INF2010

TP4-Partie 2

Analyse des différences entre les arbres binaires et les monceaux

Introduction

Les arbres binaires et les monceaux (*heaps*) sont des structures de données couramment utilisées dans le domaine de l'informatique pour stocker et organiser des données de manière efficace. Chacun d'eux a des caractéristiques distinctes qui les rendent adaptés à des types spécifiques de problèmes. Dans ce rapport, nous examinerons les différences entre les arbres binaires et les monceaux, en nous concentrant sur les opérations d'insertion et de suppression de données, ainsi que sur les cas d'utilisation spécifiques. Nous allons également présenter le résultat de plusieurs tests effectué sur des arbres binaires, des monceaux minimaux (*min-Heaps*) et des monceaux maximaux (*max-Heaps*). Un min-heap est un monceau dont la racine est la plus petite valeur de l’ensemble et dont chacun de ses sous-arbres est également un min-heap. À l’inverse, un max-heap est un monceau dont la racine est la plus grande valeur de l’ensemble et dont chacun de ses sous-arbres est également un max-heap. Dans cette analyse, les traitements de recherche du minimum n’ont pas été effectué sur les max-heaps et les traitements de recherche du maximum n’ont pas été effectué sur les min-heaps. Les monceaux de cette analyse ont été générés à l’aide de la librairie *java.util.PriorityQueue* et les arbres binaires ont été construits à l’aide de la classe *BinarySearchTree*.

Partie 1 : Résultats des tests

Le tableau 1 présente le temp moyen d’exécution en ns de 10 itérations des traitements de 10 000 insertions en cas normal, de 10 000 insertions en pire cas, de 5000 suppressions en pire cas, de 10 000 recherches du minimum et de 10 000 recherches du maximum pour les structures d’arbre binaire, de min-heap et de max-heap.

**Tableau 1** : Moyennes du temps de 10 traitements de différentes opérations en fonction de la structure

|  |  |  |  |
| --- | --- | --- | --- |
| Traitement | Arbre binaire | Min-Heap | Max-Heap |
| 10 000 insertions– Cas normal [ns] | 2 746 410 | 716 860,2 | 876 300 |
| 10 000 insertions – Pire cas [ns] | 672 893 589,9 | 930 170,1 | 1 313 969,7 |
| 5000 suppressions – Pire cas [ns] | 73 681 099,9 | 2 985 579,9 | 2 634 610,2 |
| 10 000 recherches du minimum [ns] | 720 180,1 | 231 869,9 | - |
| 10 000 recherches du maximum [ns] | 903 680,1 | - | 392 170 |

La figure 1 présente le temps d’exécutions de différents traitements sur un arbre binaire en fonction du nombre d’itérations. Le temp d’exécution est sur une échelle logarithmique. L’insertion et la suppression en pire cas présente les caractéristiques d’une complexité temporelle linéaire O(n) sachant qu’une courbe logarithmique sur une échelle logarithmique indique une progression linéaire. La recherche du minimum (*getmin*) et la recherche du maximum (*getMax*) semble aussi posséder une tendance vers la complexité linéaire et l’insertion en cas normal tend vers une complexité logarithmique O(log(n)).

**Figure 1** : Temps d’exécution de différents traitements sur un arbre binaire de recherche en fonction du nombre d’itération

La Figure 2 présente le temps d’exécutions de différents traitements sur un min-heap en fonction du nombre d’itérations. Tous les traitements démontrent une tendance vers une complexité temporelle logarithmique O(log(n)).

**Figure 2** : Temps d’exécution de différents traitements sur un monceau minimal (*min-Heap*) en fonction du nombre d’itération

La Figure 3 présente le temps d’exécutions de différents traitements sur un max-heap en fonction du nombre d’itérations. Tous les traitements démontrent une tendance vers une complexité temporelle logarithmique O(log(n)).

**Figure 3** : Temps d’exécution de différents traitements sur un monceau maximal (*max-Heap*) en fonction du nombre d’itération

Partie 2 : Différences entre les arbres binaires et les monceaux

Les résultats de la section précédente mettent en évidence plusieurs caractéristiques et différences entre les monceaux et les arbres binaires. Une distinction majeure, clairement représentée dans le tableau 1, est le temps d'exécution notablement plus long pour toutes les opérations dans le cas d'une structure d'arbre binaire. . Bien que la complexité linéaire O(n) dans les pires cas d'insertion et de suppression dans un arbre binaire puisse expliquer en partie cette durée étendue, les figures 1 à 3 révèlent que le temps d'exécution est plus long même pour un nombre plus restreint d'accès aux éléments. Une autre explication réside dans la structure des données sous forme d'objets dans l'arbre binaire, comparée à la structure sous forme de liste dans le cas des monceaux. La structure en liste semble permettre des accès plus rapides aux éléments dans la majorité des cas.

**Différences d’insertions et de suppression de donnée**

Dans le pire cas l’insertion et la suppression de donnée dans un monceau ont une complexité temporelle de O(log(n)). Cette complexité provient du maintien de la propriété du monceau lors des opérations. Les monceaux étant implémentés à l’aide de tableau permettent des accès rapides et une utilisation efficace de la mémoire.

Les arbres binaires standard ont une complexité temporelle dans le pire cas pour les opérations d’insertions et de suppression de O(n) et une complexité temporelle de O(log(n)) pour les cas normaux. La structure des arbres binaires nécessite des pointeurs, des objets et une structure généralement associée à des temps d’accès aux éléments plus long.

**Cas d’utilisation d’un monceau plutôt qu’un arbre binaire**

1. File de priorité (priorité de la file d’attente) : L’utilisation de monceau est plus efficace pour la mise en œuvre d’une file de priorité, où les éléments avec la plus haute ou la plus basse priorité doivent être traités en premier (max-heap, min-heap). Les monceaux permettent des insertions et des extractions de l’élément de priorité maximale en O(log(n)), ce qui est plus efficace que les arbres binaires.
2. Algorithme de Dijkstra : L'algorithme de Dijkstra, utilisé pour trouver les chemins les plus courts dans un graphe pondéré, peut être implémenté de manière efficace en utilisant un monceau pour gérer les nœuds à explorer en fonction de la distance actuelle.

**Comparaison avec un arbre AVL**

La comparaison d’un monceau à un arbre AVL ne présente pas de différences autant significatives en termes de complexité. En effet, pour l’insertion et la suppression l’arbre AVL et le monceau présente tout deux une complexité logarithmique O(log(n)) pour les cas normaux et pour les pires cas. Les monceaux excellent tout de même dans les scénarios de gestion efficace de priorité, alors que les arbres AVL se démarquent lors d’accès fréquents aux mêmes structures. La structure d’un arbre AVL est cependant plus complexe et coûteuse dû à ces caractéristiques de rotation pour le maintien de son équilibre.

Conclusion

L’analyse comparative entre les arbres binaires et les monceaux révèle des différences substantielles dans leurs performances lors d’opérations telles que l'insertion, la suppression, et la recherche du minimum et du maximum. Les résultats des tests démontrent que les monceaux, en particulier les min-heaps et max-heaps, se démarquent par leur efficacité dans la gestion de la priorité, offrant des temps d'exécution plus rapides pour ces opérations.

Deux cas d'utilisation spécifiques où les monceaux surpassent les arbres binaires ont été identifiés : la mise en œuvre d'une file de priorité et l'algorithme de Dijkstra pour trouver les chemins les plus courts dans un graphe pondéré. Ceux-ci mettent en évidence la pertinence des monceaux dans des applications nécessitant des opérations efficaces de priorité.

En comparant les monceaux aux arbres AVL, bien que les différences en termes de complexité soient moins prononcées, chaque structure conserve ses avantages. Les monceaux excellent dans la gestion de priorité, tandis que les arbres AVL se démarquent lors d'accès fréquents aux mêmes structures. Toutefois, la complexité accrue de la structure des arbres AVL, liée à leurs caractéristiques de rotation pour le maintien de l'équilibre, doit être prise en compte dans le choix de la structure de données en fonction des exigences.