Analyse des Arbres de Recherche

1. Introduction

Le projet consiste à tester l'efficacité pratique des méthodes d'ajout et de suppression dans un arbre binaire de recherche (ABR). Pour cela, nous allons générer un ensemble de 10 000 éléments aléatoirement entre 0 et 1 000 000, puis en conserver 1000 pour la suppression ultérieure.

Ensuite, on construit l'arbre en insérant successivement tous les éléments dans l'arbre. Si deux éléments ont la même clé, la solution retenue consiste à ne pas les insérer.

On mesure ensuite les temps d'exécution moyens pour 1000 insertions supplémentaires, toujours aléatoires et sans duplication, ainsi que pour la suppression des 1000 éléments conservés dans le tableau “toDelete”.

L'objectif final est d'analyser les temps d'exécution pour comparer l'efficacité pratique des méthodes d'ajout et de suppression dans un ABR.

(Le code est en pièce-jointe)

2. Analyse du code

(J'ai essayé de commenter mon code au maximum, les détails seront expliqués grâce à ces derniers)

Ce code crée un ensemble de valeurs aléatoires et les ajoute dans un arbre binaire de recherche (ABR). Ensuite, il supprime un nombre aléatoire d'éléments de l'arbre binaire de recherche et mesure le temps d'insertion et de suppression.

La taille de l'ensemble d'éléments est définie par la constante N, le nombre d'éléments à supprimer est défini par la constante M, et la valeur maximale possible pour les éléments est définie par la constante MAX.

Contenu des classes

- La classe Main contient la méthode main qui exécute le code. Elle crée un tableau d'entiers aléatoires en utilisant la méthode generateRandomArray et récupère un certain nombre d'éléments aléatoires de ce tableau en utilisant la méthode getRandomElements. Ensuite, elle crée un nouvel arbre binaire de recherche en utilisant la classe ABR et ajoute chaque élément du

tableau dans l'arbre en utilisant la méthode ajouter. Elle mesure ensuite le temps moyen pour l'insertion en utilisant la méthode System.nanoTime. Ensuite, elle supprime chaque élément aléatoire de l'ensemble précédent de l'arbre en utilisant la méthode supprimer et mesure le temps moyen pour la suppression. Enfin, elle affiche la hauteur de l'arbre binaire de recherche en utilisant la méthode hauteur.

- La classe Noeud représente un noeud dans l'arbre binaire de recherche. Chaque noeud a une valeur et deux enfants : un enfant gauche et un enfant droit.

- La classe ABR représente l'arbre binaire de recherche. Elle contient une référence vers la racine de l'arbre. Elle a une méthode ajouter qui ajoute une valeur dans l'arbre et une méthode supprimer qui supprime un noeud ayant la valeur donnée dans l'arbre.

3.Résultats

```
Taille de l'arbre : 25
Temps moyen pour l'insertion : 820.717 ns



Temps moyen pour la suppression : 953.321 ns
```

- Temps moyen d'insertion pour 1000 éléments aléatoires : 820.717 ns
- Temps moyen de suppression pour 1000 éléments aléatoires : 953.321 ns

```
Taille de l'arbre : 32
Temps moyen pour l'insertion : 533.707 ns



Temps moyen pour la suppression : 549.4496 ns
```

- Temps moyen d'insertion pour 10000 éléments aléatoires : 533.707 ns
- Temps moyen de suppression pour 10000 éléments aléatoires : 549.449 ns

```
Taille de l'arbre : 45
Temps moyen pour l'insertion : 548.75494 ns



Temps moyen pour la suppression : 670.12334 ns
```

- Temps moyen d'insertion pour 100 000 éléments aléatoires : 549.754 ns
- Temps moyen de suppression pour 100 000 éléments aléatoires : 670.123 ns

Nous pouvons donc constater que l'ajout d'éléments dans l'arbre est légèrement plus rapide que la suppression.

Cela peut être dû au fait que la suppression nécessite plus de traitement pour réorganiser l'arbre.

La hauteur de l'arbre binaire de recherche est relativement élevée, avec une valeur de 25 (pour 1000 éléments), 32 (pour 10000 éléments) et 45 (pour 100000 éléments).

4.Conclusion

En conclusion, cette analyse de l'efficacité pratique des méthodes d'ajout et de suppression dans un arbre binaire de recherche (ABR) nous a permis de constater que ces opérations sont réalisables en un temps raisonnable, avec des temps d'exécution de l'ordre de la milliseconde.

Nous avons pu observer que l'ajout d'éléments est légèrement plus rapide que la suppression, probablement en raison de la nécessité de réorganiser l'arbre lors de la suppression.

Cependant, il est important de prendre en compte la hauteur de l'arbre, qui peut avoir un impact sur les temps d'exécution de certaines opérations.

En somme, cette analyse a permis de mieux comprendre le fonctionnement des arbres binaires de recherche et de leur efficacité pratique dans des scénarios d'insertion et de suppression d'éléments.

Ces résultats pourraient être utilisés pour améliorer les performances de programmes qui utilisent des ABR pour stocker et organiser des données.