

Pesés et hachés

INFORMATIQUE COMMUNE - Devoir n° 1 - Olivier Reynet

Consignes :

1. utiliser une copie différente pour chaque partie du sujet,
2. écrire son nom sur chaque copie,
3. écrire de manière lisible et intelligible,
4. préparer une réponse au brouillon avant de la reporter sur la feuille.

R Pour répondre aux questions, il est autorisé d'utiliser les fonctions définies dans les questions précédentes.

A Pesées

a Préliminaires

A1. À quel type de données d'entrées la fonction `mystery` peut-elle être appliquée ? Préciser ce que contient la variable `c` après la ligne 5 et ce que contient la variable renvoyée par la fonction.

```
1 def mystery(t):
2     v_max = max(t)
3     c = [0 for _ in range(v_max + 1)]
4     for e in t:
5         c[e] += 1
6     output = []
7     for v in range(v_max + 1):
8         for j in range(c[v]):
9             output.append(v)
10    return output
```

b Optimisation d'un processus de pesée

Dans un laboratoire, il est nécessaire de peser précisément des objets à l'aide d'une balance de Rober-
val (à plateau) et d'un jeu de masses standards (1g, 2g, 5g, 10g, etc.) disponibles **en nombre illimité**.
L'objectif est de déterminer la masse M d'un objet donné **en minimisant le nombre de masses stan-
dards utilisées**.

En Python, on modélise le jeu de masse standards par une liste donnée.

Par exemple : `masses_standards = [1, 20, 5, 10, 100, 2, 50]`.

c Approche naïve

- A2.** Proposer, **en français**, un algorithme glouton **simple** pour trouver une solution à ce problème. Détailler ensuite son application à la main pour un objet \mathcal{O} de masse 139g avec les masses standards [1, 20, 5, 10, 100, 2, 50].
- A3.** Appliquer **à la main** l'algorithme précédent à l'objet \mathcal{B} de masse 137g et en utilisant les masses suivantes : [1, 24, 7, 63, 72, 11, 47]. Conclure.
- A4.** Écrire une fonction de signature `pesee_gloutonne(M: int, masses: list[int]) -> int` qui implémente votre algorithme et renvoie le nombre de masses standards utilisées pour peser l'objet.

d Programmation dynamique

Minimiser le nombre de masses utilisées pour peser est un problème qui s'exprime naturellement dans le cadre de la programmation dynamique.

Soit $S(i, M)$, le nombre minimum de masses devant être utilisées pour peser un objet de masse M avec les i premières masses d'un jeu de masses standards. La masse i possède une masse m_i .

Les équations de Bellman s'écrivent :

$$S(i, M) = \begin{cases} 0 & \text{si } M = 0 \\ +\infty & \text{si } i = 0 \text{ et } M \neq 0 \\ \min(1 + S(i, M - m_i), S(i - 1, M)) & \text{si } m_i \leq M \\ S(i - 1, M) & \text{sinon} \end{cases} \quad (1)$$

- A5.** Justifier les équations précédentes.
- A6.** En utilisant la programmation dynamique ascendante, c'est-à-dire en construisant un tableau de résolution, écrire une fonction de signature `pesee_dyn_asc(M, masses)` qui renvoie la solution optimale au problème $S(n, M)$, si l'objet a pour masse M et le jeu de masses standards possède n masses. On pourra utiliser la bibliothèque `math` en l'important correctement et la constante `inf` de cette bibliothèque qui peut représenter l'infini. Cette constante est compatible avec l'opérateur `min`.
- A7.** Quelle est la complexité de la fonction `pesee_dyn_asc`? Justifier!
- A8.** En utilisant la programmation dynamique descendante, c'est-à-dire en utilisant la récursivité et la mémoïsation, écrire une fonction de signature `pesee(M, masses, memo)` qui renvoie la solution optimale au problème $S(n, M)$. On pourra utiliser le tranchage (slicing) pour modifier la taille du jeu de masses. Le paramètre `memo` est de type dictionnaire : il est transmis à chaque appel récursif. Bien définir les clefs et les valeurs de ce dictionnaire.
- A9.** On suppose qu'on dispose d'une fonction qui construit le tableau de résolution `s` de la programmation dynamique ascendante. Écrire une fonction de signature `details(s: list[list[int]], masses: list[int]) -> list` qui renvoie le détail des masses utilisées pour une certaine pesée. Cette fonction parcourt «à l'envers» le tableau de résolution.

B Hachés

On cherche à créer une fonction de hachage polynomiale adaptée à l'étude du génome, c'est-à-dire des séquences de nucléotides $\{A, C, G, T\}$ comme 'GAGCT' par exemple.

Soit l'application $\psi : \{A, C, G, T\} \rightarrow \{0, 1, 2, 3\}$ définie par : $\psi(A) = 0, \psi(C) = 1, \psi(G) = 2$ et $\psi(T) = 3$.

B10. Écrire une fonction de signature `psi(c)` qui convertit une lettre `c` en un entier conformément à la définition de ψ . Lorsque la lettre n'est pas une nucléotide, la fonction renvoie `None`. Par exemple, `psi('G')` renvoie l'entier 2.

On choisit de transformer une séquence `s` en un nombre en base 4. Par exemple, pour la séquence `'GAGCT'`, on obtient :

$$20213_4 = 2 \times 256 + 0 \times 64 + 2 \times 16 + 1 \times 4 + 3 = 551_{10}$$

On définit donc la fonction de hachage polynomiale $h(s)$ comme suit :

$$h(s) = \sum_{i=0}^{|s|-1} \psi(s_i) \cdot 4^{|s|-1-i}$$

où s_i est une nucléotide (lettre) de la séquence `s` et $|s|$ la longueur de la séquence.

B11. Écrire une fonction de signature `h(s)` qui renvoie le nombre entier correspondant à la séquence `s` via la fonction de hachage `h`.

B12. Soit deux séquences de nucléotides différentes et non vides S_1 et S_2 . Est-il possible d'avoir deux hachés identiques, c'est-à-dire $h(S_1) = h(S_2)$?

L'entier `h('GCATGCTACGTAGCTATGGCCATGACAACACTACT')` est très grand : il est codé sur plus de 64 bits en Python. Or, les gènes sont souvent des séquences assez longues.

B13. Écrire une fonction de signature `h(s,M)` qui garantit que l'entier $h(s)$ est dans l'intervalle $\llbracket 0, M \rrbracket$. Justifier que les nombres générés restent raisonnablement grands à tout instant du calcul.

B14. Écrire une fonction de signature `generer_seq(n)` qui renvoie une séquence aléatoire de nucléotides de longueur `n` sous la forme d'une chaîne de caractères. On pourra utiliser la fonction `choice` du module `random`, en l'important correctement. On rappelle que `choice('ZORGLUB')` renvoie aléatoirement une lettre de la chaîne `'ZORGLUB'`.

B15. Écrire une fonction de signature `compter_nuc(s)` qui renvoie le nombre de nucléotides d'un certain type présent dans la séquence `s` sous la forme d'un dictionnaire. Par exemple, `compter_nuc('AGATGC')` renvoie `{ 'A': 2, 'G': 2, 'T': 1, 'C': 1 }`.

(R) Dans la suite de l'exercice, on appelle $h_2(s,M)$ le haché de la séquence `s`. On choisira systématiquement $M = 1000007$.

On dispose d'une liste de séquences génétiques. On souhaite les regrouper d'après leur nombre de nucléotides. Par exemple, les séquences `'AGATCA'`, `'CATA'` et `'TACA'` comportent le même nombre de `'A'`.

B16. Écrire une fonction de signature `grouper(L, type_nuc)` qui renvoie un dictionnaire dont les clefs sont le nombre nucléotides de type `type_nuc` dans la séquence et les valeurs associées les listes des hachés de ces séquences. Par exemple, pour la liste de séquences :

`['GATA', 'ATCCCA', 'TCTCA', 'GTTTA', 'GGTTCTAT', 'TCGTCGGCG']`,
`grouper(L, 'A')` renvoie `{2 : [140, 852], 1 : [884, 764, 44915], 0 : [224678]}`.

C Composition chimique

On dispose de fichiers csv comportant sur chaque ligne :

- une séquence de nucléotides,

- le nom de la séquence,
- une probabilité associée à la séquence.

Voici le début d'un fichier, pour l'exemple :

```
CCCTATA,wsd_i,0.7558656904544971
ACGGGTT,edqusj,0.0735733010898798
AGTGTCA,cypuguoapn,0.8885546844586558
ACCGCGCAAC,cfsj,0.2650729216960316
GCGG,nkif,0.9296453182568237
AAGGAAAA,wgci,0.2621788156014817
```

R Les fonctions suivantes permettent l'ouverture, la lecture et l'écriture des fichiers textes.

- `open(filename: str, mode: str)` permet d'ouvrir un fichier mode en lecture ('r') ou en lecture/écriture ('rw')) et renvoie un objet de type `file`.
- `readline()` **appliquée à un objet** de type `file`, lit et renvoie une ligne sous la forme d'une chaîne de caractères.
- `readlines()` **appliquée sur un objet** de type `file`, lit toutes les lignes du fichier et renvoie une liste de chaînes de caractères. Chaque élément de la liste est une ligne du fichier.
- `write(line: str)` **appliquée à un objet** de type `file` écrit la chaîne de caractères `line` dans le fichier.
- `close()` **appliquée à un objet** de type `file` permet de fermer l'objet fichier et de libérer la ressource au niveau de l'OS.
- `split(d:str)` **appliquée à une chaîne de caractères** permet de découper la chaîne de caractères selon un délimiteur `d`. Elle renvoie donc une liste de chaînes de caractères, chaque élément étant une chaîne située entre les délimiteurs `d`.

C17. Écrire une fonction de signature `import_seq(filename: str) -> list[str]` qui renvoie la liste des séquences de nucléotides importées d'un fichier formaté comme ci-dessus.

C18. Écrire une fonction de signature `trier(t)` qui trie une liste par ordre décroissant. Cette fonction implémente l'algorithme du tri par insertion et procède en place.

C19. Donner les complexités dans le pire et le meilleur des cas de la fonction `trier`. Justifier votre réponse.

R La fonction `tuple` transforme une liste d'éléments en un tuple contenant ces mêmes éléments. Par exemple, `tuple([3,4,5])` renvoie `(3,4,5)`.

R On rappelle que Python est capable de comparer deux caractères et que cette opération est élémentaire de complexité $O(1)$. Par exemple, `'H' < 'G'` vaut `False`.

On dispose du code suivant :

```
1 def signature(seq):
2     t = [c for c in seq]
3     trier(t)
4     return tuple(t)
```

C20. Appliquer la fonction `signature` aux séquences `'GCATA'` et `'CAGAT'` et expliciter les résultats. Justifier votre réponse.

On souhaite compter les séquences qui ont même composition chimique.

C21. Écrire une fonction de signature `meme_composition(data: list[str]) -> dict` dont le paramètre est une liste de séquences de nucléotides. Cette fonction renvoie un dictionnaire dont les clefs sont les signatures des séquences et les valeurs associées le nombre de séquences composées. Par exemple, `meme_composition(["AAA", "ACG", "GAC", "CAG"])` renvoie `{('A', 'A', 'A'): 1, ('G', 'C', 'A'): 3}`.

C22. Quelle est la complexité de la fonction `meme_composition` dans le pire des cas?

C23. Pourquoi le résultat de la fonction `signature` peut-il être une clef d'un dictionnaire Python?

On souhaite maintenant sauvegarder ces résultats dans un fichier.

C24. Écrire une fonction de signature `compo_vers_fichier(filename: str, data: list[str]) -> None` qui inscrit dans un fichier la composition des séquences et leur occurrence. Par exemple, le début d'un tel fichier pourra être :

```
('T', 'A', 'A', 'A'),4
('T', 'G', 'G', 'C', 'C', 'C', 'C', 'A'),3
('T', 'T', 'G', 'A'),11
('T', 'T', 'G', 'C', 'C', 'A', 'A'),5
('T', 'G', 'G', 'C', 'A', 'A', 'A'),6
```

R On rappelle que pour écrire dans un fichier, il faut utiliser la fonction `f.write(s)`, où `f` est un objet fichier ouvert et `s` est la chaîne de caractères à écrire. Le caractère `'\n'` permet d'aller à la ligne. La fonction `str` permet de transtyper un objet en chaîne de caractères.