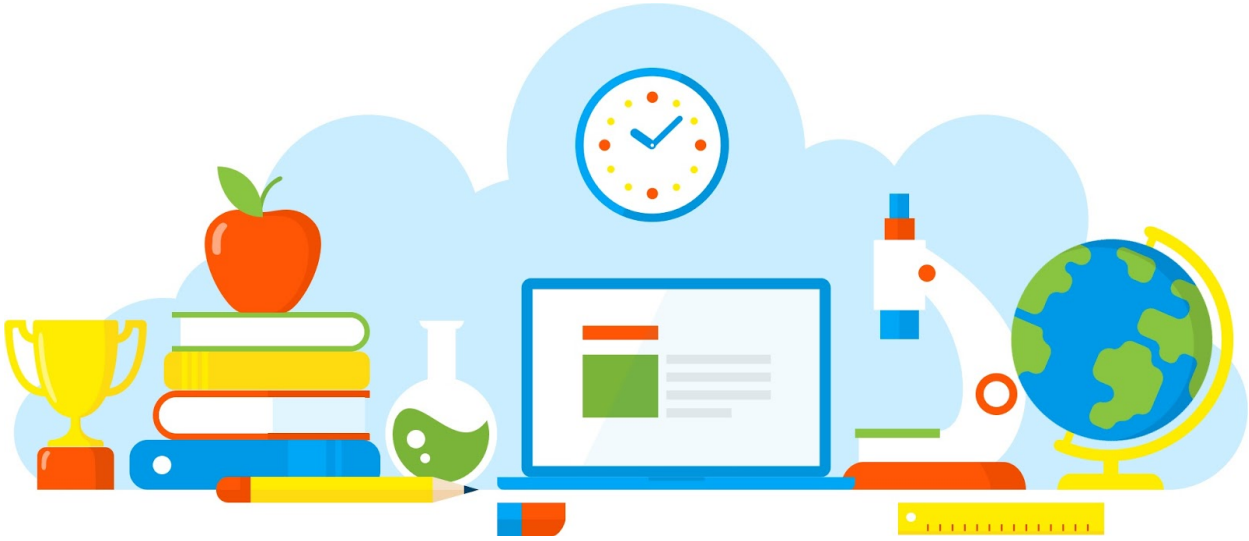


Rapport Filtre de Bloom

Sourdois Pajot Valentin
18/12/2022

Sommaire

Introduction

Un filtre de Bloom est une structure de données probabiliste utilisée pour tester l'appartenance d'un élément à un ensemble. Il se compose d'un tableau de bits de taille prédéterminée et de plusieurs fonctions de hachage. Pour ajouter un élément au filtre, on calcule les valeurs de hachage de cet élément en utilisant les fonctions de hachage et on met à 1 les bits correspondants dans le tableau de bits. Pour tester l'appartenance d'un élément à l'ensemble, on calcule à nouveau les valeurs de hachage de cet élément et on vérifie l'état des bits correspondants dans le tableau de bits. Si tous les bits sont à 1, il y a de fortes chances que l'élément appartienne à l'ensemble. Cependant, il est possible que certains bits soient à 1 même si l'élément n'appartient pas à l'ensemble, ce qui entraîne un faux positif. Le taux de faux positif d'un filtre de Bloom dépend de la taille du tableau de bits et du nombre de fonctions de hachage utilisées.

Banc d'essai (Benchmark)

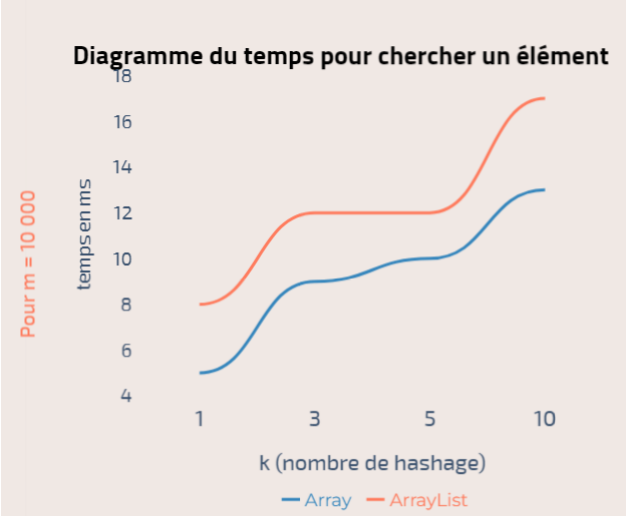
Un banc d'essai (benchmark) pour le filtre de Bloom est un ensemble de tests qui permettent de mesurer les performances d'un filtre de Bloom en termes de vitesse et de précision. Il peut être utilisé pour comparer différentes implémentations de filtres de Bloom ou pour évaluer l'influence de différents paramètres sur les performances du filtre.

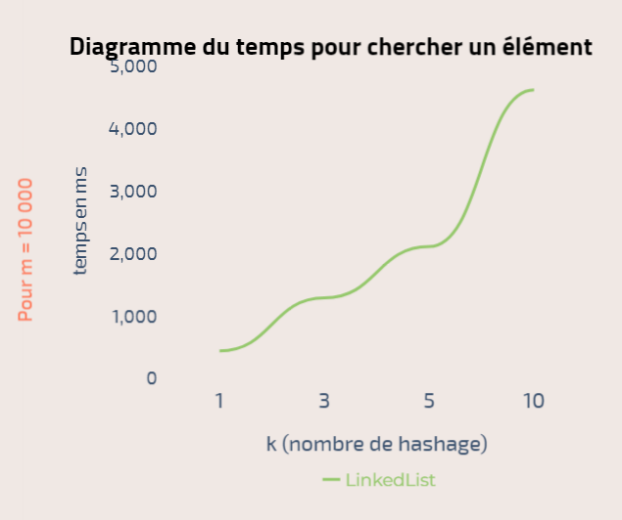
Pour réaliser un banc d'essai pour un filtre de Bloom, on peut utiliser différentes approches en fonction des objectifs visés. Par exemple, on peut mesurer le temps nécessaire pour tester leur appartenance à l'ensemble, ou bien on peut générer aléatoirement des éléments qui ne font pas partie de l'ensemble et mesurer le taux de faux positifs obtenus. On peut également combiner ces approches en mesurant à la fois le temps d'exécution et le taux de faux positifs pour différents ensembles de données et différents paramètres du filtre.

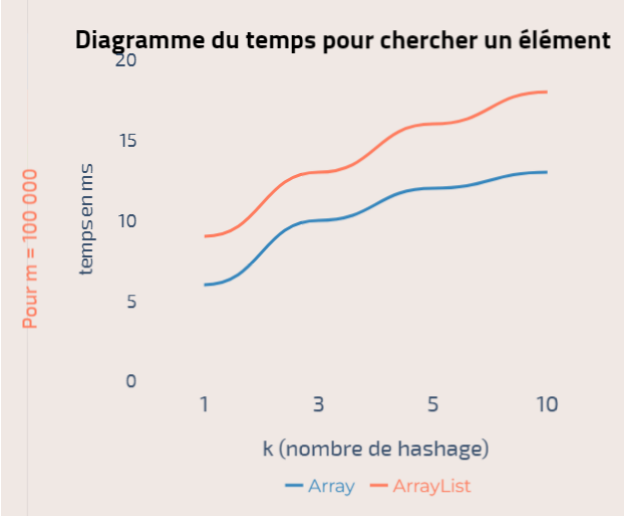
Temps d'exécutions

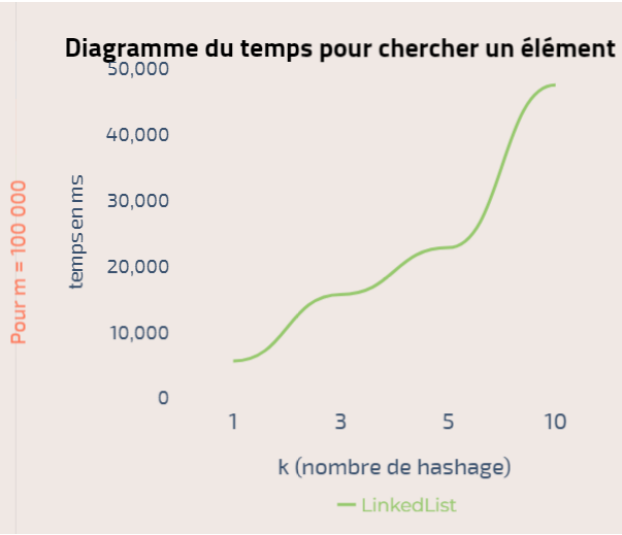
Pour les temps d'exécutions, j'ai décidé de le représenter sous forme de graphe, mais j'ai séparé la LinkedList de l'Array et de l'ArrayList car les échelles de temps sont beaucoup trop différentes. L'abscisse représente l'évolution de k, celle des ordonnées, le temps.

Il y a 2 séries de graphes car j'ai testé les temps d'exécutions pour m =10 000 et pour m = 100 000. En voici les résultats (on les interprétera plus tard) :





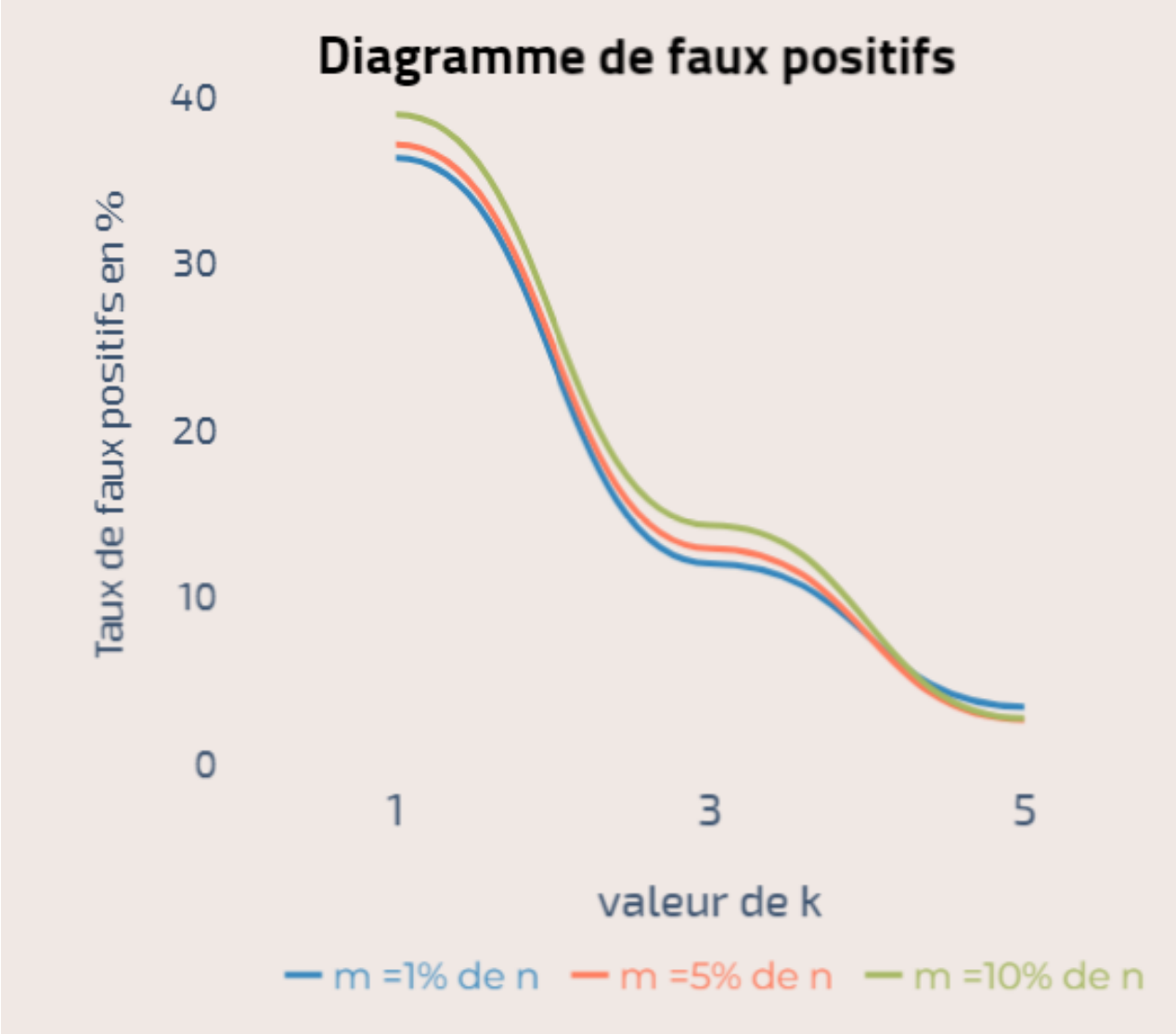




Taux d'erreur

Pour le taux d'erreur (taux de faux positifs) il y en avait 9 à calculer, pour k = 1,3,5 et pour chaque k, m qui est égale à respectivement 1, 5, 10% de n (3*3 = 9).

J'ai personnellement choisi de réaliser ce test avec l'implémentation utilisant l'ArrayList, étant donné qu il est entre les deux.



Analyse des résultats

Que ce soit pour les diagrammes ou l'analyse, on est partie sur n (nombre d'éléments qu'on ajoute dans le filtre) valant 100 000.

Commençons par les temps d'exécutions, on voit déjà que la LinkedList est drastiquement plus lente que l'Array et l'ArrayList, j'expliquerais pourquoi dans ma conclusion. Ce que nous voyons ensuite, c'est que l'Array est à chaque fois meilleur que l'ArrayList mais ce n'est pas forcément significatif vu la différence de temps. Maintenons la troisième chose que nous voyons, c'est que le temps va varier en fonction de deux paramètres : k à savoir le nombre de hachages et m la taille du "Tableau" dans le filtre.

Maintenant le taux d'erreur, toute l'analyse que je fournirais dans les prochains ligne ne concernera que l'implémentation via l'ArrayList et donc ne sera peut-être pas vrai pour les autres. Ce que nous voyons, c'est que là il n'y a qu'un seul paramètre vraiment important, k donc le nombre de hachages, pour un k=1 nous serons aux alentours des 35 40%, pour k=3 entre 10 et 15% et pour k=5 entre 2 et 4%, mais ce taux sera a chaque fois croissant en fonction du m qui augmente aussi. Cela nous permet de voir que plus k sera élevé et plus m sera bas, plus nous aurons un taux de faux positif bas et donc une précision élevée.

Conclusions

Si jamais vous voulez implémenter un filtre de bloom, veuillez utiliser un nombre de fonctions de hachages élevés et Il n'est pas judicieux d'utiliser une linkedList (liste chaînée) pour un filtre de Bloom car cela peut entraîner des performances inférieures par rapport à d'autres structures de données plus adaptées.

Une linkedList est une structure de données qui consiste en une série de noeuds liés entre eux par des pointeurs. Chaque noeud contient une valeur et un pointeur vers le noeud suivant dans la liste. Pour ajouter ou supprimer un élément dans une linkedList, il faut mettre à jour les pointeurs des noeuds adjacents, ce qui peut être coûteux en termes de temps d'exécution.

Dans le cas d'un filtre de Bloom, il est nécessaire de mettre à jour le tableau de bits de manière fréquente lors de l'ajout ou de la suppression d'éléments. Si on utilise une linkedList pour stocker le tableau de bits, cela peut entraîner une augmentation significative du temps d'exécution des requêtes.