TP Bloom filters

Description générale

Les tables de hachage sont utiles dans bien des cas. Néanmoins lorsque le nombre de clés à stocker est très grand (par exemple des millions ou milliards de *k*-mers) alors la taille de la table est telle qu'il est nécessaire de la stocker sur le disque. L'inconvénient à cela est que les temps d'accès deviennent en général bien plus importants que lorsque la table est stockée en RAM. Si on doit tester de nombreuses fois l'existence d'un élément dans la table cela peut devenir rédhibitoire, malgré la bonne complexité de la requête.

De plus, les tables de hachage ne sont pas des objets optimaux si on ne veut pas associer des clefs et des valeurs, mais simplement tester l'existence d'un élément dans un ensemble. Pour cette dernière opération, Python par exemple propose des objets set eux aussi basés sur le hachage.

Dans ce TP on va réaliser une structure de données, le **filtre de Bloom**, qui représente un ensemble de mots (chaînes de caractères) et répond à la question “un mot X est il présent dans l'ensemble indexé ?”. Cette structure est *probabiliste* : elle a des faux positifs à la requête. Ainsi, par exemple, si à la question “le *k*-mer ATGGTAC est-il présent dans le filtre ?”, si le filtre répond non on peut faire 100% confiance à la réponse, mais si le filtre répond oui on s'attend à un certain taux de positifs (donc peut-être répondra-t-il oui alors que ATGGTAC n'est pas présent en réalité).

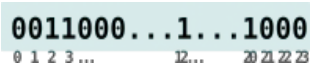
Un filtre de Bloom est donc une structure :

- dans un premier temps, qui se remplit avec les éléments d'un ensemble *S*
- dans un second temps, qui s'interroge avec des éléments et répond oui/non à leur présence dans *S* Dans notre cas, on va peupler un filtre de Bloom avec un ensemble de mots sur un alphabet plus grand que les applications générales en bioinformatique.

L'idée est la suivante.

- On crée une table *B* de booléens (en Python, cela peut-être une liste de 0/1 par exemple).
- On dispose pour les clés que nous avons à ranger d'un certain nombre *m* de fonctions de hachage *hi, 0 <= i <= m−1*.
- Pour chaque clé *c* (un mot) à ajouter à *B*, on va mettre à vrai les m cases *B[hi(c)]*.

Le principe étant que la probabilité que deux clés différentes aient les mêmes *m* valeurs pour leurs fonctions de hachage est faible. Par exemple, supposons que nous souhaitions entrer la clé “ATGGGCC” dans la table *B* de taille 24 initialement vide (toutes les cases sont à faux), que nous ayons quatre fonctions de hachage et que *h0*(“ATGGGCC”)=2, *h1*(“mot”)=12, *h2*(“TTTTTT”)=3, *h3*(“structure”)=20. L'état de la table B après l'insertion sera :



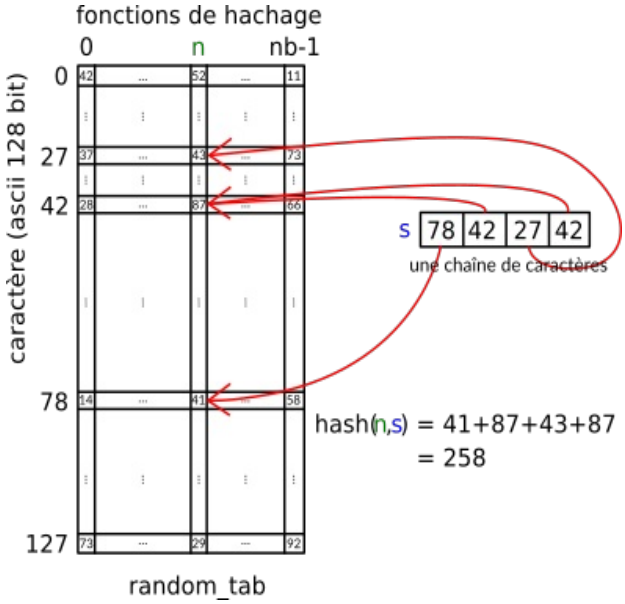
Pour savoir si une clé est présente, on s'assurera que les *m* cases de la table *B* correspondant aux valeurs des *m* fonctions de hachage sont positionnées à vrai.

Le but du TP est d'implémenter un filtre de Bloom et de tester l'efficacité de celui-ci en faisant varier sa taille ainsi que le nombre de fonctions de hachage. Pour mesurer l'efficacité on estimera le taux de faux positifs, c'est-à-dire le ratio entre le nombre de fois où le filtre se trompe et le nombre de clés interrogées.

Vous trouverez un fichier bloomfilter.py où vous implanterez les primitives de manipulation d'un filtre de Bloom et où vous ajouterez des doctests. Un programme gnuplot tp-bloom.plt est également fourni pour tracer les courbes.

Fonctions de hachage d'une chaîne de caractères

Nous allons supposer que nous manipulons des clés qui sont des chaînes de caractères écrites avec les 128 premiers caractères de la table ASCII. Afin d'obtenir différentes fonctions de hachage, nous allons attribuer un code différent aux caractères pour chaque fonction de hachage. Le hash d'un caractère sera obtenu de manière aléatoire. À cette fin on a créé un tableau random_tab contenant tous une valeur par caractère et par fonction de hachage (voir la figure ci-dessous).



La taille du tableau random_tab est égale au nombre de fonctions de hachage (horizontalement) multiplié par le nombre de caractères (verticalement). La valeur pour caractère (par exemple le caractère 78 de la table ASCII) pour une fonction de hachage *n* donnée est obtenu en lisant la colonne *n* correspondante (à la ligne 78). Dans cet exemple, la valeur associée au caractère 78 pour la fonction de hachage *n* est 41. La figure montre un exemple pour une chaine de caractères *s* composée des caractères dont les codes ASCII sont 78, 42, 27 et 42.

Classe BloomFilter

Constructeur

Ecrire un constructeur pour cette classe. Un filtre de Bloom a besoin comme paramètres :

- d'un nombre de fonctions de hachages.
- d'une taille pour la table de booléens.
- d'initialisar la table de booléens (avec toutes les positions à *faux* au départ)
- du tableau pour calculer les hash random_tab qui sera initialisé avec la méthode init_random_tab() donnée.

Méthodes du filtre de Bloom

Coder la méthode hash_of_string de la classe BloomFilter, qui prend en argument une chaine de caractère (un mot) et le numéro *m* de fonction de hachage (entre 0 et *m-1*). Elle retourne la valeur du hash pour la chaine de caractère par la _m_ième fonction de hachage en utilisant les valeurs dans random_tab.

Coder la méthode add qui prend un élément à insérer en argument et l'insère dans le filtre de Bloom. Attention, l'adresse retournée par hash_of_string n'est pas forcément dans les bornes de la table.

Puis coder la méthode contains qui prend un élément dont on veut tester la présence dans le filtre, et retourne un booléen en réponse.

Analyse des faux-positifs

Construire un ensemble de test

On va insérer 200 mots dans notre filtre et en tester 200 autres absents en les requêtants. Pour construire l'ensemble de mots à mettre dans le filtre, créer un objet *set* et le remplir de 200 mots distincts avec la fonction random_word.

Tests

Maintenant que vous disposez d'un filtre de Bloom fonctionnel, vous allez pouvoir tester l'influence du nombre de fonctions de hachage et de la taille du filtre sur le nombre de faux positifs.

Pour réaliser ces tests nous allons faire varier :

- le nombre de fonctions de hachage de 1 à 8

- le log de la taille du filtre de 10 à 20

Pour chaque jeu de test, nous allons construire un filtre de Bloom et y insérer 200 mots. Une fois ces élements insérés, on testera la présence de 200 autres.

Le pseudo-code est donc (afin de s'assurer de la compatibilité avec les fichiers distribués pour ce TP, vous respecterez l'ordre dans lequel les boucles sont faites et l'ordre dans lequel les impressions sont effectuées) :

creer l'ensemble E des mots à inserer + les faux positifs dans un set avec random_word

```
pour n = 1 à 8 faire
   pour t = 10 à 20 faire
      creer un filtre de bloom BF de taille 2^t et à n fonctions de hachage
      inserer 200 mots générés aléatoirement (ensemble E dans un set) dans BF
      pour f = 1 à 200
         générer un mot U aléatoire avec random_word
         si U n'appartient pas à E alors (tester en utilisant le set)
            augmenter le compteur de mots testés
            si U appartient à BF alors
               augmenter le compteur de faux positifs
            fin si
         fin si
      fin pour
      imprimer dans cet ordre:
         le log de la taille du filtre, le nombre de fonctions, le nombre de mots testes,
         le nombre de faux positifs, le taux de faux positifs
   fin pour
   imprimer deux lignes vides
fin pour
```

Vous devriez avoir un résultat similaire à :

```
10 1 16383 10537 0.643167
11 1 16385 6584 0.401831
12 1 16385 3663 0.223558
13 1 16385 1866 0.113885
14 1 16385 1035 0.063168
15 1 16385 529 0.032286
16 1 16385 267 0.016295
17 1 16385 174 0.010619
18 1 16385 164 0.010009
19 1 16385 164 0.010009
20 1 16385 159 0.009704
10 2 16385 12562 0.766677
11 2 16385 6634 0.404883
...
```

Enregistrer vos résultats dans un fichier nommé res.txt.

Utiliser le programme Gnuplot tp-bloom.plt pour tracer la courbe des résultats obtenus. : gnuplot < tp-bloom.plt

Expliquer les résultats que vous obtenez. En particulier, comment bien choisir le nombre de fonctions de hachage en fonction de la taille du filtre de Bloom ?