Analyse de Complexité : BST vs AVL

Cas des insertions séquentielles (IDs triés)

Auteurs : **HALLOUMI Idris**
AHAMADA Naheri

Janvier 2026

1 Introduction

Dans le cadre du projet ChargeCraft, l'indexation des stations de recharge est critique. Les identifiants de stations sont dans la plupart des cas des entiers séquentiels (ex : 101, 102, 103...). Ce rapport analyse pourquoi un Arbre Binaire de Recherche standard (BST) est inadapté pour ces données et justifie l'utilisation d'un arbre AVL.

2 Définition et Principe de l'AVL

Un arbre **AVL** (nommé d'après ses inventeurs *Adelson-Velsky* et *Landis* en 1962) est un arbre binaire de recherche **auto-équilibré**. Il s'agit historiquement de la première structure de données de ce type.

2.1 Invariant de Structure

La propriété fondamentale qui distingue l'AVL d'un BST classique est son critère d'équilibre strict. Pour **chaque nœud** de l'arbre, la différence de hauteur entre son sous-arbre gauche et son sous-arbre droit (appelée *facteur d'équilibre*) ne doit jamais excéder 1 en valeur absolue.

Mathématiquement, pour tout nœud N, si h_g est la hauteur du sous-arbre gauche et h_d celle du droit :

$$|h_g - h_d| \leq 1$$

Le facteur d'équilibre peut donc prendre uniquement les valeurs -1, 0, ou 1.

2.2 Mécanisme de Rotations

Lorsqu'une insertion ou une suppression viole cet invariant (le facteur devient ± 2), l'AVL effectue des opérations locales appelées **rotations** (simples ou doubles) pour rétablir l'équilibre. Ce mécanisme garantit que la hauteur de l'arbre reste toujours bornée par $O(\log n)$, assurant ainsi des performances optimales même dans les pires scénarios d'insertion.

3 Le Problème des Insertions Triées

3.1 Comportement du BST (Binary Search Tree)

Un BST ne possède pas de mécanisme d'auto-équilibrage. Si l'on insère des données déjà triées (croissantes ou décroissantes), chaque nouveau nœud est ajouté systématiquement à droite (ou à gauche). L'arbre "dégénère" alors en une **liste chaînée**.

Exemple : Insertion de 1, 2, 3, 4, 5.

```
1
 \
  2
   \
    3
     ...
```

Dans ce cas pire (*Worst Case*), la hauteur h devient égale au nombre de nœuds n. La complexité de recherche passe de $O(\log n)$ à $\mathbf{O(n)}$.

3.2 Comportement de l'AVL face au tri

Grâce à sa propriété d'équilibre définie en Section 2, l'AVL réagit immédiatement aux insertions séquentielles (1, 2, 3...). Dès qu'un déséquilibre est détecté, il effectue une rotation gauche (pour des insertions croissantes) pour remonter les nœuds médians vers la racine.

Résultat : Même avec des IDs strictement triés en entrée, l'arbre reste compact et touffu, avec une hauteur maximale $h \approx 1.44 \log_2 n$.

4 Comparaison des Complexités

Le tableau ci-dessous résume les coûts temporels pour n opérations sur des données triées :

Opération	**BST (Cas Pire)**	**AVL (Garanti)**
Hauteur de l'arbre (h)	n	$\log n$
Insertion d'un élément	$O(n)$	$O(\log n)$
Recherche d'un élément	$O(n)$	$O(\log n)$
Insertion de n éléments	$O(n^2)$	$O(n \log n)$

TABLE 1 – Comparaison sur flux d'entrée trié

5 Analyse Expérimentale (Scénario ChargeCraft)

Pour un réseau de $N = 1000$ stations insérées séquentiellement :

— **BST :** Il faut parcourir 1 nœud, puis 2, ..., puis 1000.

$$\text{Coût total} \approx \frac{n(n+1)}{2} \approx 500,000 \text{ comparaisons.}$$

— **AVL :** La hauteur maximale est ≈ 10.

$$\text{Coût total} \approx n \times 10 = 10,000 \text{ comparaisons.}$$

Le BST est donc **50** fois plus lent à construire dans ce cas précis, et chaque recherche ultérieure sur l'ID 1000 coûtera 1000 opérations contre 10 pour l'AVL.

6 Conclusion

L'utilisation de l'AVL est impérative pour ChargeCraft. Elle garantit la robustesse du système face à des jeux de données réels (souvent partiellement triés) et assure que le temps de réponse du système reste stable ($O(\log n)$) quelle que soit la méthode d'insertion des stations.