Concours Commun Blanc 2025

ÉPREUVE SPÉCIFIQUE - FILIÈRE TSI

INFORMATIQUE

Mercredi 12 mars 8 h - 11 h

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. ·

RAPPEL DES CONSIGNES

• Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.
• Ne pas utiliser de correcteur.
• Écrire le mot FIN à la fin de votre composition.

Les calculatrices sont interdites

Le sujet est composé de trois parties, toutes indépendantes.
Seul le document réponse est à rendre.

Le sujet comporte :
- le texte du sujet et une annexe, 7 pages,
- le Document Réponse **(DR)** à rendre en fin d'épreuve, 11 pages.

Important : vous pouvez utiliser les fonctions des questions précédentes, même si vous ne les avez pas toutes démontrées.

Vous devez répondre directement sur le Document Réponse, soit à l'emplacement prévu pour la réponse lorsque celle-ci implique une rédaction, soit en complétant les différents programmes en langage Python.

Autour du séquençage du génome

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.

Partie 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.

1. On commence par créer une chaîne de caractères vide.
2. 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.
3. 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. Que fait la fonction *mystere(seq)* qui prend en argument une séquence d' ADN 'seq' (une chaîne de caractères ne contenant que des 'A', 'C', 'G' et 'T') ?
Le code de la fonction *mystere()* se trouve dans le **DR 2**.

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

Partie 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.

Objectif
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 a 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 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()* se trouve à la question **Q11** du **DR 5**.

Q10. Quel est le type de la sortie de la *fonctionfonctionannexe()?*

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 compile la fonction.

Q12. Décrire l'exécution de la *fonctionfonctionannexe()* lorsque M= 'ACAACA' en précisant sur le **DR 6**, pour les six premiers tours dans la boucle while, à la sortie de la boucle, le contenu des variables : *i, jet F.*

Q13. Expliquer et commenter les groupements de lignes de l'algorithme KMP donnés dans le **DR 6**.

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','C,AT','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. Écrire une fonction *triinsertion()* de tri par insertion d' une liste de nombres.

Q15. Comment peut-on adapter la fonction *triinsertion()* à une liste de chaîne de caractères?

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

Q16. Écrire une fonction *recherchedichotomique()* de recherche dichotomique dans une liste de nombres triés.
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 $a_{n-1}...a_1a_0$. 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 $a_{n-1}b^{n-1}+...+ a_1b^1 + a_0b^0$*);*
- puis on calcule le reste de la division euclidienne de ce nombre par 13.

Q17. 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 1' on peut éventuellement faire ce calcul en évaluant un polynôme.

II.4.b - 11.4.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 $P=\sum_{k=0}^{n} a_k X^k$ un polynôme, il sera représenté par la liste $[a_n,...,a_0]$.

Q18. 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 : $P=\sum_{k=0}^{n} a_k b^k$.

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 Horner qui se base sur l'égalité suivante :

$$P(X) = ((... ((a_nX + a_{n-1})X + a_{n-2})X + ...)X+ a_1)X + a_0 .$$

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.

Q19. 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 Horner.

Q20. Compléter la fonction *hornerrec()* pour avoir une fonction récursive qui évalue un polynôme en utilisant l'algorithme de Horner.

Partie 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.**

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	'01-03-2018'	2028	leuS	Deletoile	Martin
...	...	...	...	...	...

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

Q21. Définir le but et le résultat de la requête(l), écrite en SQL:
SELECT count(*) FROM Sequence WHERE Date='0l-03-2018'

Q22. Écrire en SQL la requête(2), permettant d'afficher les numéros d' ADN contenant les gènes leuS.

Q23. Écrire en SQL la requête(3) qui permet d'obtenir la liste des espèces étudiées par M. Martin le 10 mars 2018.

Q24. Écrire en SQL la requête(4) permettant d'obtenir le nombre d'échantillons prélevés par chaque employé.

ANNEXE - Librairie numpy

random() : génère un nombre pseudo-aléatoire compris entre 0 et 1.
randint(a,b): génère un entier aléatoire tel que $a \leq randint(a,b) < b$.

FIN

Nom : ..... **Prénom :** .....

Consignes :
* Compléter l'entête de chaque feuille avant de commencer à composer
* Rédiger avec un stylo non effaçable bleu ou noir
* Rendre les copies dans l'ordre

DOCUMENT RÉPONSE

Q1.

```
1  seq='ATCGTACGTACG'
2  seq[3)
```

Que renvoie cette commande Python?

……..

```
1  seq='ATCGTACGTACG'
2  seq[2:6)
```

Que renvoie cette commande Python ?

……..

Q2.

```
1  def generation(n):
2      seq =
```

Nom : ..... **Prénom :** .....

Q3.

```
1   def mystere(seq):
2       a,b,c,d = 0,0,0,0
3       i = len(seq)-1
4       while i>=0 :
5           if seq[i]=='A':
6               a+=1
7               i-=1
8           elif seq[i]=='C':
9               b+=1
10              i-=1
11          elif seq[i]=='G':
12              c+=1
13              i-=1
14          else :
15              d+=1
16              i-=1
17      return [a*100/len(seq),b*100/len(seq),c*100/len(seq),d*100/len(seq)]
```

Réponse : ……

……

Q4.

Complexité : …….

Terminaison : nom de variable : ……..

justificitation : ……..

……

……

……

……

Nom : ..... **Prénom :** .....

Q5.

```
1  def recherche (M,S) :
```

Q6.

Nombre d'opérations pour chercher un motif de 50 caractères : ……

Temps mis par l'ordinateur : ……..

Nom : ..... **Prénom :** .....

Q7.

Temps pour comparer deux séquences d' ADN : ……

Est-ce intéressant d' utiliser cette méthode ? ……..

……..

Q8.

Préfixes de 'ACGTAC' : …….

Suffixes de 'ACGTAC' :…….

Q9.

Plus grand préfixe de 'ACGTAC' qui soit aussi un suffixe : ……...

Plus grand préfixe de 'ACAACA' qui soit aussi un suffixe : ……...

Q10.

Type de la variable en sortie : ....

Nom : ..... **Prénom :** .....

____Q11._____

Corriger la fonction suivante pour qu' il n'y ait plus de message d' erreur quand on compile la fonction.

```
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
```

____Q12._____

Initialisation : i= l ; j=0 ; F=[0]

Fin du premier passage dans la boucle while : i= … ; j= … ; F= …… ;

Fin du deuxième passage dans la boucle while : i= … ; j= … ; F= …… ;

Fin du troisième passage dans la boucle while : i= … ; j= … ; F= …… ;

Fin du quatrième passage dans la boucle while: i= … ; j= … ; F= …… ;

Fin du cinquième passage dans la boucle while : i= … ; j= … ; F= …… ;

Fin du sixième passage dans la boucle whi le : i= … ; j= … ; F= …… ;

Nom : ..... **Prénom :** .....

Q13.

Algorithme KMP :

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

Explication de la ligne 2 : ……

……..

Explication des 1ignes 3 et 4 : …….

……..

Quelles lignes correspondent au cas où on a trouvé le mot ? ……..

Que fait le programme dans ce cas? …….

……..

Quelles lignes correspondent au cas où on a trouvé deux lettres identiques? …….

……..

Que fait le programme dans ce cas? …….

……..

Nom : ..... **Prénom :** .....

____Q14._____

```
1   def triinsertion(L):
```

____Q15._____

……..

……..

……..

……..

Nom : ..... **Prénom :** .....

____Q16._____

```
1  def recherchedichotomique(a,L):
```

Intérêt de ce type d'algorithme: …….

……..

Nom : ..... **Prénom :** .....

Q17.

h('CCC') : ……

……..

h(' ACG·') : ……

……..

h('GAG') : ……

……..

Q18.

```
1  def eval(P,b):
```

Nom : ..... **Prénom :** .....

Q19.

```
1  def hornerit(P,b) :
```

Q20.

```
1      def hornerrec(P,b):
2          if ……...
3              return …….
4          else :
5              s = P[len(P)-1]
6              s1 = P[0: len(P)-1]
7              return ……..
```

Q21.

But de la requête(l) : ……...

……..

Nom : ..... **Prénom :** .....

Q22.

Requête(2) =

Q23.

Requête(3) =

Q24.

Requête(4) =