BRAHIMI Ismail
AJLI Zakaria

Filtre de bloom

On va tester avec un tableau de 100 mots pour bien voir le traitement de chaque phase du code :

```
def remplir(tab):
    for i in range(len(tab)):
        for j in range(len(tab[i])):
            tab[i][j] = rand.randint(97, 122)
```

On rempli le tableau par des valeurs comprises entre 97 et 122 sachant que c'est l'intervalle des valeurs de (a-z) en code ASCII.

```
Remplissage du tableau avec  100  mots contenant  4  lettres :

affichage des mots :

r l o c
l b w q
m k i q
h x z d
v c o z
j x g u
z a s o
q d k s
f u q l
```

Ce qui donne l'affichage suivant ; un tableau de mots de taille 4.
La méthode de hachage par extraction des bits 1, 2 et 5, on traduit les bits en binaire on concatène puis on traduit en décimale.

```
def hashage_extraction(tab_):
    sum = ""
    for i in range(len(tab_)):
        #traduit en binaire, on concatene
        binaire = bin(tab_[i])[2:]
        sum = binaire[1]+binaire[2]+binaire[5]
    result = int(sum, 2) #puis on convertie en base decimale
    return result
```

La méthode de hachage est celle suivant l'exemple intuitif du cours; on attribut au lettres des valeurs de 1-26 calculer la somme des lettres de e puis calculer le mod.

```
def hashage_fct(tab_): #exemple intuitif de fonction de hachage
    sum = 0
    for i in range(len(tab_)):
        sum = sum + tab_[i]
    n = sum + len(tab_)
    n = n % 26
    return n
```

La méthode de hachage suivante est celle par compression; tout d'abord on traduit le mot en binaire puis on additionne les morceaux avec un XOR

```
# hashage par compression
def hash_compression(tab_):
    tabBin = [0 for i in range(len(tab_))]
    for i in range(len(tab_)):
        tabBin[i] = bin(tab_[i])[2:]
        #print(bin(tab_[i])[2:])
    resultatXoR = 0
    for i in range(0, len(tab_)):
        #On met un XOR ENTRE Les lettres du mot, sous forme de code binaire
        resultatXoR = bin(resultatXoR)[2:]
        resultatXoR = int(resultatXoR, 2) ^ int(tabBin[i], 2) #Bitwise XOR      x ^ y
    return resultatXoR % 26
```

La méthode de hachage suivante est celle par division; on traduit le mot en binaire, on le concatène, on calcule la somme des valeurs puis on convertie on base décimale, et on calcule la valeur mod N; tel que N la taille de notre table.

```
# hashage par division
def hash_devision(tab_):
    sum = ""
    for i in range(len(tab_)):
        #traduit en binaire, on concatene
        binaire = bin(tab_[i])[2:]
        sum = sum + binaire
    varDiv = int(sum, 2) #puis on convertie en base decimale
    result = varDiv % 26 #puis on calcule le modulo h(e) = e
    return result
```

La méthode de hachage suivante est celle par multiplication; le principe est simple on a déclaré une variable téta qui favorablement doit être téta = 0,6180 ou téta = 0,3819.
On traduit les variables du mot en binaire, on calcule la somme, on multiplie par téta puis son garde la partie décimale, on multiplie par la taille de la table puis on garde la partie entière du résultat.

```
# hashage par multiplication
def hash_multip(tab):
    teta = 0.6
    mult = 0
    sum = ""
    for i in range(len(tab)):
        # mult = tab[i]*teta
        binaire = bin(tab[i])[2:]
        sum = sum + binaire
    mult = int(sum, 2)
    result_mult = mult * teta
    result_mult = result_mult % 1 #on gard
    result_mult = result_mult * 26
    result_mult = math.floor(result_mult)
    return result_mult
```

On a déclaré une méthode <hachage> ou on fait appelle aux cinq méthodes de hachages et on mets une valeur de 1 dans chaque position du tableau.

```
def hachage(tab_):
    for i in range(len(tab_)):
        bloomFiltre = [0 for i in range(26)]
        #on calcule les cles avec 5 fonctions de hashage
        H1 = hashage_fct(tab_[i])
        H2 = hash_compression(tab_[i])
        H3 = hash_multip(tab_[i])
        H4 = hash_devision(tab_[i])
        H5 = hashage_extraction(tab_[i])
        #on place des 1 dans le tableaux de filtre de bloom
        bloomFiltre[H1] = 1
        bloomFiltre[H2] = 1
        bloomFiltre[H3] = 1
        bloomFiltre[H4] = 1
        bloomFiltre[H5] = 1
    print(bloomFiltre)
```

Méthode du calcule des faux positifs, tel que :
-k le nombre de fonction de hachage
-N la taille du tableau
-10000 le nombre d'élément d'une liste

```
def fauxPositive():
    #methode pour calculer les faux positifs
    P = math.pow((1 - math.exp((-10000 * k / N))), k)
    print(P)
```

L'affichage finale:

```
Remplissage du tableau avec  10000  mots contenant  4  lettres :

affichage des mots :

Representation du filtre apres insertion des cles :

[0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

taux de faux positive :

0.9667612155708726
```