Concours Centrale-Supélec — Sujet 03 — MPI

Sous-séquences ADN

Proposition de corrigé (avec les remarques du jury)

Nicolas Pécheux

☑

Une proposition de correction est indiquée dans un encadré vert.

ℹ

Les remarques du jury que l'on peut trouver sur le site

https://www.concours-centrale-supelec.fr/

sont reportées dans un encadré bleu.

1 Introduction

Dans ce sujet on s'intéresse à la représentation, à l'extraction et au décompte de k-mers, c'est-à-dire de facteurs de longueur k, de chaînes d'ADN.

1.1 Contexte

Les progrès techniques dans le séquençage de l'ADN permettent aujourd'hui de connaître le génome de nombreux organismes vivants. La reconstitution du génome s'effectue généralement à partir d'un séquençage de l'ADN d'un organisme, ce qui produit un grand nombre de sous-séquences, appelées *lectures*, qui sont largement redondantes et se chevauchent partiellement. Il est ensuite nécessaire, à la manière d'un immense puzzle, de réassembler ces sous-séquences pour reconstituer la séquence complète. Le séquençage est cependant loin d'être parfait et les lectures produites comportent systématiquement des erreurs, certaines lettres pouvant être remplacées, supprimées ou ajoutées.

Depuis 2009, la goélette *Tara* sillonne les océans en collectant des micro-organismes en divers endroits du globe. On obtient alors des milliards de lectures correspondant à un mélange de nombreuses espèces, que l'on apelle des *métagénomes*. Les données de séquençage ainsi collectées peuvent difficilement être assemblées avec les techniques usuelles et la comparaison avec des espèces déjà connues est limitée. Il reste cependant intéressant de pouvoir se dôter d'outils afin de pouvoir comparer deux ou plusieurs métagénomes.

Une idée prometteuse est la suivante. Si deux séquences sont similaires, alors elle partagent un grand nombre de leurs sous-séquences de taille k, appelées k-mers. Inversement, deux séquences qui partagent beaucoup de k-mers peuvent être considérées comme similaires. Ainsi, la similarité de deux séquences peut être estimée par le nombre de k-mers en commun.

Il est donc nécessaire de disposer de moyens pour décompter efficacement le nombre de k-mers apparaissant dans une lecture ou dans un métagénome. En pratique, un métagénome marin peut comporter plus de 10 milliards de k-mers différents et des techniques d'optimisation d'utilisation de la mémoire se révèlent cruciales.

Enfin, remarquons que les k-mers qui apparaissent peu souvent dans un métagénome résultent le plus souvent d'erreurs de séquençage et ne présentent que peu d'intérêt. On s'intéressent donc à des techniques qui permettent d'extraire efficacement les k-mers qui apparaissent au moins c fois dans un jeu de données.

1.2 Applications

La connaissance des fréquences d'occurrences des k-mers dans une séquence ou un ensemble de séquences ADN comporte de nombreuses applications en bio-informatique. Elle peut être utilisée comme indiqué précédemment comme outil de comparaison entre séquences ou métagénomes, comme signature d'un génome, comme première étape préliminaire à un alignement, pour estimer la taille du génome ou encore comme aide pour l'assemblage de novo.

1.3 Formalisation

Dans tout le sujet on considère l'alphabet $\Sigma = \{A, C, G, T\}$ et des mots de $\Sigma^\star$. Une *séquence* ou une *chaîne* est un mot sur Σ. Pour $u, v \in \Sigma^\star$ on dit que v est un *facteur*, une *sous-chaîne* ou encore *une sous-séquence*[1] de u s'il existe $x, y \in \Sigma^\star$ tels que $u = xvy$, autrement dit si v est constitué d'une suite de lettres *consécutives* de u. Dans tout le sujet, les mots *facteur*, *sous-chaîne* et *sous-séquence* sont considérés comme des synonymes. Un *k-mer* est une sous-séquence de longueur $k > 0$.

▷ **Question 1.** ✍ Combien de k-mers différents existe-t-il ? Dans une séquence de longueur n, majorer et minorer le nombre de k-mers différents qui peuvent apparaître dans cette séquence.

☑

Il y a 4^k k-mers différents envisageables. Si $n < k$ il y a 0 k-mers. Sinon il y en a au minimum un et au maximum $n - k + 1$.

ℹ

Cette question a permis au jury de s'assurer que la notion de k-mers était bien assimilée, et aiguiller les quelques candidats pour lesquels ce n'était pas le cas. Le jury n'attendait pas le traitement du cas particulier $n < k$, mais attendait une majoration correcte et précise. La majoration incorrecte $n - k$ a souvent été proposée. Le jury invitait alors le candidat à vérifier sa formule sur des exemples. Cette question est à rapprocher du cours sur l'algorithmique du texte.

▷ **Question 2.** ✍ 🧍 Donner les valeurs correspondantes pour $n = 100$ et $k = 30$. Que pensez-vous d'une approche qui pour stocker les k-mers utilise un tableau de booléens indexé par tous les k-mers existants ?

☑

On trouve $4^{30} \approx 10^{18}$ 30-mers différents. Même à raison d'un bit par 30-mer, cela dépasse largement la capacité de stockage des machines du concours, qui ont une capacité de l'ordre du Tera. Cela serait de surcroît complètement inefficace car la séquence de taille 100 ne contiendra qu'au plus 71 30-mers à stocker. Cette approche n'est pas envisageable.

ℹ

Cette question n'a pas posé de difficultés. Il était possible d'utiliser la machine à disposition pour obtenir une approximation sous forme d'une puissance de 10. De nombreux candidats ont signalé spontanément l'existence des tables de hachage pour résoudre le problème évoqué.

1.4 Organisation du sujet

Le sujet comporte quatre parties distinctes qui doivent être abordées dans l'ordre. En particulier, la dernière section 5 ne doit être abordée que si vous avez terminé les trois autres et le choix du langage y est libre.

1. Ce *n'est pas* la dénomination du programme où une sous-séquence désigne une sous-suite de lettres non nécessairement consécutives.

La section 2 s'intéresse à une base de données de séquences ADN et les requêtes sont à écrire dans le langage SQL. La section 3, à traiter en utilisant le langage OCaml, propose d'extraire tous les k-mers d'une séquence qui apparaissent au moins c fois. La section 4, à traiter en utilisant le langage C, propose une structure de données probabiliste pour réduire l'empreinte mémoire. Enfin, la section 5 vous propose d'implémenter une stratégie d'apprentissage automatique pour prédire la classe d'une séquence.

1.5 Description des données

Ce sujet est accompagné d'un ensemble de fichiers :

- une base de données SQLite `dna.db`, pour la partie 2 ;
- un fichier `kmers.ml` à compléter pour la partie 3 ;
- un fichier d'en-tête `kmers.h` et un fichier `kmers.c` qui est à compléter pour la partie 4 ;
- le répertoire `data` qui contient un ensemble de fichiers de données représentant des séquences d'ADN pour un chien, un chimpanzé et un humain. Les trois fichiers dont l'extension est `.seq` contiennent un ensemble de séquences ADN à raison d'une séquence par ligne et sont à utiliser dans les sections 3 et 4. Les trois fichiers dont l'extention est `.class` contiennent pour chaque ligne la classe respective de la séquence correspondante dans le fichier `.seq`, constituant un corpus étiqueté pour une approche par apprentissage statistique pour la section 5.

Dans les données proposées, un certain nombre de nucléotides (A, C, G ou T) n'ont simplement pas pu être identifiés et sont remplacés par la lettre N, par exemple on trouve la sous-séquence `CCAAGTCCTCAANNNNNNNNNNNNNNNNGTCCCGAGGCGC`. Dans tout le sujet, pour simplifier, on ignorera la présence de ces nucléotides inconnus, on ne cherchera pas à les supprimer ou à effectuer un traitement différent : on pourra considèrer que l'alphabet est constitué des cinq lettres $\{A, C, G, T, N\}$.

2 Base de données de séquences ADN

Une base de données `dna.db` compatible `SQLite` vous est proposée pour cet exercice. Pour interagir avec cette base de données on utilisera l'outil `sqlite3` en ligne de commande. Pour cela, il suffit de lancer la commande `sqlite3 dna.db` depuis le répertoire qui contient la base de données, qui sera alors ouverte. Une invite de commande `sqlite>` s'ouvre. La commande `.schema` permet d'afficher le schéma des tables de la base de données. La commande `.quit` permet de quitter. On peut entrer une requête SQL que l'on terminera par un `;` (point-virgule). Il est fortement recommandé de copier vos requêtes SQL une fois que celles-ci fonctionnent, par exemple dans un fichier annexe, pour pouvoir ensuite les retrouver rapidement et pouvoir les présenter à votre examinateur lors de son passage.

Il n'était pas demandé de recopier les requêtes sur le compte rendu (seulement le résultat de ces requêtes), mais de les sauvegarder quelque part pour pouvoir les retrouver rapidement lors du passage du jury, plutôt que de rechercher longuement dans l'historique. Le jury apprécie le fait que les candidats anticipent son passage.

Cette base de données comporte trois tables de même schéma :

- **id** : identifiant de la séquence ;
- **sequence** : une séquence ADN ;
- **class** : la classe associée à cette séquence.

En SQL, la fonction `LENGTH(s)` permet d'obtenir la longueur d'un attribut `s` de type textuel. La fonction `SUBSTR(s, deb, len)` permet d'extraire la sous-chaîne d'un attribut `s` de type textuel à partir de la position `deb` (incluse) et de longueur `len`. Attention, les positions commencent à 1 et pas à 0 contrairement à la fonction OCaml `String.sub` que nous utiliserons plus tard. Par exemple, la requête suivante permet d'obtenir chez le chien les trois premières lettres des séquences de longueurs inférieures ou égales à 250 :

```
SELECT SUBSTR(sequence, 1, 3) FROM dog WHERE LENGTH(sequence) <= 250;
```

Ce sujet introduit des fonctions qui ne figurent pas au programme, avec des explications et des exemples. La capacité à les utiliser par la suite fait partie des compétences attendues et évaluées.

▷ **Question 3.** ⌨ ✍ 🧍 Pour chacune des questions suivantes, il est demandé d'écrire une requête qui permet de répondre précisément à la question, d'indiquer le résultat sur le rendu et de présenter succinctement votre requête à votre examinateur ou examinatrice.

Le jury attendait une requête qui répond précisément à la question, pas uniquement une requête à partir de laquelle on peut trouver/déduire la réponse. Lorsque la requête n'était pas entièrement satisfaisante, le jury a invité le candidat à préciser sa requête. Tous les candidats ont bien compris ce jeu de dialogue.

(a) Combien de séquences différentes sont présentes pour le chimpanzé ?

☑
```
SELECT COUNT(DISTINCT sequence) FROM chimpanzee;
```

L'utilisation du mot clé `DISTINCT` a parfois posé problème.

(b) Quelles sont exactement les classes possibles pour l'ensemble des séquences des trois espèces ?

☑
```
SELECT class FROM human
UNION
SELECT class FROM chimpanzee
UNION
SELECT class FROM dog;
```

Le mot-clé `UNION`, qui figure pourtant au programme, n'est pas connu d'un grand nombre de candidats. Rappelons qu'il n'est pas du tout interdit — et même encouragé — d'utiliser la documentation du langage SQL qui est mise à disposition. Le jury a souvent dû le signaler explicitement aux nombreux candidats qui ne se souvenaient plus de la syntaxe, ce qui dans une épreuve pratique n'est pas du tout pénalisé si on arrive à utiliser la documentation à bon escient.

(c) Combien de séquences différentes présentes chez le chimpanzé sont également présentes chez l'humain ?

```sql
SELECT COUNT(DISTINCT c.sequence)
FROM human h JOIN chimpanzee c
ON h.sequence = c.sequence;
```

(d) Quelle est la longueur de la plus longue séquence chez le chien qui soit étiquetée par la classe 2 ?

```sql
SELECT MAX(LENGTH(sequence)) FROM dog WHERE class = 2;
```

(e) Donner, chez le chimpanzé, pour chaque classe, la longueur minimale d'une séquence étiquetée par cette classe.

```sql
SELECT class, MIN(LENGTH(sequence)) FROM chimpanzee GROUP BY class;
```

Question plutôt bien réussie.

(f) Quel est chez l'humain le 4-mer le plus fréquent en tout début de séquence (doublons compris) et quel est son nombre d'occurrences ? Si plusieurs 4-mers réalisent le maximum on pourra en choisir un arbitrairement ou les donner tous.

```sql
SELECT SUBSTR(sequence, 1, 4), COUNT(*) AS c
FROM human
GROUP BY SUBSTR(sequence, 1, 4)
ORDER BY COUNT(*)
DESC LIMIT 1;
```

Question plus difficile que beaucoup de candidats n'ont pas traitée. La grande majorité de ceux qui l'ont traitée l'ont faite correctement. On pouvait utiliser `MAX` ou `ORDER BY LIMIT 1`.

3 Décompte de tous les k-mers (en OCaml)

Dans cette partie et la suivante, on cherche à trouver tous les k-mers d'une séquence ou d'un ensemble de séquences avec leur fréquence d'apparition. Comme les k-mers peu fréquents résultent le plus souvent d'erreurs de séquençage et sont en général peu porteurs d'information statistique on s'intéresse souvent plutôt à dénombrer tous les k-mers qui apparaissent au moins un certain nombre de fois.

On cherche à compléter le programme `kmers.ml` qui vous est proposé pour que celui-ci détermine le nombre de k-mers qui apparaissent au moins un certain nombre de fois dans un fichier de séquences. Pour cela on va construire une table de hachage dont les clés représentent les k-mers et les valeurs associées leur nombre d'occurrences.

On pourra consulter la documentation du module `Hashtbl` et utiliser librement toutes les fonctions qui s'y trouvent.

▷ **Question 4.** ⌨ Implémenter la fonction

```
add_one : ('a, int) Hashtbl.t -> 'a -> unit
```

qui incrémente de un la valeur associée à une clé dans une table de hachage. *Remarque : une clé non présente est présente* 0 *fois.*

On rappelle que la fonction `open_in : string -> in_channel` permet d'ouvrir un fichier, que la fonction `close_in : in_channel -> unit` permet de fermer un fichier, que la fonction `input_line : in_channel -> string` permet de lire une nouvelle ligne[2] d'un fichier s'il en reste et lève l'exception `End_of_file` sinon. On rappelle que la syntaxe pour rattraper une telle exception est `try expr1 with End_of_file -> expr2`.

▷ **Question 5.** ⌨ Implémenter la fonction

```
file_iter : (string -> unit) -> string -> unit
```

telle que `file_iter f file_name` applique la fonction `f` sur chaque ligne du fichier de nom `file_name`.

▷ **Question 6.** 🕴 Consulter la documentation du module `String` et expliquer très brièvement ce que réalise la fonction `String.sub` et quels sont ses arguments valides.

▷ **Question 7.** ⌨ Implémenter la fonction

```
build_seq : int -> (string, int) Hashtbl.t -> string -> unit
```

qui met à jour le décompte des occurrences de k-mers présents dans une table de hachage avec toutes les nouvelles occurrences des k-mers présents dans une chaîne de caractères.

L'appel `build file_name k` permet de construire une table de hachage qui répertorie pour chaque k-mer son nombre d'occurrences dans un fichier de séquences.

▷ **Question 8.** ⌨ Implémenter la fonction `delete_all` qui supprime dans une table de hachage toutes les liaisons dont la *valeur* est la même que celle passée en paramètre. Le type de cette fonction est donc `('a, 'b) Hashtbl.t -> 'b -> unit`. On pourra utiliser la fonction `filter_map_inplace` du module `Hashtbl` dont on lira très attentivement la documentation.

2. Le caractère `'\n'` est consommé mais n'est pas présent dans la chaîne de caractère renvoyée.

Le programme principal est donné dans la fonction `main`. La compréhension de la syntaxe de la chaîne de format utilisée pour la `Printf.printf` n'est pas attendue.

▷ **Question 9.** Expliquer ce que réalise ce programme. Exécuter ce programme sur une entrée de votre choix pour illustrer son fonctionnement.

Il suffit d'écrire en ligne de commande `./exec nom_de_fichier_avec_sequences k` pour obtenir :

- Le nombre de k-mers différents observés ;
- Pour i entre 1 et 9 inclus, le nombre de k-mers différents observés strictement plus de i fois et le pourcentage de k-mers observés que cela représente.

La fonction `file_iter` va lire chaque ligne du fichier (correspondant aux différentes séquences) et appliquer une fonction qui consiste à faire glisser une fenêtre de taille k sur cette séquence pour récupérer les différents k-mers et les ajouter à la table de hachage. Contrairement à un tableau de booléens, on ne va stocker que les k-mers rencontrés. Grâce à la fonction `Hashtbl.filter_map_inplace`, on peut simplement écarter les k-mers dont la valeur associée est i, c'est-à-dire les k-mers qui ont été observés exactement i fois. La taille d'une table de hachage peut être déterminée avec `Hashtbl.length` ce qui permet de savoir combien de k-mers il reste après avoir enlevé tous les k-mers apparaissant au plus i fois.

Encore une fois le recul, la clarté et la concision des explications ont été valorisés. Le jury demande des précisions si nécessaire en particulier sur les fonction utilisées, qui pour certaines ne sont pas au programme et dont la documentation doit donc avoir au moins rapidement été consultée.

▷ **Question 10.** Combien de 10-mers différents sont présents chez le chien ? Combien de 20-mers différents apparaissent strictement plus de 5 fois chez le chimpanzé ? Quel est le pourcentage de 30-mers différents qui apparaissent au moins 10 fois chez l'humain ?

On trouve 563539 10-mers pour le chien, 71945 20-mers apparaissant strictement plus de 5 fois chez le chimpanzé et 5.49% des 30-mers de l'humain apparaissant au moins 10 fois.

On remarque que le nombre de k-mers qui apparaissent au moins c fois peut ne représenter qu'une petite fraction du nombre total de k-mers différent. L'approche proposée ici nécessite de stocker l'ensemble de tous les k-mers différents avant d'en déduire ceux qui apparaissent au moins c fois. Si le nombre de k-mers différents est trop important, cette approche peut ne pas être utilisable en pratique.

4 Filtres de Bloom pour le décompte des k-mers (en C)

Dans cette partie, on propose une structure de donnée probabiliste, les *filtres de Bloom* pour améliorer l'empreinte mémoire de l'approche précédente.

4.1 Préliminaires

Un fichier `kmers.c` vous est proposé et est à compléter. Le fichier d'en-tête `kmers.h` contient les définitions et les prototypes des fonctions à implémenter. On peut compiler le fichier sans aucune option à l'aide de la ligne de compilation `gcc kmers.c -o kmers`. On peut également activer les avertissements et une vérification de salubrité de l'utilisation mémoire en utilisant la ligne de compilation :

```
gcc -g -Wall -Wextra -fsanitize=address kmers.c -o kmers
```

Le jury pourra vous demander d'utiliser cette deuxième version pour vérifier que votre implémentation est correcte.

▷ **Question 11.** 🧍 Rappeler à quoi correspond le type `uint64_t` en C.

☑
Il s'agit d'un entier non signé sur 64 bits.

Le fichier d'en-tête définit un alias de type `typedef uint64_t u64;` pour simplifier la suite. On peut afficher un élément de ce type avec la chaîne de format `%llu`.

ℹ
Ceci n'est pas toujours vrai mais l'était sur les machines proposées.

▷ **Question 12.** ⌨ Implémenter une fonction `u64 exp_mod(u64 x, u64 n, u64 p);` telle que `exp_mod(x, n, p)` calcule $x^n \mod p$ par exponentiation rapide. *Vérifier que* $2^{1000} \mod 131 = 39$ *et non* 0 *auquel cas vous avez vraisemblablement un débordement de capacité des entiers.*

ℹ
Comme le jury l'avait anticipé, cette question s'est avérée très difficile pour les candidats. Nous invitons les futurs candidats à savoir résoudre ce problème classique. Outre les problèmes de dépassements d'entiers, plusieurs candidats n'étaient pas à l'aise sur l'exponentiation rapide.

Le fichier `kmers.c` définit un tableau `P` de nombres premiers et un tableau `B` de bases, une constante `M` ainsi qu'une chaîne de caractères `ex_seq` correspondant à la première séquence ADN des données sur le chien.

▷ **Question 13.** ✍ Que vaut `exp_mod(B[0], 42, P[0])` ?

☑
On a $257^{42} \mod 4242424243 = 3047293040$.

4.2 Implémentation d'une famille de table de hachage

Dans cette partie on se propose d'implémenter une famille de fonctions de hachage. Une fonction de hachage est une fonction $h : \Sigma^\star \to \mathbb{N}$. Dans ce sujet, on considère une famille

de fonctions de hachage $(h_{b,p})_{(b,p)}$ paramétrées par une base $b > 2$ et un nombre premier p. Pour une base b et un nombre premier p on définit la fonction de hachage $h_{b,p}$ par

$$h_{b,p} : u \in \Sigma^k \mapsto \sum_{i=0}^{k-1} b^{k-i-1} u_i \mod p$$

On peut transformer un caractère `c` de type `char` en entier `uint64_t` en transtypant explicitement : `(u64)c`.

▷ **Question 14.** ⌨ Implémenter une fonction `u64 hash(char *s, int k, u64 b, u64 p)` telle que `hash(s, k, b, p)` calcule $h_{b,p}(s_0 s_1 \dots s_{k-1})$. On suppose que `s` est un pointeur valide vers une chaîne de caractères de longueur au moins k. *Une approche efficace n'utilise pas la fonction* `exp_mod`. *On veillera à nouveau à ne pas provoquer de dépassement de capacité. Vérifier que* `hash(ex_seq, 100, 2, 131) = 73`.

▷ **Question 15.** ✍ Que vaut `hash(ex_seq, 10, B[0], P[0])` ?

☑

$h_{257,4242424243}(\text{ex\_seq}) = 2964998361$.

▷ **Question 16.** 🧍 Soit $x, y \in \Sigma$ et $u \in \Sigma^{k-1}$. Supposons que l'on connaisse $h_{b,p}(xu)$ expliquer comment calculer efficacement $h_{b,p}(uy)$ et donner la complexité de ce calcul. Expliquer l'avantage du choix la famille de fonction de hachage proposée dans l'optique de calculer succesivement les valeurs de hachages des k-mers d'une séquence. Citer une autre application de ce genre de fonctions de hachage et décrire brièvement l'algorithme associé.

☑

Notons $u = u_1 u_2 \dots u_{k-1}$. On a

$$h_{b,p}(xu) = xb^{k-1} + \sum_{i=1}^{k-1} b^{k-i-1} u_i \mod p$$

et

$$h_{b,p}(uy) = \sum_{i=1}^{k-1} b^{k-i} u_i + y \mod p$$

Donc

$$h_{b,p}(u, y) = b(h_{b,p}(xu) - xb^{k-1}) + y \mod p$$

En précalculant b^{k-1} une fois pour toutes, cette opération se fait en temps constant. On peut donc recalculer la valeur de la fonction de hachage pour chaque décalage en temps constant. Cette même astuce est utilisée dans l'algorithme de Rabin-Karp pour rechercher un mot dans un texte. Comme dans l'algorithme naïf on étudie toutes les positions où peut se placer le mot, mais avant de comparer lettre à lettre, on élimine toutes les positions pour lesquelles la fonction de hachage ne donne pas le même résultat. Calculer la valeur de la fonction de hachage se fait par mise à jour de sa valeur pour la position précédente en temps constant.

Comme toutes les questions « de cours », cette question valorise particulièrement les candidats qui réussissent à démontrer un bon recul et une réelle compréhension des concepts étudiés pendant la formation.

On pourrait utiliser cette manière de calculer successivement les valeurs de la fonction de hachage pour les k-mers d'une séquence. Dans un souci de simplification, nous réutiliserons cependant la fonction `hash` à chaque étape.

4.3 Implémentation d'une table de hachage

Dans cette partie on se propose d'implémenter une table de hachage pour stocker les k-mers. Une table de hachage est un tableau de taille $m \in \mathbb{N}^*$ contenant des listes d'association, ainsi qu'une fonction de hachage. Une liste d'association est une liste chaînée dont les éléments sont des couples de clés et de valeurs. Dans une table de hachage de longueur m et de fonction de hachage h, un couple de clé-valeur (c, v) est placé dans la liste d'association de la case $h(c) \mod m$ du tableau. La position dans cette liste n'a pas d'importance. Une clé ne peut être présente qu'une seule fois dans une table.

On représente une liste d'association en C par la structure suivante :

```
struct cell {
  char* kmer;
  int count;
  struct cell* next;
};
typedef struct cell cell;
```

Un cellule contient une clé qui est un k-mer associé à une valeur qui est son nombre d'occurrences. Une liste d'association vide est représentée par le pointeur `NULL`.

Une table de hachage est représentée par la structure suivante :

```
struct hashtbl {
  cell** table;
  int m;
  u64 b;
  u64 p;
};
typedef struct hashtbl hashtbl;
```

Cette structure comporte un pointeur sur un tableau de taille m ainsi que les paramètres b et p de la fonction de hachage $h_{b,p}$.

▷ **Question 17.** ⌨ Écrire une fonction `hashtbl* hashtbl_create(int m, u64 b, u64 p)` qui permet d'allouer et d'initialiser une table de hachage à partir de ses paramètres.

Trois fonctions qui permettent respectivement d'incrémenter de un la valeur associée à une clé ou l'insérer si elle n'était pas présente, d'afficher le contenu et de compter le nombre d'éléments présents dans une table de hachage vous sont proposées dans le fichier `kmers.c`. Ces fonctions peuvent vous être utiles pour vos tests.

▷ **Question 18.** 🧍 Détailler le fonctionnement de la fonction `hashtbl_add_one`. On pourra consulter la documentation des fonctions utilisées.

La fonction `add_one` commence par trouver l'indice de l'alvéole à considérer en calculant le haché du k-mer modulo m. Ensuite, la fonction parcourt la liste d'association s'y trouvant en comparant la clé avec le k-mer, à l'aide de la fonction `strncmp` qui vaut 0 si et seulement si les deux chaînes de caractères qu'elle prend en argument sont identiques, cette comparaison ne prenant en compte que les k premiers caractères de ces chaînes. Si une cellule contient la bonne clé, alors la fonction incrémente la valeur associée puis s'arrête. Sinon une cellule est allouée et ajoutée en début de liste d'association avec ce nouveau k-mer comme clé.

▷ **Question 19.** ⌨ Écrire une fonction `void hashtbl_delete(hashtbl* h)` qui permet de libérer toute la mémoire allouée pour une table de hachage. Expliquer pourquoi cette fonction libère bien tout ce qui doit l'être.

On libère chaque maillon de chaque liste d'associations, puis on libère la table, puis on libère la structure de la table de hachage. On libère exactement ce qui a été alloué, à chaque `malloc` étant associé un unique `free`.

La question à détailler à l'oral permet de s'assurer que le candidat n'a rien oublié. Dans le cas contraire, le jury le met sur la bonne voie par des questions.

▷ **Question 20.** ✍ Combien il y a-t-il de 3-mers différents dans la séquence `ex_seq` ? On pourra utiliser une table de hachage de paramètres `M`, `B[0]` et `P[0]`.

On trouve 49 3-mers différents dans la séquence `ex_seq`.

La fonction `hashtbl* build(char* file_name, int k)` permet de compter le nombre de k-mers dans un fichier.

▷ **Question 21.** Vérifier que vous retrouvez le même nombre de 30-mers différents pour l'humain avec le programme C et avec le programme OCaml.

On retrouve bien 1470161 30-mers chez l'humain.

4.4 Filtres de Bloom

Cette partie, qui constituait pourtant le cœur du sujet, n'a été abordée que par quelques candidats, le sujet étant déjà relativement long jusque-là.

On pourrait, comme précédement, filtrer les k-mers qui apparaissent moins d'un certain nombre de fois, mais on va chercher, dans cette dernière partie, à éliminer directement ces éléments sans avoir à les représenter explicitement en mémoire, ce qui permet d'améliorer

la complexité spatiale de cette méthode et de pouvoir l'appliquer lorsque la capacité nécessaire pour stocker temporairement *tous* les k-mers excéderait celle de la mémoire principale.

On s'intéresse à une structure de données probabiliste, appelée *filtre de Bloom*, qui permet de représenter un ensemble de manière compacte, au prix d'un faible ratio de faux positifs pour le test d'appartenance.

Un filtre de Bloom est constitué d'un tableau d'entiers de taille $m > 0$ ainsi que d'une collection de $r > 0$ fonctions de hachage $(h_i)_{0 \leqslant i \leqslant r-1}$ à valeurs dans $[\![0; m-1]\!]$. Au départ, toutes les cases du tableau sont initialisées à 0. Pour ajouter un élément x au filtre, on affecte à toutes les cases $h_0(x), h_1(x), \ldots, h_{r-1}(x)$ la valeur 1. Pour tester si un élément x est présent, on vérifie si toutes les cases $h_0(x), h_1(x), \ldots, h_{r-1}(x)$ contiennent un 1.

▷ **Question 22.** 🕴 On considère un filtre avec $m = 10$ et $r = 3$. On insère un élément x tel que $h_0(x) = 1, h_1(x) = 3, h_2(x) = 4$ puis un élément y tel que $h_0(y) = 4, h_1(y) = 6, h_2(y) = 8$. L'élément z tel que $h_0(z) = 0, h_1(z) = 1, h_2(z) = 9$ est-il présent dans le filtre ? Et l'élément w tel que $h_0(w) = 3, h_1(w) = 4, h_2(w) = 6$?

☑

On obtient le tableau :

0	1	2	3	4	5	6	7	8	9
0	1	0	1	1	0	1	0	1	0

L'élément z n'est pas présent puisque la case $h_0(z) = 0$ ne contient pas un 1. En revanche, l'élément w, bien qu'il n'ait pas été ajouté, est considéré comme étant présent.

▷ **Question 23.** 🕴 Justifier que si un élément a été ajouté dans le filtre, alors tout test d'appartenance de cet élément sera toujours positif. Si un test d'appartenance d'un élément dans le filtre est positif, peut-on en déduire de manière certaine que l'élément a été ajouté au filtre ?

☑

Si un élément a été ajouté au filtre, on a affecté un 1 dans toutes les cases correspondantes. Une case à 1 ne peut jamais redevenir une case à 0, ainsi toutes ces cases restent définitivement à 1. Tout test ultérieur d'appartenance de cet élément sera donc positif. Inversement, on vient de voir à la question précédente qu'il était possible d'avoir un test d'appartenance positif sans que l'élément n'appartienne effectivement à la structure (et cela même si son image par les fonctions de hachage est différente de celle de tous les éléments déjà ajoutés).

On suppose que les r fonctions de hachage sont indépendantes et vérifient toutes l'hypothèse d'un hachage uniforme, c'est-à-dire que la probabilité qu'un élément soit haché vers une case est égale à $\frac{1}{m}$.

▷ **Question 24.** 🕴 Pour une case donnée, quelle est la probabilité que cette case soit laissée à 0 par une des r fonctions de hachage après insertion d'un élément ? Et après insertion de n éléments (en supposant que toutes ces insertions sont indépendantes) ? Montrer que la probabilité pour une case, après insertion de n éléments, d'avoir la valeur 1 est environ $1 - e^{\frac{-rn}{m}}$, en supposant m assez grand.

Par hachage uniforme, la probabilité de mettre une case à 1 est de $\frac{1}{m}$ donc celle de laisser une case à 0 est de $1-\frac{1}{m}$. Comme on suppose que les r fonctions de hachage sont indépendantes, tout comme les n insertions ensuite, on a une probabilité de $(1-\frac{1}{m})^{rn} \approx e^{\frac{-rn}{m}}$ de laisser une case à 0. La probabilité d'avoir la valeur 1 est donc environ de $1-e^{\frac{-rn}{m}}$.

En supposant en première approximation que la probabilité qu'une case vaille 1 est indépendante des autres cases, on a donc lors du test d'appartenance une probabilité de faux positif de l'ordre de $(1-e^{\frac{-rn}{m}})^r$. On peut montrer que sous certaines hypothèses cette approximation est réaliste et qu'un bon choix pour r est $r = \frac{m}{n}\ln 2$.

▷ **Question 25.** ✍ Avec un choix de $m = 8 \times n$ et $r = 5$, donc une complexité spatiale d'un octet par k−*mer*, quelle est l'estimation du taux de faux positifs ?

L'application numérique donne un taux de faux positifs de l'ordre de 2.16%, ce qui semble raisonnable.

Dans ce sujet on ne cherchera pas à choisir m et r de manière optimale. On prendra $r = 3$ et $m = 10\,000\,000$ (qui est défini par la constante `M`).

▷ **Question 26.** 🕴 Expliquer comment on peut utiliser un filtre de Bloom et une table de hachage pour extraire d'une suite de séquences tous les k-mers qui apparaissent au moins deux fois (donc en filtrant ceux qui apparaîssent une seule fois), en utilisant une complexité mémoire de l'ordre de m plus le nombre de k-mers qui apparaîssent au moins deux fois (et donc avec un gain de l'ordre de 50 % par rapport à l'approche qui consiste à insérer *tous* les k-mers dans une table et de réaliser un filtrage *a posteriori*). On pourra réaliser deux passes sur les séquences. Que se passe-t-il si on effectue une seule passe, c'est-à-dire si on ne s'autorise à parcourir qu'une seule fois les données ?

On parcourt les k-mers et on regarde pour chacun s'il est déjà présent ou non dans le filtre de Bloom. S'il est déjà présent, on l'ajoute à la table de hachage associée à la valeur 2 ; sinon, on l'ajoute au filtre de Bloom. Ainsi, après cette première passe, la table de hachage contient tous les k-mers qui apparaissent au moins deux fois, ainsi que quelques éventuels faux positifs. On peut alors réaliser une seconde passe pour éliminer les faux positifs. On reparcourt tous les k-mers en décrémentant pour chaque occurrence le compteur associé à un k-mer si celui-ci est présent dans la table de hachage. Enfin, on élimine les k-mers dont le compteur est resté strictement positif dans la table de hachage, qui correspondent aux faux positifs. La complexité mémoire est bien celle attendue, sous l'hypothèse que le nombre de faux positifs soit négligeable.

À ce stade du sujet, cette question consistait davantage en une discussion avec le candidat dont on n'attendait pas une proposition complète et parfaitement rodée avant le passage du jury.

On se propose d'étendre la structure de données de filtre de Bloom pour représenter, de manière approximative, le nombre de fois que l'on a rencontré un élément. Pour cela, plutôt que de mettre à 1 les valeurs des cases données par les r fonctions de hachage lors de l'ajout, il suffit d'incrémenter les valeurs de ces cases.

▷ **Question 27.** 🕴 Justifier que si un élément x a été ajouté c fois à un filtre, alors toutes les cases $h_0(x), h_1(x), \ldots, h_{r-1}(x)$ ont une valeur supérieure ou égale à c. Quel test peut-on effectuer pour essayer de savoir si un élément a été vu au moins c fois ? Peut-on affirmer avec certitude qu'un élément n'a pas été vu plus de c fois ? Peut-on affirmer avec certitude qu'un élément a été vu au moins c fois ?

☑

Remarquons que la valeur d'une case du filtre ne peut qu'augmenter et jamais diminuer. Si un élément x a été ajouté c fois, alors toutes les cases correspondantes ont été incrémentées au moins c fois et ont donc une valeur supérieure ou égale à c. Pour tester si un élément a été vu au moins c fois, il suffit de vérifier que dans toutes les cases correspondantes on trouve une valeur supérieure ou égale à c. On vient d'indiquer que si un élément avait été vu au moins c fois, alors ce test est nécessairement positif. On peut donc affirmer avec certitude que si ce test échoue alors c'est que l'élément n'a pas été vu c fois ou plus. En revanche, si ce test réussit, on ne peut pas affirmer que l'élément a effectivemement été observé au moins c fois puisque les incrémentations des cases correspondantes peuvent avoir été réalisées lors de l'ajout d'autres éléments.

On se propose d'implémenter un filtre de Bloom de largeur $m > 0$ en utiliser la famille de fonctions de hachage définie précédemment. Pour $(b_i)_{0\leqslant i\leqslant r-1}$ une famille de bases et $(p_i)_{0\leqslant i\leqslant r-1}$ une famille de nombres premiers, on définit la fonction de hachage $\tilde{h}_i$ par $\tilde{h}_i : u \mapsto h_{b_i,p_i}(u) \mod m$. Attention à ne pas oublier le modulo m dans votre implémentation. On prendra $r = 3$ et on utilisera les familles de bases et de nombres premiers définies par les tableaux `B` et `P` de `kmers.c`.

On représente un filtre par la structure suivante en C, où le champ `count` représente le tableau de taille m.

```c
struct filter {
  int m;
  int r;
  int* count;
};
typedef struct filter filter;
```

▷ **Question 28.** ⌨ Implémenter les quatre fonctions nécessaires à son utilisation : `filter_create` qui créé un filtre de largeur m avec r fonctions de hachage ; `filter_add_one` qui ajoute une occurrence d'un k-mers au filtre ; `filter_mem` qui vérifie si un k-mers a déjà été rencontré au moins c fois ; et `filter_delete` qui permet de libérer un filtre.

▷ **Question 29.** ⌨ Implémenter une fonction `fast_build`, sur le modèle de la fonction `build`, qui renvoie une table de hachage contenant tous les k-mers des séquences d'un fichier qui apparaissent au moins c fois ainsi qu'un nombre raisonnable de faux positifs, c'est-à-dire de k-mers qui n'apparaissent pas c fois. On utiliser un filtre de Bloom pour ne pas avoir à stocker explicitement en mémoire l'ensemble de *tous* les k-mers mais seulement ceux de la table renvoyée.

▷ **Question 30.** ✍ Pour les séquences du chien, combien de 20-mers apparaissant au moins 4 fois obtenez-vous avec cette méthode approchée ? Quel est le nombre exact de 20-mers qui apparaissent au moins 4 fois, que l'on peut obtenir avec le programme OCaml de la section 4.2 ? Cette stratégie vous semble-t-elle efficace ?

☑

On trouve 2371 20-mers apparaissant au moins 4 fois chez le chien contre 1979 pour la valeur exacte avec le programme OCaml. La marge d'erreur est acceptable : il y a à peine 400 faux positifs. On évite ainsi de stocker les 1242670 k-mers, mais on utilise tout de même une table de taille m.

5 Prédiction de la classe d'une séquence

ℹ

Aucun candidat n'a eu le temps d'aborder cette partie, qui, davantage guidée, pourrait même constituer un sujet complet. Traiter cette partie n'était pas une attente du jury et n'était nullement nécessaire pour atteindre la note maximale. Cette ouverture, plus libre, aurait éventuellement permis à des candidats extrêmement rapides de continuer à s'exprimer. Nous proposons aux lecteurs curieux le soin d'étudier et de tester différentes approches.

Cette partie ne doit être abordée que si vous avez raisonnablement avancé dans les autres parties et impérativement après accord de votre examinateur/examinatrice.

Dans cette partie, le choix du langage n'est pas imposé. Les séquences proposées dans les données disponibles pour ce sujet sont *étiquetées*. Les fichiers `.class` correspondent, ligne par ligne, aux étiquettes des séquences des fichiers `.seq`. L'étiquette est une classe à prédire à partir de la séquence. Comme indiqué dans l'introduction, on peut utiliser les k-mers pour inférer une similarité entre deux séquences et utiliser cette similarité comme « distance » dans une méthode d'apprentissage statistique.

▷ **Question 31.** 🕴 Proposer une approche d'apprentissage automatique pour résoudre le problème proposé, c'est-à-dire d'être ensuite capable, pour une séquence appartenant à une certaine espèce, de prédire sa classe. Décrire la méthode que vous pensez utiliser. Détailler la manière dont vous pensez utiliser les données pour construire et évaluer la qualité de votre fonction de prédiction.

▷ **Question 32.** ⌨ Implémenter votre approche dans le langage de votre choix.

▷ **Question 33.** ✍ Détailler le protocole d'évaluation mis en place et les résultats que vous obtenez.