Défi Big Data

Organisation des données #1

TP 3 - Structures internes de données

Lorrie PUGET - Rayan VERONNEAU

Compte-rendu du TP

Filtre de Bloom

Pour obtenir l'intersection des contenus des fichiers 1 et 2, nous avons implémenté un filtre de Bloom sur Python de la manière suivante :

```
def Bloom(fichier1, fichier2, m, k):
    debut = time.time()
    T=[0 for i in range(m)] # filtre de Bloom : tableau de taille m
    L_doublons=[] # liste des intersections trouvées dans les deux fichiers
    s1=read_fichier(fichier1)
    s2=read_fichier(fichier2)
    # On commence par hacher chaque mot du premier fichier k fois
    for mot in s1:
        for x in liste_fun[0:k]:
            i = x(mot, M) # hachage du mot par la fonction x
            T[i]+=1 # on ajoute 1 à la cellule i, valeur de hachage obtenu
            #pour mot par la fonction x
    # Phase de recherche des doublons
    for mot in s2:
        test=1
        for x in liste_fun[0:k]:
            i = x(mot, M)
            if T[i]==0: # on teste si le mot testé a la même valeur de
            # hachage qu'un mot du fichier précédent
                test=0 # si ce n'est pas le cas, le test est faux et ce n'est
                # pas un mot du fichier 1
        if test!=0: # si le mot testé a les mêmes valeurs de hachage qu'un mot
        # du fichier 1 alors il est dans le fichier 1 et 2 et on l'ajoute à la
        # liste des intersections
            L_doublons.append(mot)
    fin=time.time()

    # Suppression des doublons de l'intersection
    L_sans_doublons = list(set(L_doublons))
    return L_sans_doublons, len(L_sans_doublons),fin - debut # on retourne
    # l'intersection des fichiers et le temps d'execution du filtre
```

Avant d'implémenter notre fonction, nous avons préalablement défini plusieurs variables :

- M=834000 : taille de la table de hachage comme expliqué dans le TP1
- n = 250000 : nombre de mots dans un fichier
- k = 4 : nombre de fonction de hachage utilisée lors du premier test
- m = $\frac{k*n}{ln(2)}$: taille optimale du filtre de Bloom

En ayant testé le filtre de Bloom avec k = 4, on obtient une liste d'intersections de taille 144221 mots en 23 secondes.

Implémentation du filtre en un tableau de bits

Avoir un filtre de Bloom en un tableau d'entiers courts permet d'avoir accès au nombre de collisions pour chaque valeur de hachage. Néanmoins, nous pouvons également réaliser ce filtre en un tableau de bits avec et sans le nombre de collisions.

- *Sans le nombre de collisions : on teste si la cellule T[i]=1, si non, on l'égale à 1*

```
def Bloom(fichier1, fichier2, m, k):
    debut = time.time()
    T=[0 for i in range(m)] # filtre de Bloom : tableau de taille m
    L_doublons=[] # liste des intersections trouvées dans les deux fichiers
    s1=read_fichier(fichier1)
    s2=read_fichier(fichier2)
    # On commence par hacher chaque mot du premier fichier k fois
    for mot in s1:
        for x in liste_fun[0:k]:
            i = x(mot, M) # hachage du mot par la fonction x
            if T[i]!=1:
                T[i]=1 # on ajoute 1 à la cellule i, valeur de hachage obtenu
                #pour mot par la fonction x
    # Phase de recherche des doublons
    for mot in s2:
        test=1
        for x in liste_fun[0:k]:
            i = x(mot, M)
            if T[i]==0: # on teste si le mot testé a la même valeur de
            # hachage qu'un mot du fichier précédent
                test=0 # si ce n'est pas le cas, le test est faux et ce n'est
                # pas un mot du fichier 1
        if test!=0: # si le mot testé a les mêmes valeurs de hachage qu'un mot
        # du fichier 1 alors il est dans le fichier 1 et 2 et on l'ajoute à la
        # liste des intersections
            L_doublons.append(mot)
    fin=time.time()

    # Suppression des doublons de l'intersection
    L_sans_doublons = list(set(L_doublons))
    return L_sans_doublons, len(L_sans_doublons),fin - debut # on retourne
    # l'intersection des fichiers et le temps d'execution du filtre
```

- *Avec le nombre de collisions : au lieu de faire un filtre de Bloom initialement en tableau nul, on l'initialise en tableau de liste nulle ie T=[[0],[0],....,[0]] puis en testant, on crée des listes de 1 pour garder des informations sur les collisions*

```python
def Bloom(fichier1, fichier2, m, k):
    debut = time.time()
    T=[[0] for i in range(m)] # filtre de Bloom : tableau de taille m
    L_doublons=[] # liste des intersections trouvées dans les deux fichiers
    s1=read_fichier(fichier1)
    s2=read_fichier(fichier2)
    # On commence par hacher chaque mot du premier fichier k fois
    for mot in s1:
        for x in liste_fun[0:k]:
            i = x(mot, M) # hachage du mot par la fonction x
            if T[i][0]==0:
                T[i][0]=1 # on égalise T[i][0] à 1
            if T[i][0]!=1:
                T[i].append(1) # on ajoute un 1 à la liste T[i]
    # Phase de recherche des doublons
    for mot in s2:
        test=1
        for x in liste_fun[0:k]:
            i = x(mot, M)
            if T[i][0]==0: # on teste si le mot testé a la même valeur de
            # hachage qu'un mot du fichier précédent
                test=0 # si ce n'est pas le cas, le test est faux et ce n'est
                # pas un mot du fichier 1
        if test!=0: # si le mot testé a les mêmes valeurs de hachage qu'un mot
        # du fichier 1 alors il est dans le fichier 1 et 2 et on l'ajoute à la
        # liste des intersections
            L_doublons.append(mot)
    fin=time.time()

    # Suppression des doublons de l'intersection
    L_sans_doublons = list(set(L_doublons))
    return L_sans_doublons, len(L_sans_doublons),fin - debut # on retourne
    # l'intersection des fichiers et le temps d'execution du filtre
```

Test en faisant varier m et k

Avec k=4, on obtient une probabilité de faux positifs de 6%. Or, dans le TP2, nous avions obtenu un taux de faux positifs de 3%. Nous avions en effet une liste de doublons de 13821 mots soit 400 de différences avec ce que nous avons obtenu ici avec k=4. L'erreur théorique est donc supérieure à l'erreur expérimentale ici. Néanmoins, nous restons dans le même ordre de grandeur.
Avec k=7, on obtient une intersection de taille 13831 soit 10 de différence avec ce qu'on avait obtenu dans le TP2. L'erreur expérimentale est donc ici beaucoup plus petite (de l'ordre de 10^{-3}) alors que l'erreur théorique est de l'ordre de 10^{-2}.
Avec k=11, on obtient 138822 soit un mot faux positif. L'erreur expérimentale est donc ici beaucoup plus petite (de l'ordre de 10^{-4}) alors que l'erreur théorique est de l'ordre de 10^{-3}. Par ailleurs, dans ce cas, le temps d'exécution est de 31s.

On voit que plus le nombre de fonctions de hachage augmente, plus l'erreur expérimentale et l'erreur théorique diminuent. Néanmoins, elles ne sont pas égales et gardent un ordre de grandeur d'écart.