant les valeurs approchées de $y(t_i)$ et $z(t_i)$ données à la question II.B.1. Montrer que $E_{i+1} - E_i = h^2\omega^2 E_i$.

On peut alors montrer de la même manière que dans le *schéma de Verlet*, on a $E_{i+1} - E_i = O(h^3)$.

c) La mise en œuvre du schéma d'Euler et du schéma de Verlet avec les mêmes paramètres donne le résultat de la figure 1 (calculs menés avec $y_0 = 3, z_0 = 0, t_{\min} = 0, t_{\max} = 3, \omega = 2\pi\ et\ n = 100$). Sans calcul, que peut-on conclure sur le schéma numérique d'Euler et sur celui de Verlet ? Quelle solution aurait donné un schéma numérique « exact » ?

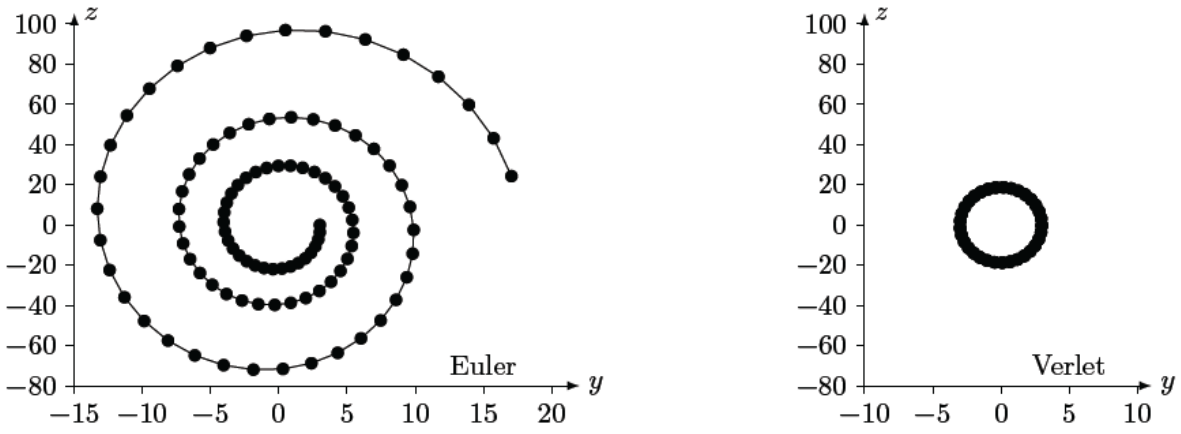

Figure 1

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 $a \leq randint(a,b) < b$