Arbres RN (cont)

Thm. L'hauteur dans un arbre RN : pour chaque nœud x,
sa hauteur $h(x) \leq 2 \cdot \mathsf{rang}(x)$.

Preuve. On doit avoir au moins autant de nœuds noirs que des nœuds rouges dans un chemin de x à une feuille. $\square$

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

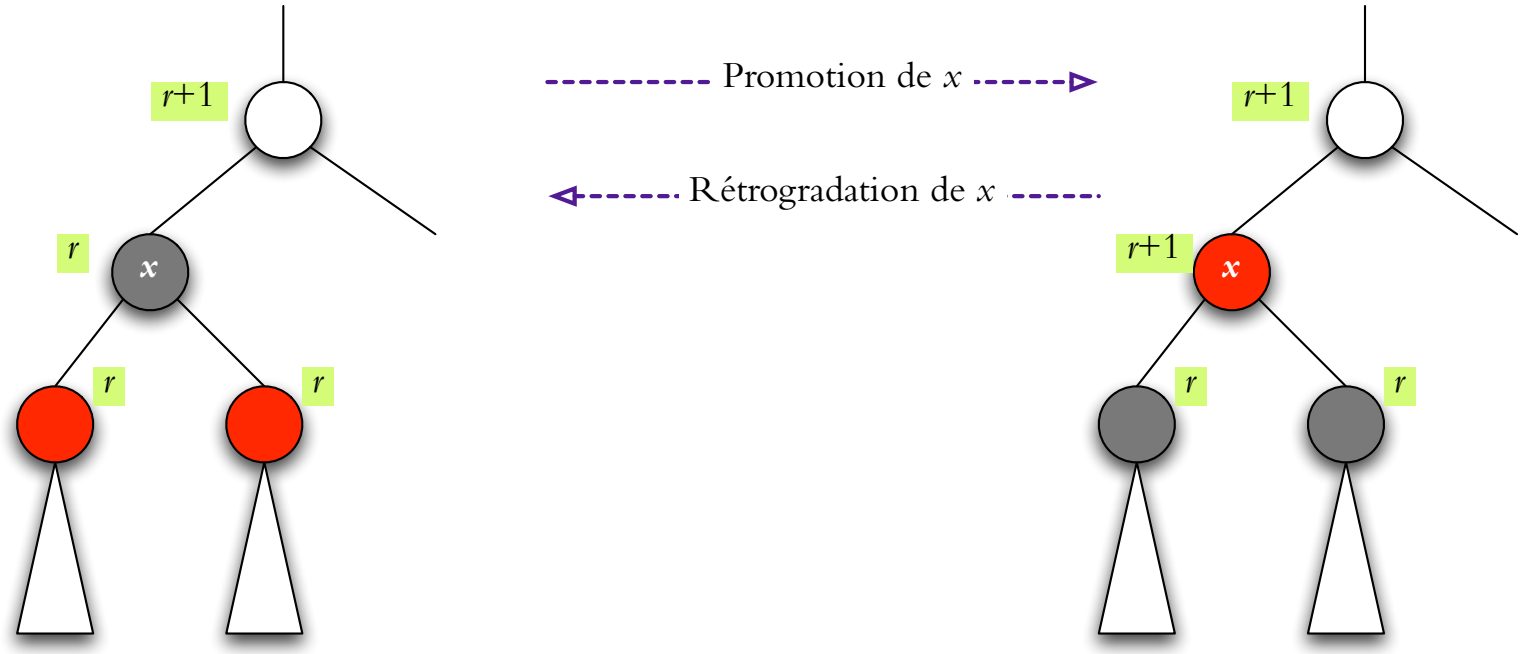
Preuve. Par induction. Le théorème est vrai pour une feuille x quand $\mathsf{rang}(x) = 0$. Supposons que le théorème est vrai pour tout x avec une hauteur $h(x) < k$. Considérons un nœud x avec $h(x) = k$ et ses deux enfants u, v avec $h(u), h(v) < k$. Par l'hypothèse d'induction, le nombre des descendants de x est $\geq 1 + (2^{\mathsf{rang}(u)} - 1) + (2^{\mathsf{rang}(v)} - 1)$. Or, $\mathsf{rang}(x) - 1 \leq \mathsf{rang}(u), \mathsf{rang}(v)$. $\square$

Arbres RN (cont)

Thm. Un arbre RN avec n nœuds internes a une hauteur $\leq 2\lfloor \lg(n+1) \rfloor$.

Arbres RN — balance

Pour maintenir la balance, on utilise les **rotations** comme avant
+ **promotion/rétrogradation** : incrémenter ou décrementer le rang



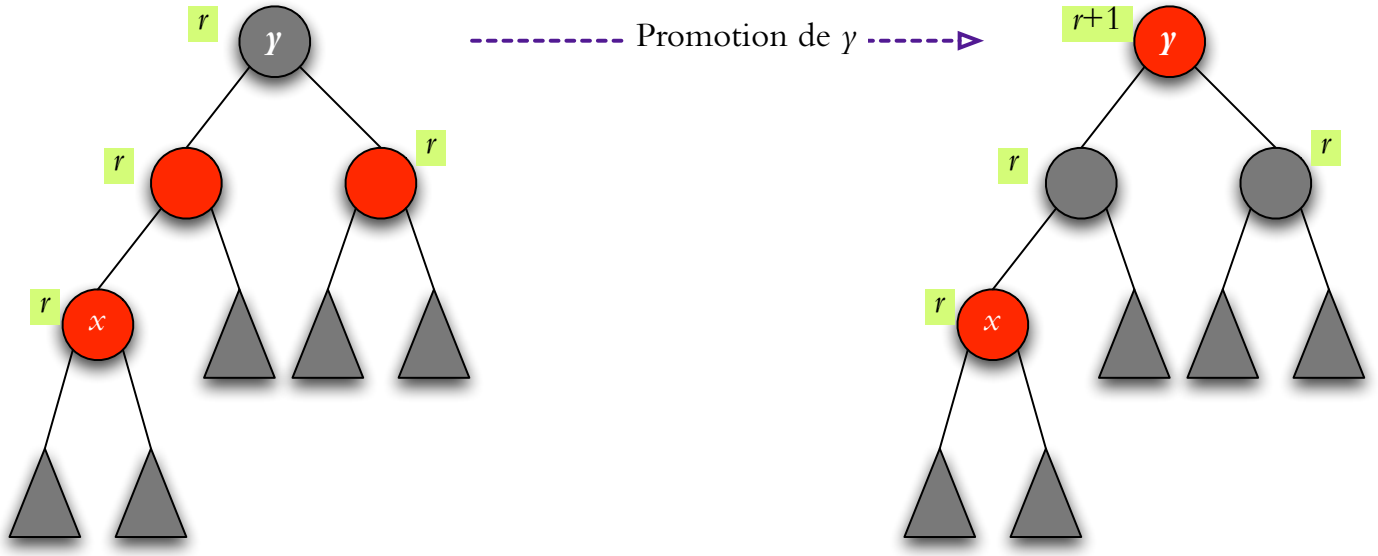
→ promotion/rétrogradation change la couleur d'un nœud et ses enfants
on peut promouvoir x ssi il est noir avec deux enfants rouges

Arbres RN — insertion

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

Solution : soit $y = \mathsf{parent}(\mathsf{parent}(x))$ le grand-parent — il est noir. Si y a deux enfants rouge, alors promouvoir y et retourner au test avec $x \leftarrow y$.



Arbres RN — insertion (cont)

On a fini les promotions et il y a toujours le problème que x est rouge, son parent est rouge aussi, mais l'oncle de x est noir.

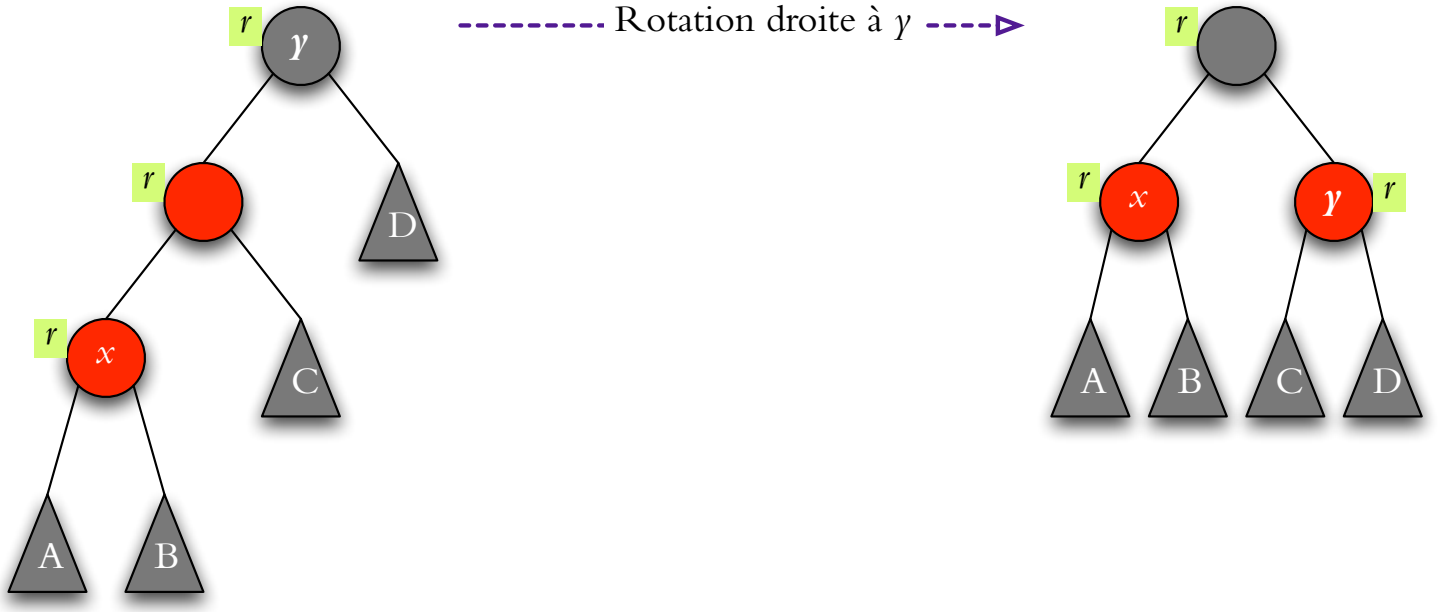
Deux cas :
Cas 1 quand x et $\mathsf{parent}(x)$ sont au même côté (enfants gauches ou enfants droits) — une rotation suffit

Cas 2 quand x et $\mathsf{parent}(x)$ ne sont pas au même côté (l'un est un enfant gauche et l'autre un enfant droit) — rotation double est nécessaire

(enfin, c'est quatre cas : 1a, 1b, 2a et 2b)

Arbres RN — insertion (cont)

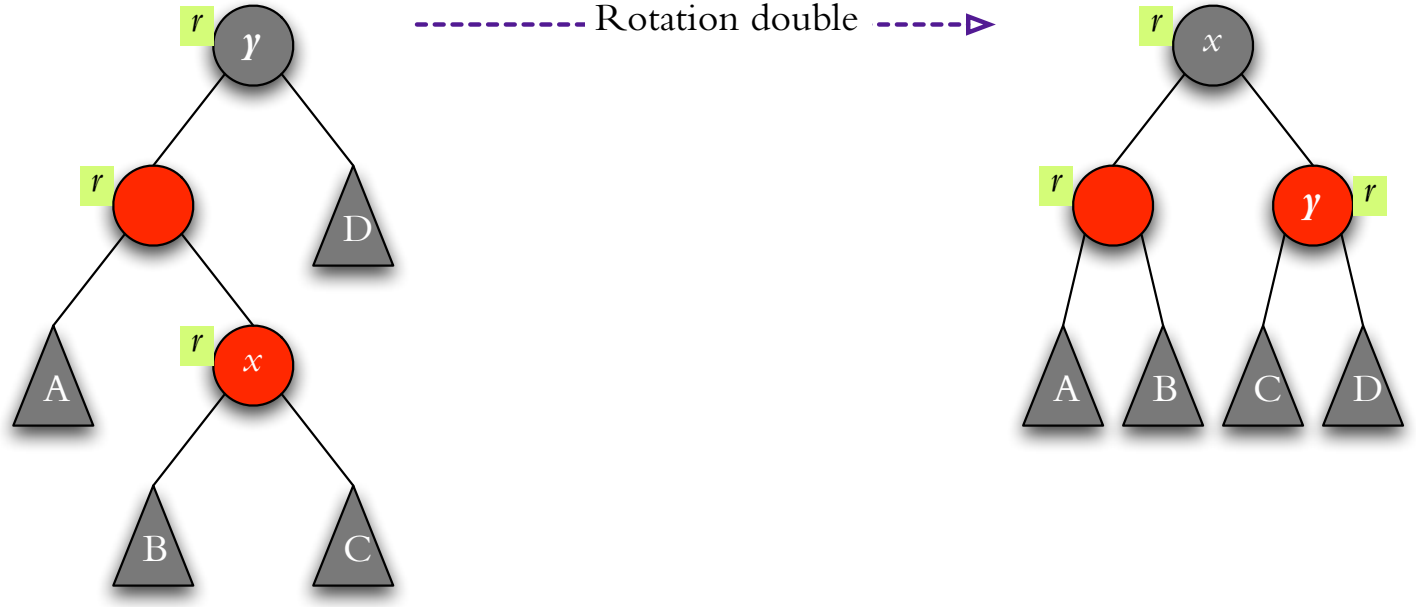
Cas 1a : x est rouge, son parent est rouge, son oncle est noir, et x et $\mathsf{parent}(x)$ sont des enfant gauches



Cas 1b (enfants droits) est symétrique

Arbres RN — insertion (cont)

Cas 2a : x est rouge, son parent est rouge, son oncle est noir, x est un enfant droit et $\mathsf{parent}(x)$ est un enfant gauche



Cas 2b (x est gauche et $\mathsf{parent}(x)$ est droit) est symétrique

Arbres RN — suppression

Et suppression d'un nœud ?

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

Arbres RN — efficacité

Un arbre rouge et noir avec n nœuds internes et hauteur $h \in O(\log n)$.

Recherche : $O(h)$ mais $h \in O(\log n)$ donc $O(\log n)$

Insertion :
1. $O(h)$ pour trouver le placement du nouveau nœud
2. $O(1)$ pour initialiser les pointeurs
3. $O(h)$ promotions en ascendant si nécessaire
4. $O(1)$ pour une rotation simple ou double si nécessaire

$O(h)$ en total mais $h \in O(\log n)$ dons $O(\log n)$

Suppression : $O(\log n)$

Usage de mémoire : il suffit de stocker la couleur (1 bit) de chaque nœud interne