Arbres *splay*

On utilise souvent des variables auxiliares pour maintenir l'équilibre de l'arbre
p.e., arbre rouge et noir : au moins un bit (couleur)

Arbre *splay* : aucune variable

mais $O(\log n)$ seulement comme coût amorti

Arbres *splay* (cont)

Idée principale : rotations sans tests spécifiques pour l'équilibre

Quand on accède à nœud x, on performe des rotations sur le chemin de la racine à x pour monter x à la racine.

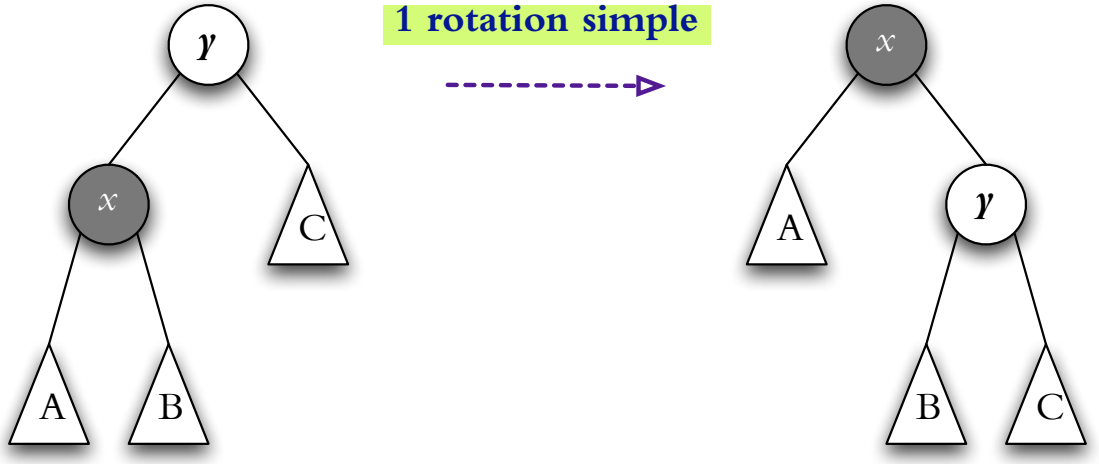
Déploiement (*splaying*) du nœud x : étapes successives jusqu'à ce que $\mathsf{parent}(x)$ devienne null
(et donc x devient la racine de l'arbre)

Zig et zag

Trois cas majeurs pour une étape de déploiement :

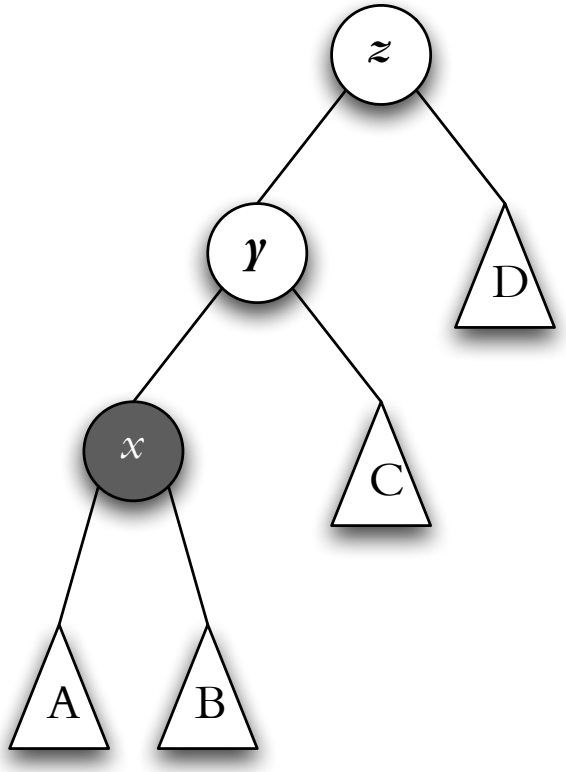
1. x sans grand-parent (zig ou zag)
2. x et $\mathsf{parent}(x)$ au même coté (gauche-gauche ou droit-droit : zig-zig ou zag-zag)
3. x et $\mathsf{parent}(x)$ à des côtés différents (gauche-droit ou droit-gauche : zig-zag ou zag-zig)

Cas 1: *zig*

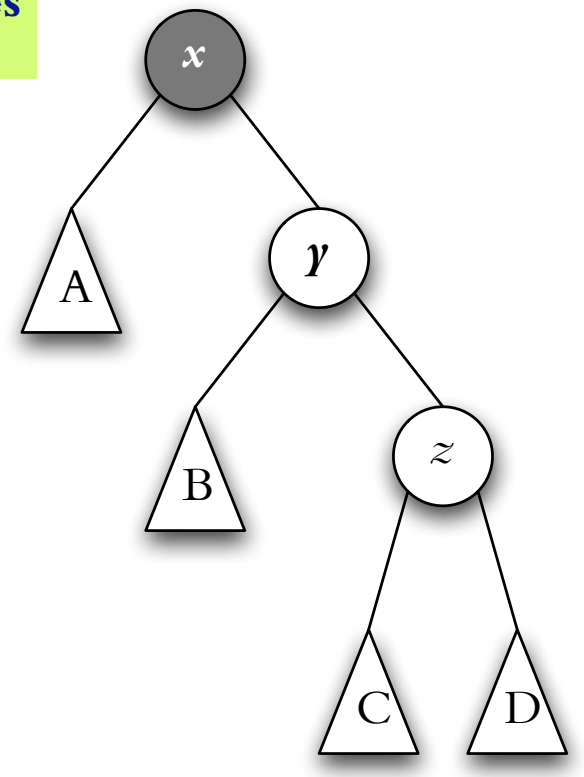


Zig et zag (cont)

Cas 2: *zig-zig*

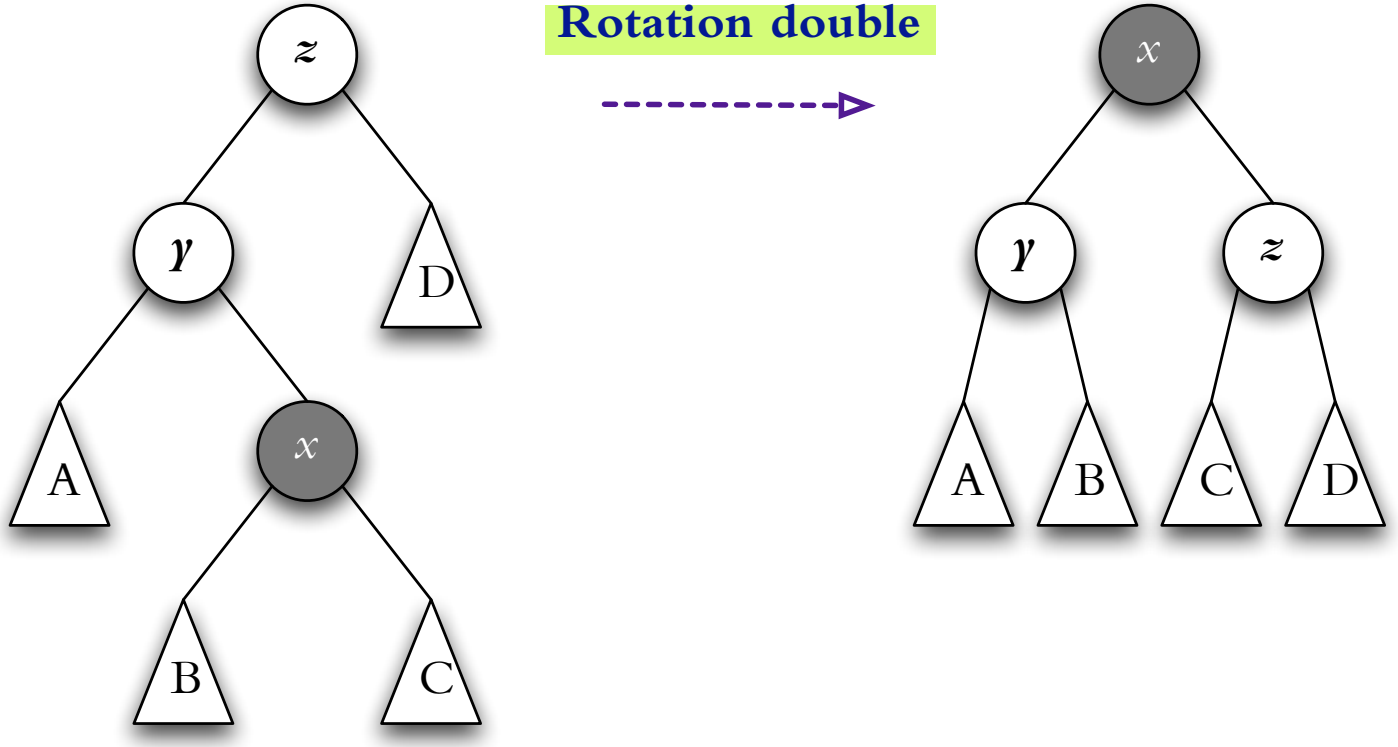


2 rotations simples
(à z et à y)



Zig et zag (cont)

Cas 3: *zig-zag*



Déploiement

Choix de x pour déploiement :

- insert : x est le nouveau nœud
- search : x est le nœud où on arrive à la fin de la recherche
- delete : x est le parent du nœud *effectivement* supprimé
  attention : c'est le parent ancien du successeur (ou prédecesseur) si on doit supprimer un nœud à deux enfants
  (logique : échange de nœuds, suivi par la suppression du nœud sans enfant)

Coût amorti

Temps moyen dans une **série** d'opérations

«moyen» ici : temps total divisé par nombre d'opérations
(aucune probabilité)

Théorème. Le temps pour exécuter une série de m opérations (search, insert et delete) en commençant avec l'arbre vide est de $O(m \log n)$ où n est le nombre d'opérations d'insert dans la série.

$\longrightarrow$ il peut arriver que l'exécution est très rapide au début et tout d'un coup une opération prend très long. . .

$\longrightarrow$ tout à fait acceptable si utilisé dans un algorithme

Arbres rouges et noirs

Idée : une valeur entière non-négative, appellée le *rang*, associée à chaque nœud.
Notation : $\mathsf{rang}(x)$.

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 grand-parent $y = \mathsf{parent}(\mathsf{parent}(x))$,

$$\mathsf{rang}(x) < \mathsf{rang}(y).$$

3. Pour chaque feuille (null) on a $\mathsf{rang}(x) = 0$ et $\mathsf{rang}(\mathsf{parent}(x)) = 1$.

Arbres RN (cont)

D'où vient la couleur ?

Les nœuds peuvent être coloriés par rouge ou noir.

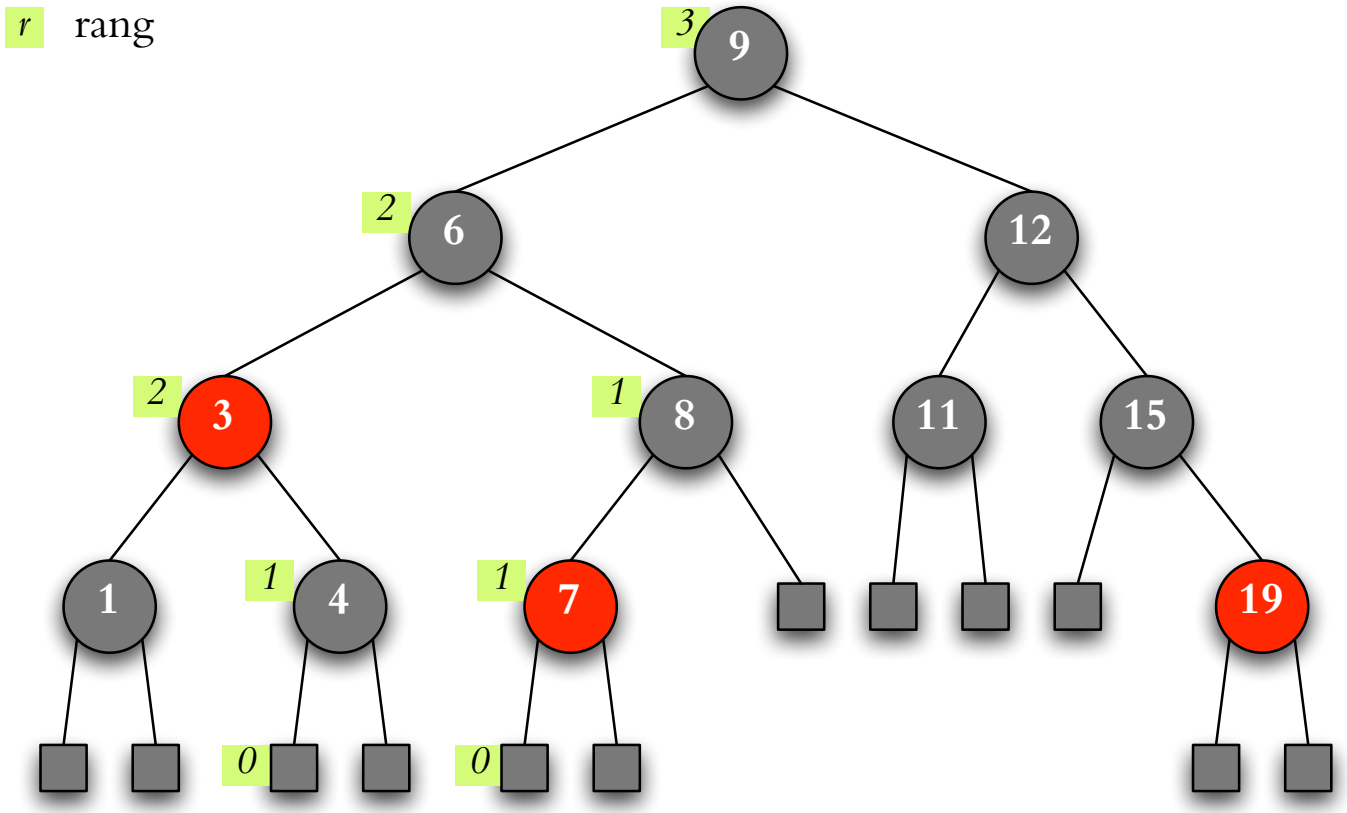
- si $\mathsf{rang}(\mathsf{parent}(x)) = \mathsf{rang}(x)$, alors x est colorié par rouge
- si x est la racine ou $\mathsf{rang}(\mathsf{parent}(x)) = \mathsf{rang}(x) + 1$, alors x est colorié par noir

Thm. Coloriage :
(0) chaque nœud est soit noir soit rouge
(i) chaque feuille (null) est noire
(ii) le parent d'un nœud rouge est noir
(iii) chaque chemin reliant un nœud à une feuille dans son sous-arbre contient le même nombre de nœuds noirs

Preuve En (iii), le nombre de nœuds noirs sur le chemin est égal au rang. □

⇒ rang est parfois appelé «hauteur noire»

Arbres RN (cont)



Arbres RN (cont)

Thm. La 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. □

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

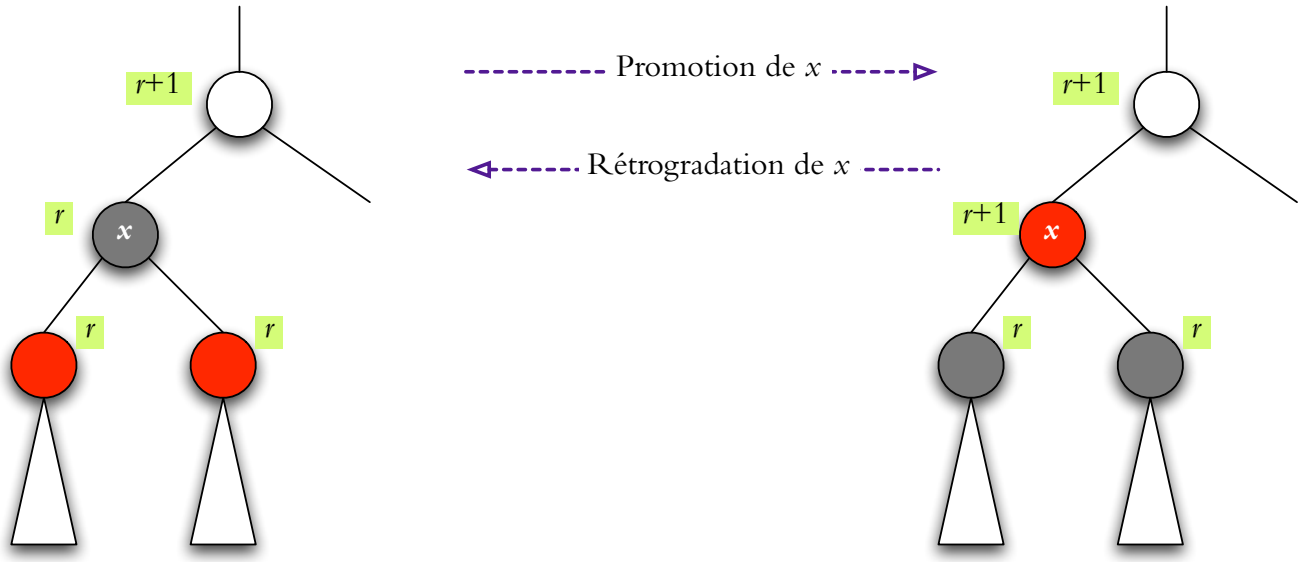
Preuve. Par induction. Le théorème est vrai pour une feuille x quand $\mathsf{rang}(x) = 0$. Supposons que le théorème est vrai pour tout x avec une hauteur $h(x) < k$. Considérons un nœud x avec $h(x) = k$ et ses deux enfants u, v avec $h(u), h(v) < k$. Par l'hypothèse d'induction, le nombre des descendants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

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



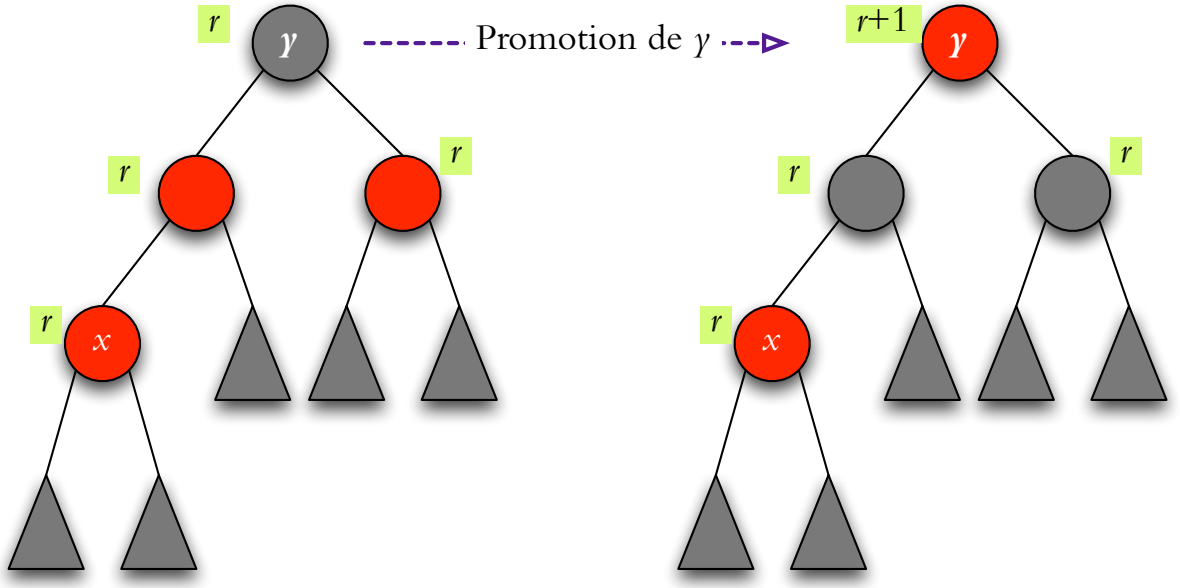
→ promotion/rétrogradation change la couleur d'un nœud et ses enfants
on peut promouvoir x ssi il est noir avec deux enfants rouges

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.

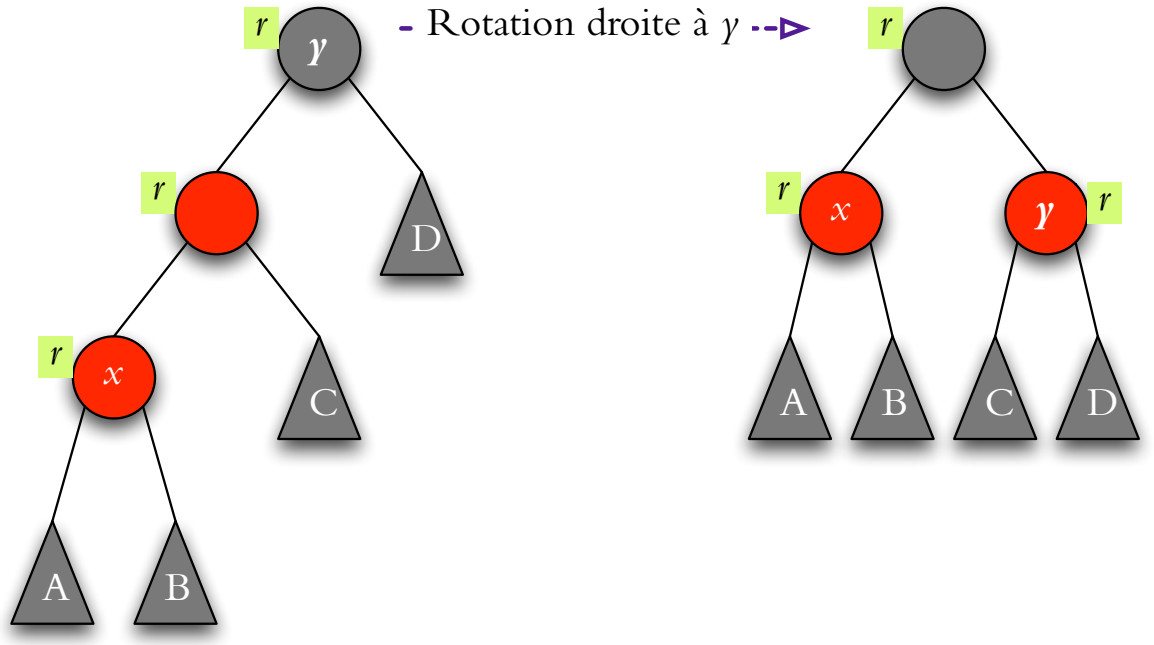
Deux cas :
Cas 1 quand x et $\mathsf{parent}(x)$ sont au même côté (enfants gauches ou enfants droits) — une rotation suffit

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)

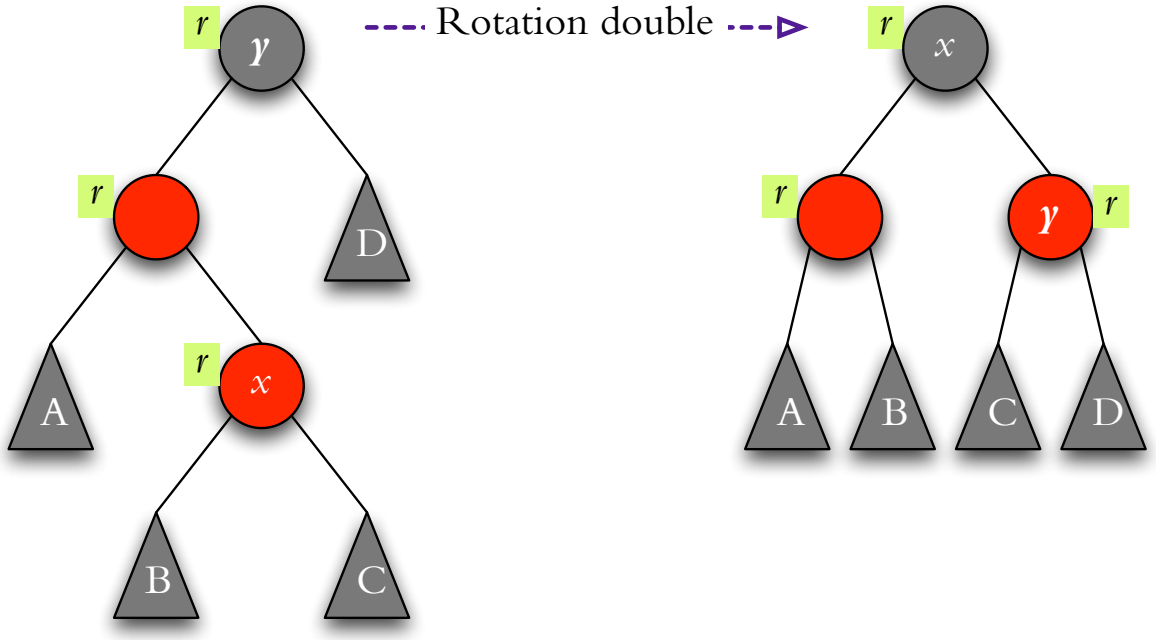
Cas 1a : x est rouge, son parent est rouge, son oncle est noir, et x et $\mathsf{parent}(x)$ sont des enfant gauches



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