SAE 3.8 Algorithmique Projet – Filtre de Bloom

Rapport

IBOS
Geoffrey

S3A'

Sommaire

1. Objectif

L’objectif de ce projet est d’implémenter un filtre de Bloom.

Pour cela, il faut utiliser plusieurs structures de données et comparer ces différentes implémentations par un banc d’essai (benchmark).

Ce banc d'essai permet de calculer le temps d'exécution pour la recherche d'éléments dans le filtre et le taux d'erreurs du test d'appartenance, pour chaque implémentation du filtre de Bloom : tableau, ArrayList et LinkedList.

2. Présentation du filtre de Bloom

Un filtre de Bloom est une structure de données permettant de tester de façon efficace la présence ou non d'un élément dans le filtre.
En particulier, le filtre de Bloom permet de tester :
• avec certitude : l'absence d'un élément
• avec une certaine probabilité : la présence d'un élément

Un filtre de Bloom est constitué d'un tableau de booléens (ou de bits) de taille m, ainsi qu'une collection de k fonctions de hachage. Chaque fonction de hachage associe tout élément à une case du tableau.

Pour ajouter un élément, on met true dans les cases des indices calculés par les fonctions de hachage.
Pour tester si un élément est présent, on vérifie que ces cases contiennent toutes true. Il se peut que le test d'appartenance renvoie vrai alors que l'élément est absent, c'est un faux positif.

3. Explication du code

Tout d'abord, il y a plusieurs paquets : "app" contenant la classe qui lance l'interface (la partie IHM), "main" avec la classe Main, "s3.pkg8_algorithmique" avec toute la partie métier (implémentation du filtre pour chaque variante) et "test" qui contient le banc d'essai (benchmark).

L'interface IFiltreBloom permet de déclarer les méthodes qui servent notamment à ajouter et tester la présence d'éléments dans le filtre.

La classe abstraite AFiltreBloom implémente l'interface IFiltreBloom et définie la méthode de hachage, la taille du filtre et le nombre de hachages.

Enfin, les classes FiltreBloom_Tableau, FiltreBloom_ArrayList et FiltreBloom_LinkedList étendent AFiltreBloom. Elles implémentent respectivement un tableau, une ArrayList et une LinkedList comme filtre.

4. Tableau

Tests pour le tableau, avec m = 100 000, k = 1 et n = 1 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :

Temps moyen d'exécution pour la recherche de 1000 éléments dans
un filtre de taille 100000 et avec 1 hachages : 421 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 100000 et n = 1000.0 est de 0.7000000000000001 %
```

Temps : 421 nanosecondes

Taux d'erreurs : 0,7 %

4. Tableau

Tests pour le tableau, avec m = 100 000, k = 3 et n = 5 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :

Temps moyen d'exécution pour la recherche de 5000 éléments dans
un filtre de taille 100000 et avec 3 hachages : 359 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 100000 et n = 5000.0 est de 2.6599999999999997 %
```

Temps : 359 nanosecondes

Taux d'erreurs : 2,66 %

4. Tableau

Tests pour le tableau, avec m = 100 000, k = 5 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :

Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 100000 et avec 5 hachages : 420 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 100000 et n = 10000.0 est de 3.71 %
```

Temps : 420 nanosecondes

Taux d'erreurs : 3,71 %

4. Tableau

Tests pour le tableau, avec m = 200 000, k = 1 et n = 2 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :
Temps moyen d'exécution pour la recherche de 2000 éléments dans
un filtre de taille 200000 et avec 1 hachages : 323 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 200000 et n = 2000.0 est de 0.6 %
```

Temps : 323 nanosecondes

Taux d'erreurs : 0,6 %

4. Tableau

Tests pour le tableau, avec m = 200 000, k = 3 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :
Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 200000 et avec 3 hachages : 330 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 200000 et n = 10000.0 est de 1.81 %
```

Temps : 330 nanosecondes

Taux d'erreurs : 1,81 %

4. Tableau

Tests pour le tableau, avec m = 200 000, k = 5 et n = 20 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec un tableau :
Temps moyen d'exécution pour la recherche de 20000 éléments dans
un filtre de taille 200000 et avec 5 hachages : 234 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 200000 et n = 20000.0 est de 2.005 %
```

Temps : 234 nanosecondes

Taux d'erreurs : 2.005 %

5. ArrayList

Tests pour l'ArrayList, avec m = 100 000, k = 1 et n = 1 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :

Temps moyen d'exécution pour la recherche de 1000 éléments dans
un filtre de taille 100000 et avec 1 hachages : 602 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 100000 et n = 1000.0 est de 0.6 %
```

Temps : 602 nanosecondes

Taux d'erreurs : 0,6 %

5. ArrayList

Tests pour l'ArrayList, avec m = 100 000, k = 3 et n = 5 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :

Temps moyen d'exécution pour la recherche de 5000 éléments dans
un filtre de taille 100000 et avec 3 hachages : 411 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 100000 et n = 5000.0 est de 2.76 %
```

Temps : 411 nanosecondes

Taux d'erreurs : 2,76 %

5. ArrayList

Tests pour l'ArrayList, avec m = 100 000, k = 5 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :

Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 100000 et avec 5 hachages : 463 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 100000 et n = 10000.0 est de 3.7199999999999998 %
```

Temps : 463 nanosecondes

Taux d'erreurs : 3,72 %

5. ArrayList

Tests pour l'ArrayList, avec m = 200 000, k = 1 et n = 2 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :
Temps moyen d'exécution pour la recherche de 2000 éléments dans
un filtre de taille 200000 et avec 1 hachages : 488 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 200000 et n = 2000.0 est de 0.65 %
```

Temps : 488 nanosecondes

Taux d'erreurs : 0,65 %

5. ArrayList

Tests pour l'ArrayList, avec m = 200 000, k = 3 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :
Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 200000 et avec 3 hachages : 316 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 200000 et n = 10000.0 est de 1.9900000000000002 %
```

Temps : 316 nanosecondes

Taux d'erreurs : 1,99 %

5. ArrayList

Tests pour l'ArrayList, avec m = 200 000, k = 5 et n = 20 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une ArrayList :
Temps moyen d'exécution pour la recherche de 20000 éléments dans
un filtre de taille 200000 et avec 5 hachages : 214 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 200000 et n = 20000.0 est de 1.975 %
```

Temps : 214 nanosecondes

Taux d'erreurs : 1,975 %

6. LinkedList

Tests pour la LinkedList, avec m = 100 000, k = 1 et n = 1 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :

Temps moyen d'exécution pour la recherche de 1000 éléments dans
un filtre de taille 100000 et avec 1 hachages : 110238 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 100000 et n = 1000.0 est de 0.8999999999999999 %
```

Temps : 110 238 nanosecondes

Taux d'erreurs : 0,9 %

6. LinkedList

Tests pour la LinkedList, avec m = 100 000, k = 3 et n = 5 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :

Temps moyen d'exécution pour la recherche de 5000 éléments dans
un filtre de taille 100000 et avec 3 hachages : 306420 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 100000 et n = 5000.0 est de 2.8000000000000003 %
```

Temps : 306 420 nanosecondes

Taux d'erreurs : 2,8 %

6. LinkedList

Tests pour la LinkedList, avec m = 100 000, k = 5 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :

Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 100000 et avec 5 hachages : 505921 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 100000 et n = 10000.0 est de 3.55 %
```

Temps : 505 921 nanosecondes

Taux d'erreurs : 3,55 %

6. LinkedList

Tests pour la LinkedList, avec m = 200 000, k = 1 et n = 2 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :
Temps moyen d'exécution pour la recherche de 2000 éléments dans
un filtre de taille 200000 et avec 1 hachages : 190281 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 1,
m = 200000 et n = 2000.0 est de 0.7000000000000001 %
```

Temps : 190 281 nanosecondes

Taux d'erreurs : 0,7 %

6. LinkedList

Tests pour la LinkedList, avec m = 200 000, k = 3 et n = 10 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :
Temps moyen d'exécution pour la recherche de 10000 éléments dans
un filtre de taille 200000 et avec 3 hachages : 642612 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 3,
m = 200000 et n = 10000.0 est de 1.6400000000000001 %
```

Temps : 642 612 nanosecondes

Taux d'erreurs : 1,64 %

6. LinkedList

Tests pour la LinkedList, avec m = 200 000, k = 5 et n = 20 000.

Temps d'exécution pour la recherche de n éléments dans le filtre et taux d'erreurs :

```
Avec une LinkedList :
Temps moyen d'exécution pour la recherche de 20000 éléments dans
un filtre de taille 200000 et avec 5 hachages : 1139617 nanosecondes
Le taux d'erreurs du test d'appartenance avec k = 5,
m = 200000 et n = 20000.0 est de 1.8950000000000002 %
```

Temps : 1 139 617 nanosecondes

Taux d'erreurs : 1,90 %

7. Conclusion

Tout d'abord, nous pouvons voir que c'est globalement le tableau qui est le plus rapide pour la recherche d'éléments, ensuite c'est l'ArrayList (très légèrement derrière) et enfin c'est la LinkedList qui est de loin la plus lente.

Pour ce qui est du taux d'erreurs, on voit qu'il varie en fonction de chaque implémentation mais il est en moyenne à peu près le même pour le tableau, l'ArrayList et la LinkedList (variation dues aux différents nombres random ajoutés au filtre par exemple).

En revanche, nous pouvons remarquer que plus il y a d'éléments dans le filtre plus il y a d'erreurs et que plus il y a de hachages effectués, plus c'est précis et donc moins il y a d'erreurs.
Donc le nombre de hachages dépend du taux de faux positifs souhaité.
Cependant, un plus grand nombre k de hachages augmente très légèrement le temps de recherche (car il faut hacher k fois l'élément pour le rechercher dans le filtre).

Pour finir, un des inconvénients du filtre de Bloom est que plus il y a d'éléments et plus il y a de faux positifs.