14 Arbres équilibrés

14.1 Arbre AVL [Adelson-Velsky et Landis (1962)]

W(en)

Définition 14.1. *Un arbre binaire est un arbre AVL ssi à chaque nœud interne, la hauteur du sous-arbre gauche et la hauteur du sous-arbre droit diffèrent par 1 au plus.*

Pour vérifier l'équilibre AVL, on peut stocker la hauteur de chaque sous-arbre à sa racine. Dans une implantation il suffit d'indiquer le cas applicable correspondant à une des trois relations possibles $<$, $=$, $>$ entre les hauteurs des sous-arbres. Lors d'une insertion (ou suppression), il faut rétablir l'équilibre en performant $O(1)$ (ou $O(\log n)$) rotations.

Lemme 14.1. *Soit $N(h)$ le nombre minimal de nœuds internes dans un arbre AVL de hauteur $h \geq 0$. On a $N(0) = 0$, $N(1) = 1$. Pour tout $h > 1$, $N(h) = N(h-1)+N(h-2)+1$. En conséquence, $N(h) = F_{h+2}-1$, où F_k est le k-ème nombre Fibonacci.*

Exercice 14.1. *Dessinez les arbres AVL à hauteur maximale et minimale pour 7, 12 et 20 nœuds internes.*

Théorème 14.2. *La hauteur h d'un arbre AVL avec n nœuds [non-null] est bornée comme*

$$\lg(n+1) \leq h \leq \log_\phi(n+1) = \frac{\lg(n+1)}{\lg \phi} \approx 1.44 \lg n. \tag{14.1}$$

Démonstration. La borne inférieure correspond à l'arbre binaire complet avec n nœuds internes dont la hauteur est $h = \lceil \lg(n+1) \rceil$. Par Lemme 14.1,

$$N(h) + 1 = F_{h+2} = \frac{\phi^{h+2} - (\phi-1)^{h+2}}{\sqrt{5}} = \big(1 - o(1)\big) \cdot \frac{\phi+1}{\sqrt{5}} \phi^h \approx 1.17\phi^h,$$

où on a utilisé que $F_k = (\phi^k - (\phi-1)^k)/\sqrt{5}$, et que $\phi^2 = \phi + 1$, $-1 < \phi - 1 < 0$. ■

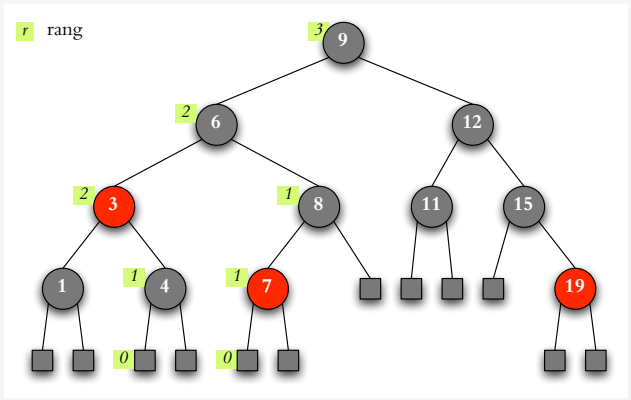
14.2 Arbre rouge et noir

W(en)

On associe une valeur entière non-négative à chaque nœud. On va l'appeler le «rang» (avant de trouver un meilleur nom), dénoté par $\mathsf{rang}(x)$. Le rang est croissant vers la racine, comme la hauteur, mais avec un équilibre permissif entre deux sous-arbres frères, parce que parfois le rang du père est le même que celui de l'enfant. Le but est de contrôler le déséquilibre (trop de rangs identiques mènent à une plus grande hauteur possible), mais pas excessivement (il faut ajuster la structure en $O(\log n)$ temps au pire à chaque opération). On va démontrer que les règles à côté accomplissent exactement ce but.

Règles.

1. Pour chaque nœud x excepté la racine,
$$\mathsf{rang}(x) \leq \mathsf{rang}(\mathsf{parent}(x)) \leq \mathsf{rang}(x) + 1.$$
2. Pour chaque nœud x avec un grand-parent
$$\mathsf{rang}(x) < \mathsf{rang}\big(\mathsf{parent}(\mathsf{parent}(x))\big).$$
3. Pour chaque nœud externe (null) on a
$$\mathsf{rang}(x) = 0 \qquad \mathsf{rang}(\mathsf{parent}(x)) = 1.$$



Au lieu de stocker les rangs explicitement, il suffit de stocker la différence entre parent et enfant, en coloriant les nœuds par rouge ou noir.

- ⋆ si $\mathsf{rang}(\mathsf{parent}(x)) = \mathsf{rang}(x)$, alors x est colorié par **rouge**
- ⋆ si x est la racine ou $\mathsf{rang}(\mathsf{parent}(x)) = \mathsf{rang}(x)+1$, alors x est colorié par **noir**

Théorème 14.3. *Dans un coloriage valide,*
(0) chaque nœud est soit noir soit rouge
*(i) chaque nœud externe (*null*) est noire*
(ii) le parent d'un nœud rouge est noir
(iii) tout chemin d'un nœud x à un nœud externe dans son sous-arbre contient le même nombre de nœuds noirs

Démonstration de Théorème 14.3. Propriété (0) ⇔ Règle 1, (i) ⇔ Règle 3, (ii) ⇔ Règle 2. En (iii), le nombre de nœuds noirs égale $\mathsf{rang}(x)$, en justifiant l'appellation **hauteur noire**. ■

Lemme 14.4. *Pour chaque nœud x, sa hauteur $h(x) \leq 2 \cdot \mathsf{rang}(x)$.*

Démonstration. Sur un chemin jusqu'à un nœud externe, il y a au moins autant de nœuds noirs que des rouges. ■

Lemme 14.5. *Le nombre de déscendants internes de chaque nœud x est $\geq 2^{\mathsf{rang}(x)} - 1$.*

Démonstration. Par induction [dans la hauteur de x]. Le théorème est vrai pour un nœud externe x quand $\mathsf{rang}(x) = 0$. Supposons que le théorème est vrai pour tout x avec une hauteur $h(x) < k$. Considérons un nœud x avec $h(x) = k$ et ses deux enfants u, v avec $h(u), h(v) < k$. Par l'hypothèse d'induction, le nombre des descendants de x est $\geq 1+(2^{\mathsf{rang}(u)}-1)+(2^{\mathsf{rang}(v)}-1) = 2^{\mathsf{rang}(u)}+2^{\mathsf{rang}(v)}-1$. Or, $\mathsf{rang}(x)-1 \leq \mathsf{rang}(u), \mathsf{rang}(v)$, ou $2^{\mathsf{rang}(u)}+2^{\mathsf{rang}(v)} \geq 2^{\mathsf{rang}(x)}$. Le théorème reste donc vrai pour x avec $h(x) = k$, et, en conséquence pour tout x. ■

Théorème 14.6. *La hauteur d'un arbre RN avec n nœuds internes est bornée comme*

$$\lg(n+1) \leq h \leq 2\lg(n+1). \tag{14.2}$$

Démonstration. La borne inférieure correspond à l'arbre binaire complet. La borne supérieure vient de 14.4 et 14.5, appliqués à la racine :

$$\begin{aligned} 2^{\mathsf{rang(racine)}} - 1 &\leq n && \text{par 14.5} \\ \mathsf{rang(racine)} &\leq \lg(n+1) \\ h/2 &\leq \lg(n+1) && \text{par 14.4} \end{aligned}$$

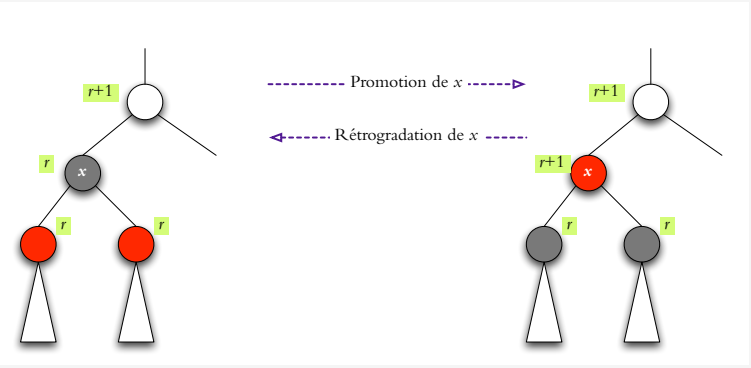
■

14.3 Arbre RN : temps de calcul des opérations

On ajuste la structure lors d'une insertion ou suppression de noeud en parcourant un chemin vers la racine, en $O(h)$ temps pour un arbre de hauteur h. Par Théorème 14.6, la hauteur d'un arbre RN est toujours $h = \Theta(\log n)$, donc toutes les opérations s'exécutent en $O(\log n)$, même dans le pire cas. Le théorème montre aussi que pour un grand n, la recherche est au pire seulement deux fois plus lente que possible avec l'arbre idéal.

Usage de mémoire : il suffit de stocker la couleur (1 bit) de chaque nœud interne. On peut même épargner ce bit : on échange les pointeurs gauche $\leftrightarrow$ droit pour les nœuds noirs — la couleur est retrouvée par la comparaison des clés aux enfants ou en examinant la couleur du parent (si enfants nulls). Il faut altérer le code de recherche de noeuds en descendant de la racine aussi pour toujours examiner si c'est le sous-arbre gauche ou droit qui contient les clés inférieures. En général, le mémoire récuperé ne vaut pas le prix du ralentissement causé par les complications inévitables dans le code.

14.4 Ajustement de la structure



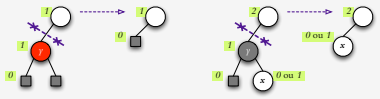
Pour maintenir l'équilibre, on utilise les **rotations** comme avant + **promotion/rétrogradation** (incrémenter ou décrementer le rang par 1).
$\rightarrow$ promotion/rétrogradation change la couleur d'un nœud et ses enfants.
$\rightarrow$ on ne peut promouvoir x que s'il est noir avec deux enfants rouges (pour ne pas violer Règle 1 et Propriété (ii) de Théorème 14.3)

14.5 Insertion dans l'arbre RN

On insère x avec $\mathsf{rang}(x) = 1 \Rightarrow$ sa couleur est rouge.

Test : est-ce que le parent de x est rouge ? Si oui, on a un problème ; sinon, rien à faire (cas 0a). Solution : examiner le grand-parent $y = \mathsf{parent}(\mathsf{parent}(x))$: il est forcément noir par Propriété (ii) de Théorème 14.3. Selon le coloriage des enfants de y (le parent et l'oncle de x), on fait une série de promotions (cas 0b : oncle rouge), suivie par une ou deux rotations (cas 1 ou 2 : oncle noir), comme les zig-zag, zag-zig, etc., en déploiement.

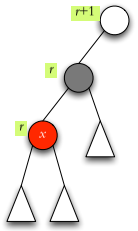
14.6 Suppression dans l'arbre RN



Pour la suppression, on utilise une technique similaire : procéder comme avec l'arbre binaire de recherche, puis retrogradations en ascendant vers la racine + $O(1)$ rotations (trois au plus) à la fin.
L'ajustement de la structure départ par l'examination du nœud y physiquement enlevé (donc, c'est le successeur ou prédecesseur si on a supprimé la clé d'un nœud avec deux enfants non-null). Le nœud y est remplacé par x qui est un des enfants de y. Selon la différence de rangs entre x et y (donc couleur de x), on fait une série de rétrogradations, et quelques rotations.

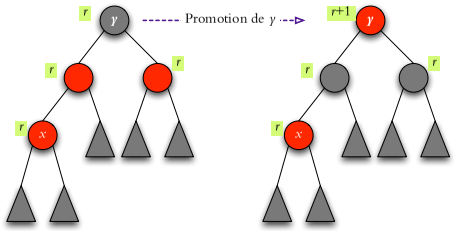
Insertion

Cas 0a : parent noir



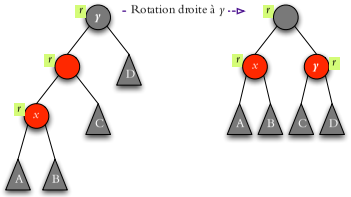
rien à faire

Cas 0b : parent rouge, oncle rouge



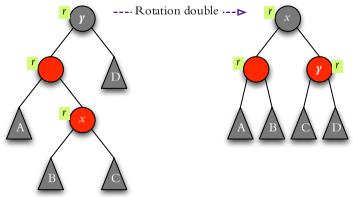
promotion du grandparent, continuer avec $x \leftarrow y$

Cas 1 : parent rouge, oncle noir, zig-zig



une rotation simple

Cas 2 : parent rouge, oncle noir, zig-zag

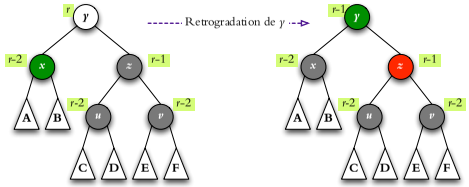


une rotation double

Suppression

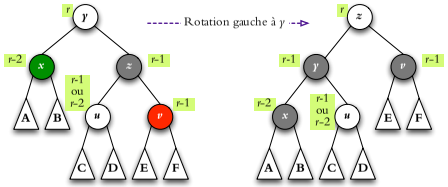
Cas 0 : nœud rouge x — il devient noir, et on arrête

Cas 1a : sœur noire, neveux noirs



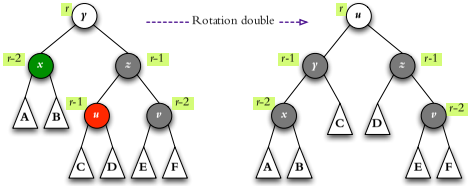
retrogradation du parent, continuer avec $x \leftarrow y$ en cas 0, 1, ou 2

Cas 1b : sœur noire, neveu distant rouge



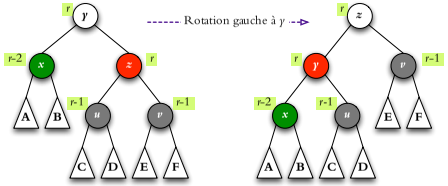
une rotation simple

Cas 1c : sœur noire, neveu proche rouge



une rotation double

Cas 2 : sœur rouge



une rotation simple, continuer avec x en cas 1 (pas de récursion en 1a)