ESISAR

CS353 - Algorithmique

TP numéro 7 et 8
Filtres de Bloom

Table des matières

1 Objectifs du TP numéro 7 et 8

Ces deux séances de TP vont permettre l'implémentation d'un filtre de Bloom(vu en cours).

Pour faire ces 2 TP, vous avez besoin de :

- une séance de 1h30 avec l'enseignant en salle de TP pour commencer
- 4H de préparation entre les 2 TP
- une autre séance de 1h30 avec l'enseignant en salle de TP pour terminer

Tous les programmes réalisés seront obligatoirement en Java. Le rendu se fera à l'aide de la grille de réponse disponible sur Chamilo.

2 Les documents du TD

Relisez votre cours sur le filtre de Bloom.

Vous pouvez aussi lire
https://en.wikipedia.org/wiki/Bloom_filter

3 Préparation des exercices

3.1 Les fichiers nécessaires.

Pour pouvoir faire les exercices, vous avez besoin de 3 fichiers :

- infected-urls.txt
- valides-urls.txt
- test-url.txt

Le fichier **infected-urls.txt** contient 20 millions d'URL correspondant à des sites douteux ou infectés par des virus.

Le fichier **valides-urls.txt** contient 20 millions d'URL légitimes (sans virus connu). Aucune URL de ce fichier n'est incluse dans le fichier infected_urls.txt.

Le fichier **test-url.txt** contient 20 millions d'URL. Certaines de ces URLs sont infectés, donc font partie du fichier infected-urls . Les autres sont des URLs légitimes.

Pour pouvoir obtenir ces 3 fichiers, vous devez télécharger sur Chamilo le fichier **url.jar**.

Vous vous placez ensuite dans le répertoire où vous avez mis votre fichier url.jar, et vous lancer la commande

```
java -jar url.jar
```

Ceci va lancer un programme un java qui va créer 3 fichiers de 1Go (infected-urls.txt , valides-urls.txt , test-url.txt).

Pour vérifier que les fichiers en votre possession sont bien corrects, voici les hashs MD5 des fichiers :

Nom du fichier	Hash MD5
infected-urls.txt	b1 04 3c 3b a6 e3 3f 91 25 b0 de e8 66 57 57 1f
valides-urls.txt	54 92 d9 23 5b 4e bc 36 6d 34 d3 ea f3 aa a4 b6
test-url.txt	63 f2 cb 92 04 70 9a 22 07 92 07 32 c8 a5 83 b8

Sous Linux, la commande md5sum vous permet de calculer facilement le hash d'un fichier.

3.2 La structures des fichiers

Le fichier **infected-urls.txt** contient donc 20 millions d'URL douteuses.

Sa structure est la suivante :
- une première URL sur 50 caractères
- une seconde URL sur 50 caractères
- et ainsi de suite

Si l'URL fait moins de 50 caractères, alors elle est complétée avec des espaces pour arriver à 50.
Il n'y a pas d'URL de plus de 50 caractères.

Exemple avec le début du fichier

```
http://sxv.aw/BFHuRvb                    http://rastvybcwvmgn.lk/tFRZvWebpA5WHf
http://ylueg.do/XQqPyuxbr                ht...
```

Il y aucun retour à ligne dans tout le fichier.

Les deux autres fichiers valides-urls.txt , test-url.txt ont exactement la même structure.

4 Exercice 1

Votre objectif est de construire un filtre de Bloom , prenant en entrée une URL, et qui répondra si l'URL fait partie ou non des URL douteuses(comme tout filtre de Bloom, il peut apparaître des faux positifs, mais par contre il ne peut pas apparaître de faux négatifs).

Pour cela, vous prendrez :
- m : taille en bits du filtre de Bloom : 200 millions de bits
- k : nombre de fonction de hachage : 11

Voici ci dessous quelques indications pour l'implémentation :

4.1 Les fonctions de hachage

Pour un filtre de Bloom, si vous souhaitez insérer la chaîne "xyz", vous avez besoin hasher k fois la chaîne "xyz" avec k fonctions de hachage indépendantes.

Ceci s'obtient en invoquant hash("xyz",0) puis hash("xyz",1) puis .. puis hash("xyz",k-1).

```
	private int hash(String value,int numFonction)
	{
		int h1 = h1(value);
		int h2 = h2(value);

		int h = ( (h1+numFonction*h2) % m);
		h = (h+m) % m;

		return h;
	}


	private int h1(String value)
	{
		 char val[] = value.toCharArray();

		 int h=0;
	   for (int i = 0; i < val.length; i++)
	   {
	       h = 31 * h + val[i];
	   }
	   return h;

	}


	private int h2(String value)
	{
		 char val[] = value.toCharArray();

		 int h=0;
	   for (int i = 0; i < val.length; i++)
	   {
	       h = 57 * (h <<2) + val[i];
	   }
	   return h;

	}
```

Pour la justification théorique, je vous conseille la lecture de
https://www.eecs.harvard.edu/~michaelm/postscripts/tr-02-05.pdf

4.2 Comment lire les fichiers infected-urls.txt, ...

Pour lire le fichier infected-urls.txt, je vous propose le code suivant (simple mais un peu lent : 20 secondes pour lire tout le fichier) :

```
long start = System.currentTimeMillis();
```

```
SeekableByteChannel sbc = Files.newByteChannel(Paths.get("/tmp/url.txt"),
StandardOpenOption.READ);

int count =0;

// Lecture des 50 premiers octets du fichier
ByteBuffer buf = ByteBuffer.allocate(50);
int ret = sbc.read(buf);

while(ret>0)
{
	// Conversion des 50 octets en une chaine de caractères
	String str = new String(buf.array());

	// Suppression des espaces en fin de chaine
	str = str.trim();

	// Affichage de la chaine de caractères sur la console
	System.out.println(str);
	count++;

	// Lecture des 50 octets suivants
	buf.clear();
	ret = sbc.read(buf);
}

System.out.println("Nombre d'URL dans le fichier ="+count);
System.out.println("Elapsed Time="+(System.currentTimeMillis()-start)/1000+"s");

// Fermeture du fichier
sbc.close();
```

Voici maintenant une version plus optimisée (1 seconde pour lire tout le fichier) :

```
long start = System.currentTimeMillis();

SeekableByteChannel sbc = Files.newByteChannel(Paths.get("c:/tmp/url1.txt"),
StandardOpenOption.READ);

int count =0;

// Lecture des 50 000 premiers octets du fichier
ByteBuffer buf = ByteBuffer.allocate(50_000);
int ret = sbc.read(buf);

while(ret>0)
{
	// Conversion des 50_0000 octets en 1000 chaine de caractères
	for (int i = 0; i < 1000; i++)
	{
		String str = new String(buf.array(),i*50,50);

		// Suppression des espaces en fin de chaine
		str = str.trim();

		// Affichage de la chaine de caractères sur la console
```

```
                // System.out.println(str);
                count++;
        }



        // Lecture des 50_000 octets suivants
        buf.clear();
        ret = sbc.read(buf);
}

System.out.println("Nombre d'URL dans le fichier ="+count);
System.out.println("Elapsed Time="+(System.currentTimeMillis()-start)/1000+"s");


// Fermeture du fichier
sbc.close();
```

5 Exercice 2 : utilisation du filtre de Bloom

Faites un programme qui :

- lit les 20 millions d'URL valides contenues dans le fichier « valides_urls.txt »
- demande pour chaque URL au filtre de Bloom si celle ci est infectée ou non.

Déduisez en le taux de faux positifs de votre filtre de Bloom.

6 Exercice 3 : variation de k

Faites le même exercice que l'exercice 2, mais en faisant varier k entre 1 et 20.

Remplissez le tableau suivant :

K	Taux de faux positifs en %
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

K	Taux de faux positifs en %
12	
13	
14	
15	
16	
17	
18	
19	
20	

7 Exercice 4 – Une application concrète

L'objectif est de réaliser un programme :

- qui occupe au maximum de 512 Mo en mémoire vive
- qui est capable de dire le plus rapidement possible si une URL est infectée ou non, et ceci de façon certaine (c'est à dire sans faux positifs et sans faux négatifs)

Votre programme fonctionnera de la façon suivante :

- il utilisera d'abord un filtre de Bloom en mémoire pour savoir si l'URL est infectée
- en cas de réponse positive, il utilisera un fichier sur le disque dur de la machine pour éliminer les faux positifs et répondre avec certitude.
- Ce fichier sur le disque dur aura été calculé par vos soins à l'avance et peut contenir ce que vous voulez dans le format que vous voulez, sa taille doit seulement être inférieure à 100 Go. C'est à vous d'imaginer sa structure !

Voici ci dessous quelques indications pour l'implémentation :

7.1 Gérer la mémoire de son programme.

Pour connaître la mémoire utilisée par votre programme, vous pouvez utiliser la commande top sous Linux.

Pour obliger votre programme à utiliser au maximum 512 Mo de mémoire, vous allez dans Eclipse, puis Run puis Run Configurations puis Arguments. Dans la case « VM Arguments », vous ajouter le texte « -Xmx512M » puis vous cliquez sur Run. Désormais, votre programme ne

pourra pas consommer plus de 512 Mo de mémoire.

7.2 Gérer l'accès à un fichier en Java.

Considérons un fichier de 100 000 000 octets. Vous souhaitez lire les octets compris entre la position 1000 et 1050 de ce fichier. Voici un exemple pour le faire :

```java
SeekableByteChannel sbc = Files.newByteChannel(Paths.get("c:/tmp/url1.txt"), StandardOpenOption.READ);

// On se déplace à la position 1000 dans le fichier
sbc.position(1000);

// Lecture de 50 octets
ByteBuffer buf = ByteBuffer.allocate(50);
int ret = sbc.read(buf);

// Conversion des 50 octets en une chaine de caractères
String str = new String(buf.array());

// Suppression des espaces en fin de chaine
str = str.trim();

// Affichage de la chaine de caractères sur la console
System.out.println(str);

// Fermeture du fichier
sbc.close();
```

Considérons un fichier de 100 000 000 octets. Vous souhaitez écrire les octets compris entre la position 1000 et 1050 de ce fichier. Voici un exemple pour le faire :

```java
SeekableByteChannel sbc = Files.newByteChannel(Paths.get("c:/tmp/small-url1.txt"), StandardOpenOption.WRITE);

// On se déplace à la position 1000 dans le fichier
sbc.position(1000);

// Ecriture de 50 octets
String content = "01234567890123456789012345678901234567890123456789";
ByteBuffer buf = ByteBuffer.wrap(content.getBytes());
sbc.write(buf);

// Fermeture du fichier
sbc.close();
```

Attention : ceci fonctionne si votre fichier a déjà une taille d'au moins 1000 octets (sinon le `sbc.position(1000);` ne marchera pas).

Voici un exemple pour créer un fichier de 1 000 000 octets avec uniquement des espaces par exemple :

```
SeekableByteChannel sbc = Files.newByteChannel(Paths.get("c:/tmp/url4-small.txt"), StandardOpenOption.WRITE ,
StandardOpenOption.CREATE_NEW);

// Création d'un byte array  de 1000 espaces
byte[] bs = new byte[1000];
for (int i = 0; i < bs.length; i++)
{
        bs[i] = ' ';
}

// Copie de ces 1000 espaces 1000 fois
for (int i = 0; i < 1000; i++)
{
        ByteBuffer buf = ByteBuffer.wrap(bs);
        sbc.write(buf);
}

// Fermeture du fichier qui contient donc 1 million d'espaces
sbc.close();
```

7.3 Tester votre programme.

Pour tester votre programme, utilisez le fichier « test-url.txt ».

Ce fichier contient 20 millions d'url, déterminer le nombre d'url réellement douteuses dans ce fichier.

Combien de temps votre programme met il pour trouver le résultat ?