

Programmation fonctionnelle

enslIE

Semestre 3 – 2026/27

Arbres bien équilibrés

Pour rendre les ABR efficaces, il faut minimiser la hauteur par rapport au nombre de nœuds

Définition

*Un arbre est **bien équilibré** si :*

- ▶ *la différence de hauteur entre ses deux sous-arbres est d'au plus 1*
- ▶ *chacun de ses sous-arbres est bien équilibré*

Hauteur d'un arbre bien équilibré

Théorème

Pour un arbre bien équilibré à n nœuds et de hauteur h

$$\log_2(1 + n) \leq h \leq \alpha \log_2(2 + n)$$

avec $\alpha = \frac{1}{\log_2\left(\frac{1+\sqrt{5}}{2}\right)} < 1,44$

Par conséquent, l'appartenance, l'ajout et la suppression sont en $O(\log n)$ dans les arbres bien équilibrés

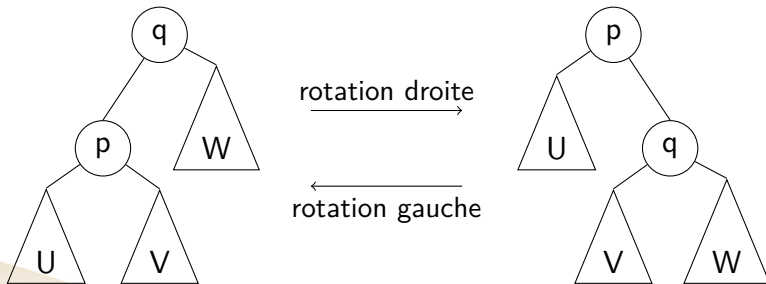
Arbres auto-équilibrants

Problème : l'insertion et la suppression peuvent déséquilibrer un arbre bien équilibré

- Modifier les fonctions d'ajout et de suppression pour garantir un invariant de bon équilibre

Arbres AVL (Adelson-Velskii et Landis, 1962) : rééquilibrage par rotations

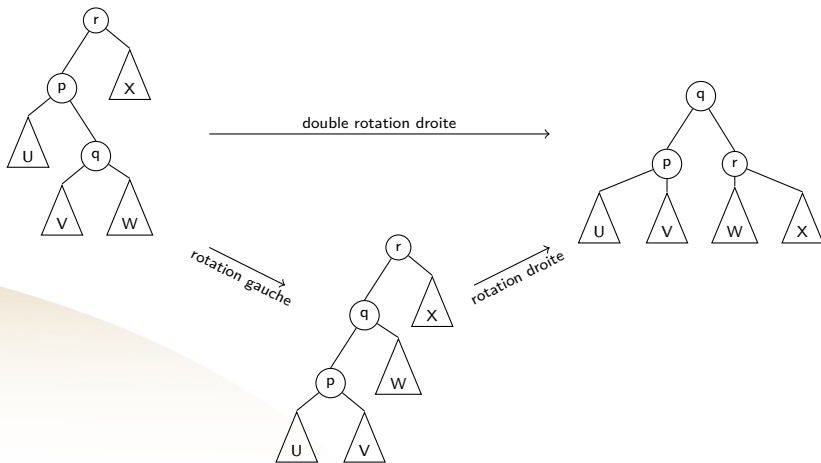
Rotations



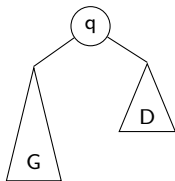
Les rotations préservent l'invariant des ABR

Elles peuvent être réalisées en temps constant

Double rotations



Rééquilibrage

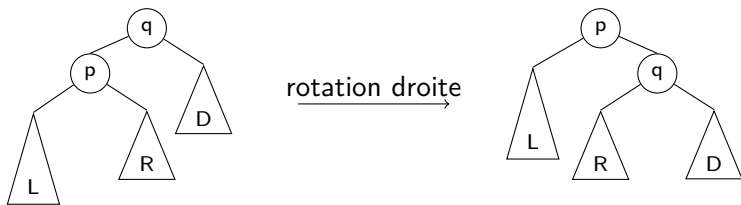


Une insertion ou une suppression perturbe d'au plus 1 les hauteurs des sous-arbres

Par conséquent la différence de hauteur entre deux sous-arbres est d'au plus 2 si l'arbre était bien équilibré avant

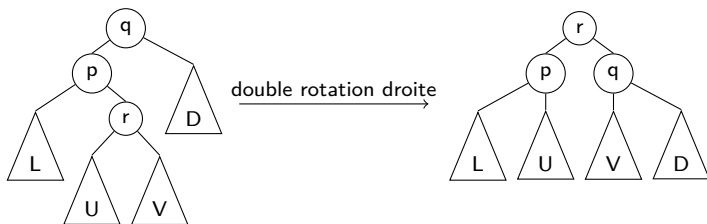
On suppose $h(G) = h(D) + 2$ (l'autre cas par symétrie)

Rééquilibrage (suite)



- Si $h(L) \geq h(R)$: rotation droite

Rééquilibrage (suite)



- ▶ Si $h(L) \geq h(R)$: rotation droite
- ▶ Si $h(L) < h(R)$: double rotation droite

Hauteur

Pour savoir quelles rotations effectuer, il faut connaître la hauteur

- ▶ stockée dans les nœuds

Utilisation de constructeurs intelligents : la hauteur est calculée pendant la création du nœud et

On appelle les constructeurs intelligents au lieu des anciens constructeurs pour créer un nœud

Calcul de hauteur en temps constant

Encapsulation

On veut ne manipuler que des arbres bien équilibrés

- ▶ Il faut interdire la création d'arbres non équilibrés à l'extérieur du module

On utilise l'encapsulation liée à la modularité : la seule façon de construire des AVL sera via le constructeur intelligent `ba1`

Intérêt : dans `ba1` on peut supposer que les arguments sont bien équilibrés, puisqu'ils proviennent des appels aux constructeurs intelligents

Également : `join` dans le cas où la différence de hauteur est plus de deux

Complexité

bal : $\mathcal{O}(1)$

join l _ r : $\mathcal{O}(|h_l - h_r|)$

- ▶ vide : $\mathcal{O}(1)$
- ▶ appartenance : $\mathcal{O}(\log(n))$
- ▶ ajout : $\mathcal{O}(\log(n))$
- ▶ cardinal : $\mathcal{O}(n)$
- ▶ union : $\mathcal{O}(n_1 \times \log(n_2))$
- ▶ intersection : $\mathcal{O}(n_1 \times \log(n_2))$
- ▶ retrait : parcours $\mathcal{O}(\log(n))$
- ▶ agrégation $\mathcal{O}(n)$, map $\mathcal{O}(n^2)$
- ▶ égalité $\mathcal{O}(n \log(n))$