Filtre de Bloom | Fonctions de hachage | TP 3

Présenté par: Omar ALLOUCH & Charaf Eddine EL KIHAL

Implémentation / Code

Dans un premier temps, on va présenter les étapes d'implémentation du filtre comme vu dans le cours.

- **Initialisation du filtre :**

  Ca se traduit par un tableau de bits, de taille *m* est initialisé à 0.

```
bloom_filter = [0] * size
```

- **Insertion des éléments du premier fichier :**

  Ca se traduit par la mise à 1 des cellules d'indice $h_1(x), h_2(x), ...h_k(x)$.

```
for line in f1:  # Single word per line
    word = line.strip()
    for k in range(num_hash):
        index = functions[k](word, size)
        bloom_filter[index] = 1
```

- **Recherche d'un élément z : (calcul d'intersection)**

  On calcule $h_1(z), h_2(z), ...h_k(z)$ (*z* étant l'élément à chercher), si l'une des cellules $T[h_i(z)]$ est 0, on déduit $z \notin A$, sinon on a une réponse positive.

```
count = 0
for line in f2:  # Single word per line
    is_in = True
    word = line.strip()
    for k in range(num_hash):
        index = functions[k](word, size)
        # If one hash function returns 0, the word is not in file1
        if bloom_filter[index] == 0:
            is_in = False
            break
    if is_in:
        count += 1
```

Les autres éléments du programme sont des choses qu'on a déjà utilisé dans les TP précédents.

Test / Benchmark

Dans ce qui suit, on va présenter quelques résultats obtenus et comparer les performances en fonctions des paramètres *m* et *k*.

	1	2	3	4	5	6	7
1,000	58,110	58,110	58,110	58,110	58,110	58,110	58,110

	1	2	3	4	5	6	7
10,000	53,411	56,962	56,962	57,940	58,090	58,090	58,096
100,000	25,872	22,361	22,365	21,339	21,382	21,372	21,867
1,000,000	17,216	16,048	16,049	15,809	15,767	15,767	15,767

(Pour exécuter ce benchmark, décommentez l'appel à la fonction "benchmark()")

Dans ce benchmark, on fait varier *m* (colonne) et *k* (ligne), le premier fichier est *"texte Shakespeare.txt"* et le second est *"corncob lowercase.txt"*.

Analyse

D'après le benchmark, on peut clairement voir que le résultat dépend fortement des paramètres d'entrée du programme :

- Avec *m* petit, le programme nous dit que tous les mots du deuxième fichier se trouvent dans le premier, ce qui est faux vu que le premier contient moins de mots que le second.
- Avec *m* grand, les résultats sont plus proche de la vérité (**15670**), même avec une seule fonction de hachage.
- Faire varier le nombre de fonctions de hachage peut permettre de se rapprocher de la solution sans avoir à trop augmenter la taille du filtre. Et si on utilise la formule $k = \frac{m}{n} ln(2)$, on peut trouver un bon compromis entre les deux, par exemple : pour $m = 670,680 \; et \; k = 8$, on obtient 15769 (9 faux positifs seulement).