teaching sciences for innovation

Proposition de corrigé

Concours : Concours Commun INP

Année : 2020

Filière : TSI

Épreuve : Informatique

Ceci est une proposition de corrigé des concours de CPGE, réalisée bénévolement par des enseignants de Sciences Industrielles de l'Ingénieur et d'Informatique, membres de l'UPSTI (Union des Professeurs de Sciences et Techniques Industrielles), et publiée sur le site de l'association :

https://www.upsti.fr/espace-etudiants/annales-de-concours

A l'attention des étudiants

Ce document vous apportera des éléments de corrections pour le sujet traité, mais n'est ni un corrigé officiel du concours, ni un corrigé détaillé ou exhaustif de l'épreuve en question.

L'UPSTI ne répondra pas directement aux questions que peuvent soulever ces corrigés : nous vous invitons à vous rapprocher de vos enseignants si vous souhaitez des compléments d'information, et à vous adresser à eux pour nous faire remonter vos éventuelles remarques.

Licence et Copyright

En cas de doute, n'hésitez pas à nous contacter à : corrigesconcours@upsti.fr.

Informez-vous !

Retrouvez plus d'information sur les Sciences de l'Ingénieur, l'orientation, les Grandes Ecoles ainsi que sur les Olympiades de Sciences de l'Ingénieur et sur les Sciences de l'Ingénieur au Féminin sur notre site : www.upsti.fr

L'équipe UPSTI

Autour du séquençage du génome

Corrigé UPSTI

Partie I – Génération d'une séquence d'ADN

```
from numpy.random import randint
seq ='ATCGTACGTACG'
```

Question 1 Que renvoie la commande `seq[3]` ? Que renvoie la commande `seq[2:6]` ?

```
>>>seq[3]
'G'
>>>seq[2:6]
'CGTA'
```

Question 2 Écrire une fonction `generation()`

```
def generation(n):
    Alphabet =['A', 'C', 'G', 'T']
    seq =''
    for _ in range(n):
        i =randint(1, 5) # cf entête pour pouvoir utiliser la fonction ainsi
        seq =seq +Alphabet[i -1]
    return seq

>>>L =generation(12)
>>>print(L)
CTTTCCGCTGGA
```

Question 3 Que fait la fonction `mystere(seq)` ?

```
def mystere(seq):
    a, b, c, d =0, 0, 0, 0
    i =len(seq) -1
    while i >=0:
        if seq[i] =='A':
            a +=1
            i -=1
        elif seq[i] =='C':
            b +=1
            i -=1
        elif seq[i] =='G':
            c +=1
            i -=1
        else:
            d +=1
            i -=1
    return [a *100 / len(seq), b *100 / len(seq), c *100 / len(seq), d *100 / len(seq)]

>>>mystere(L)
[8.333333333333334, 33.333333333333336, 25.0, 33.333333333333336]
```

La fonction `mystere(seq)` renvoie la liste des proportions en % des lettre `'A'`, `'C'`, `'G'` et `'T'` dans la séquence `seq` dans cet ordre.

Question 4 Quelle est la complexité de la fonction `mystere()` ?

La boucle `while` est parcourue n fois avec n la longueur de la séquence `seq`. Dans cette boucle on réalise entre 1 et 3 tests et on réalise 2 incrémentations. On a donc $2n$ opérations et $2n$ tests en moyenne. On a donc une complexité en $\mathcal{O}(n)$.

La terminaison de la fonction est montrée avec la variable `i` qui est un variant de boucle. A chaque passage dans la boucle `while`, `i` est décrémenté de 1, `i` finira donc par être négatif, ce qui terminera l'algorithme.

Partie II – Recherche d'un motif

II.1 Algorithme naïf

Objectif

Rechercher une sous-chaîne de caractères `M` de longueur m appelée motif dans une chaîne de caractères `S` de longueur n.

```
S ='ACTGGTCACT'
```

Question 5 Écrire une fonction `recherche()`

Le document réponse propose une fonction `recherche(M, T)`, mais d'après l'énoncé ce serait plutôt : `recherche(M, S)`.

```
def recherche(M,S):
    m, n =len(M), len(S)
    present =-1 # on renvoie -1 si aucun motif n'a été trouvé
    for i in range(n -m +1): # on arrête si le nombre de caractères restant est inférieur àcelui du motif
        k =0
        commun =0
        while k <m and present ==-1 :
            if M[k] ==S[i +k]:
                commun +=1
            k +=1
        if commun ==m :
            present =i # on interrompt la boucle àla première occurence de M dans S
    return present

>>>recherche('GGT', S)
3
```

Question 6 Combien faut-il d'opérations pour chercher un motif de 50 caractères dans une séquence d'ADN en utilisant l'algorithme naïf ?

On a une complexité donnée en $\mathcal{O}(nm)$, soit pour une séquence de 50 caractères à chercher dans une séquence d'ADN de $3 \cdot 10^9$ caractères, un nombre de $1{,}5 \cdot 10^{11}$ opérations.

Si l'ordinateur réalise 10^{12} opérations par seconde, il faut donc 0,15s pour réaliser le calcul.

Question 7 Combien de temps faut-il pour un ordinateur pour comparer deux séquences d'ADN avec l'algorithme naïf ?

Dans une séquence d'ADN de $3 \cdot 10^9$ caractères, il y a $6 \cdot 10^7$ séquences de 50 caractères. L'ordinateur comparera les 2 séquences en $0{,}15 \times 6 \cdot 10^7$s , soit $9 \cdot 10^6$s , ce qui représente une centaine de jours!!!

Cela n'est évidemment pas envisageable.

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

II.2.a Préfixe et suffixe

Question 8 Donner tous les préfixes et les suffixes du motif `'ACGTAC'`.

Le mot `'ACGTAC'` a les préfixes suivants :

- `'A'`
- `'AC'`
- `'ACG'`
- `'ACGT'`
- `'ACGTA'`

Ses suffixes sont :

- `'C'`
- `'AC'`
- `'TAC'`
- `'GTAC'`
- `'CGTAC'`

Question 9 Quel est le plus grand préfixe qui soit aussi un suffixe ?

Le plus grand préfixe de `'ACGTAC'` qui soit aussi suffixe est `'AC'`.

Le plus grand préfixe de `'ACAACA'` qui soit aussi suffixe est `'ACA'`.

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

Question 10 Quel est le type de la sortie de la fonction `fonctionannexe()` ?

la fonction `fonctionannexe()` renvoie une variable de type `list` qui contient des entiers `int`.

Question 11 Identifier la ou les erreur(s) et corriger la fonction `fonctionannexe()`.

Il y a 3 erreurs de syntaxe qui empêchent l'exécution du code (cf code corrigé ci-après) :

- la variable `m` n'est pas définie,
- la comparaison dans le premier `if` doit se faire avec l'opérateur == et non =,
- il manque les `:` à la fin de la ligne du premier `else`.

```
1  def fonctionannexe(M):
2      F =[0]
3      i =1
4      j =0
5      m =len(M) # variable m àdéfinir
6      while i <m:
7          if M[i] ==M[j]: # signe '==' pour la comparaison
8              F.append(j +1)
9              i =i +1
10             j =j +1
11         else: # mettre ':' àla fin de la ligne
12             if j >0:
13                 j =F[j -1]
14             else:
15                 F.append(0)
16                 i =i +1
17     return F
18
19 >>>fonctionannexe('ACAACA')
20 [0, 0, 1, 1, 2, 3]
```

Question 12 Décrire l'exécution de la fonction `fonctionannexe()`.

M='ACAACA'

Initialisation : i = 1 ; j = 0 ; F = [0]

1. Fin du premier passage dans la boucle while : i = 2 ; j = 0 ; F = [0, 0]
2. Fin du deuxième passage dans la boucle while : i = 3 ; j = 1 ; F = [0, 0, 1]
3. Fin du troisième passage dans la boucle while : i = 3 ; j = 0 ; F = [0, 0, 1]
4. Fin du quatrième passage dans la boucle while : i = 4 ; j = 1 ; F = [0, 0, 1, 1]
5. Fin du cinquième passage dans la boucle while : i = 5 ; j = 2 ; F = [0, 0, 1, 1, 2]
6. Fin du sixième passage dans la boucle while : i = 6 ; j = 3 ; F = [0, 0, 1, 1, 2, 3]

Question 13 Expliquer et commenter les groupements de lignes de l'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 du code :

- **Explication de la ligne 2** : on appelle la fonction fonctionannexe avec en paramètre la chaîne de caractères M et on stocke la liste résultat dans la variable F,
  - ◇ la liste F telle que F[i] correspond à la longueur du sous-mot préfixe de M et suffixe de M[:i+1]
- **Explication des lignes 3-4** : on initialise les variables i et j à 0,
- **Cas où on a trouvé le mot** : lignes 7-8

- ◇ dans ce cas le programme renvoie la position de la première lettre du mot identifié, c'est-à-dire i - j
- **Cas où on trouve 2 lettres identiques** : ligne 6 à 11
  - ◇ dans ce cas on teste si on est à la fin du mot M, alors on a trouvé le mot, sinon on incrémente i et j pour tester la lettre suivante
- **Cas où on trouve 2 lettres différentes** : ligne 12 à 16
  - ◇ dans ce cas on teste si j est nul et on incrémente i pour tester le même mot à la position suivante, sinon on remplace j par F[j - 1] pour repartir ne pas retester un sous-mot déjà testé.

II.3 Algorithme utilisant la structure de liste

Question 14 Écrire une fonction `triinsertion()` de tri par insertion d'une liste de nombres.

```
1  def triinsertion(L):
2      n =len(L)
3      for i in range(1, n):
4          j =i
5          while L[j] <L[j -1] and j >0:
6              L[j -1], L[j] =L[j], L[j -1]
7              j -=1
8      return L
```

Question 15 Comment peut-on adapter la fonction `triinsertion()` à une liste de chaîne de caractères ?

La fonction `triinsertion()` telle que présentée ci-dessus n'a pas besoin d'être adaptée pour une liste de chaînes de caractères puisque l'opérateur de comparaison < fonctionne sur les chaînes de caractères. Ainsi `'a'<'b'` renvoie `True`.

Question 16 Écrire une fonction `recherchedichotomique()` de recherche dichotomique dans une liste de nombres triés.

Nous avons fait le choix d'écrire une fonction qui renvoie `True` ou `False` en fonction de la présence ou non de `a` dans `L`.

La recherche dichotomique est intéressante car elle est assez rapide, pour une liste de taille n la complexité est en $\mathcal{O}(\log n)$, ce qui est bien plus intéressant qu'une recherche naïve linéaire qui aurait une complexité en $\mathcal{O}(n)$.

```
1  def recherchedichotomique(a, L):
2      n =len(L)
3      deb, fin =0, len(L) -1
4      while fin -deb >1:
5          mil =(deb +fin) // 2
6          if L[mil] ==a:
7              return True
8          elif L[mil] <a:
9              deb =mil
10         else:
11             fin =mil
12     return False
13
14 >>>Liste=['A','AT','ATC','ATCG','C','CA','CAT','CATC','CATCG','CG','G','T','TC','TC','TCG']
15 >>>recherchedichotomique('ATCG',Liste)
16 True
```

II.4 Fonction de hachage et évaluation de polynôme

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

Question 17 Que renvoie la fonction de hachage avec les motifs `'CCC'`, `'ACG'`, `'GAG'` ?

- `h('CCC')` **renvoie 8**
  - ◇ On associe la suite de chiffres 111 en base $b = 4$
  - ◇ On obtient en base 10 : $1 \cdot 4^2 + 1 \cdot 4^1 + 1 \cdot 4^0 = 21$
  - ◇ On prend le reste de la division euclidienne par 13 pour obtenir $21\%13 = 8$
- `h('ACG')` **renvoie 6**
  - ◇ On associe la suite de chiffres 012 en base $b = 4$
  - ◇ On obtient en base 10 : $0 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0 = 6$
  - ◇ On prend le reste de la division euclidienne par 13 pour obtenir $6\%13 = 6$
- `h('GAG')` **renvoie 8**
  - ◇ On associe la suite de chiffres 202 en base $b = 4$
  - ◇ On obtient en base 10 : $2 \cdot 4^2 + 0 \cdot 4^1 + 2 \cdot 4^0 = 34$
  - ◇ On prend le reste de la division euclidienne par 13 pour obtenir $34\%13 = 8$

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

Question 18 Écrire une fonction `eval()` ayant pour paramètre un polynôme P et un nombre b. Cette fonction doit renvoyer la valeur de P en b.

```
def eval(P, b):
    n =len(P)
    res =0
    for i in range(n):
        res +=P[i] *b **(n -1 -i)
    return res
```

Question 19 Écrire une fonction itérative `hornerit()` renvoyant $P(b)$ en utilisant l'algorithme de Hörner

```
def hornerit(P, b):
    n =len(P)
    res =P[0]
    for i in range(1, n):
        res =res *b +P[i]
    return res
```

Question 20 Compléter la fonction `hornerrec()` pour avoir une fonction récursive qui évalue un polynôme en utilisant l'algorithme de Hörner.

```
def hornerrec(P, b):
    if len(P) ==1:
        return P[0]
    else:
        s =P[len(P) -1]
        sl =P[0:len(P) -1]
        return s +b *hornerrec(sl, b)
```

Partie III – Collection Française de Bactéries Phytopathogènes

Question 21 Définir le but et le résultat de la requête(1), écrite en SQL.

```
SELECT count(*) FROM Sequence WHERE Date='01-03-2018'
```

La requête renvoie le nombre de séquences de la table extraites le 1er mars 2018. Sur la vue de de la table sequence que l'on nous donne le résultat serait 2, mais on ne sait pas si toutes les lignes de cette date nous sont données.

Question 22 Écrire en SQL la requête(2), donnée en algèbre relationnel

$$\pi_{ADN}\left(\sigma_{Gène='leuS'}(Sequence)\right)$$

```
SELECT ADN FROM Sequence WHERE Gène = 'leuS';
```

Question 23 Écrire en SQL la requête(3) qui permet d'obtenir la liste des espèces étudiées par M. Martin le 10 mars 2018.

```
SELECT DISTINCT Espèce
FROM Sequence AS S JOIN Echantillon AS E ON E.ADN = S.ADN
WHERE Employé = 'Martin' AND Date='10-03-2018'
;
```

Question 24 Écrire en SQL la requête(4) permettant d'obtenir le nombre d'échantillons prélevés par chaque employé.

```
SELECT Employé, COUNT (DISTINCT E.ADN) AS ADN
FROM Sequence AS S JOIN Echantillon AS E ON E.ADN = S.ADN
GROUP BY Employé
;
```