Arbres rouges et noirs

Olivier Carton

Les *arbres rouges et noirs* sont une variante des arbres binaires de recherche. Leur intérêt principal est qu'ils sont relativement équilibrés. On considère ici des arbres binaires de recherche où les valeurs sont portées par les nœuds internes. Les feuilles ne sont pas prises en compte dans la hauteur de l'arbre.

Définition

Un *arbre rouge et noir* est un arbre binaire de recherche ou chaque nœud est de couleur rouge ou noire de telle sorte que

1. les feuilles sont noires,
2. les fils d'un nœud rouge sont noirs,
3. le nombre de nœuds noirs le long d'une branche de la racine à une feuille est indépendant de la branche.

La première condition est simplement technique et sans grande importance. La seconde condition stipule que les nœuds rouges ne sont pas trop nombreux. La dernière condition est une condition d'équilibre. Elle signifie que s'il on oublie les nœuds rouges d'un arbre on obtient un arbre binaire parfaitement équilibré.

Dans un arbre rouge et noir, on peut toujours supposer que la racine est noire. Si elle est rouge, on change sa couleur en noire et toutes les propriétés restent vérifiées.

Hauteur

Un arbre binaire complet de hauteur h possède au plus $1 + 2 + \ldots + 2^h = 2^{h+1}-1$ nœuds internes. Autrement dit, un arbre ayant n nœuds internes est de hauteur au moins égale à $\ln(n+1)-1$. La hauteur minimale d'un arbre à n nœuds internes est atteinte lorsque l'arbre est parfaitement équilibré et que feuilles sont toutes sur un ou deux niveaux.

$\ln(n+1)-1 \leq h$.

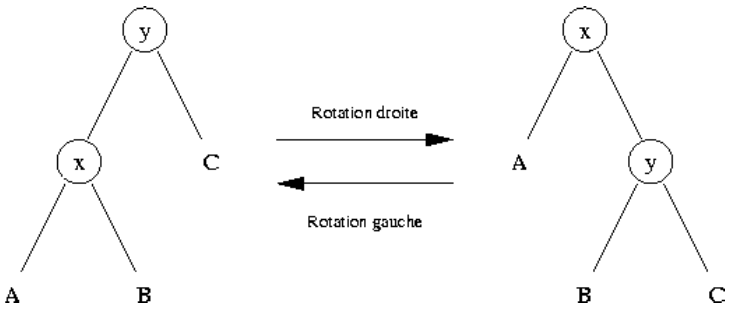
Les arbres rouges et noirs sont relativement bien équilibrés. La hauteur d'un arbre rouge et noir est de l'ordre de grandeur de $\ln(n)$ où n est le nombre d'éléments dans l'arbre. En effet, la hauteur h un arbre rouge et noir ayant n éléments vérifie l'inégalité

$h \leq 2\ln(n+1)$.

Pour un arbre rouge et noir, on appelle *hauteur noire* le nombre k de nœuds internes noirs le long d'une branche de la racine à une feuille. On montre par récurrence sur cette hauteur noire, qu'un arbre de hauteur noire égale à k possède au moins 2^k-1 nœuds internes. Si cette hauteur noire vaut 0, l'arbre est réduit à une seule feuille et il ne possède aucun nœud interne. Si un arbre est de hauteur noire égale à $k > 0$, alors ses sous-arbres gauche et droit sont au moins de hauteur noire k-1. Par hypothèse de récurrence, ils ont chacun au moins $2^{k-1}-1$ nœuds internes et l'arbre a au moins $(2^{k-1}-1)+(2^{k-1}-1)+1 = 2^k-1$ nœuds internes au total. Ceci montre que la hauteur noire k d'un arbre vérifie $k \leq \ln(n+1)$. Puisque la racine peut être supposée noire et qu'une branche ne contient pas deux nœuds rouges consécutifs, la hauteur h de l'arbre vérifie $h \leq 2k$. Elle vérifie donc finalement $h \leq 2\ln(n+1)$.

Rotations

Les rotations sont des modifications locales d'un arbre binaire. Elles consistent à échanger un nœud avec l'un de ses fils. Dans la rotation droite, un nœud devient le fils droit du nœud qui était son fils gauche. Dans la rotation gauche, un nœud devient le fils gauche du nœud qui était son fils droit. Les rotations gauche et droite sont inverses l'une de l'autre. Elle sont illustrées à la figure ci-dessous. Dans les figures, les lettres majuscules comme A, B et C désignent des sous-arbres.



Insertion d'une valeur

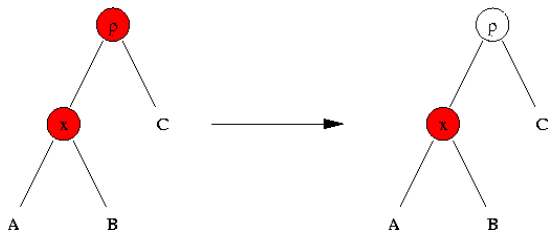
L'insertion d'une valeur dans un arbre rouge et noire commence par l'insertion usuelle d'une valeur dans un arbre binaire de recherche. Le nouveau nœud devient rouge de telle sorte que la propriété (3) reste vérifiée. Par contre, la propriété (2) n'est plus nécessairement vérifiée. Si le père du nouveau nœud est également rouge, la propriété (2) est violée.

Afin de rétablir la propriété (2), l'algorithme effectue des modifications dans l'arbre à l'aide de rotations. Ces modifications ont pour but de rééquilibrer l'arbre.

Soit x le nœud et p son père qui sont tous les deux rouges. L'algorithme distingue plusieurs cas.

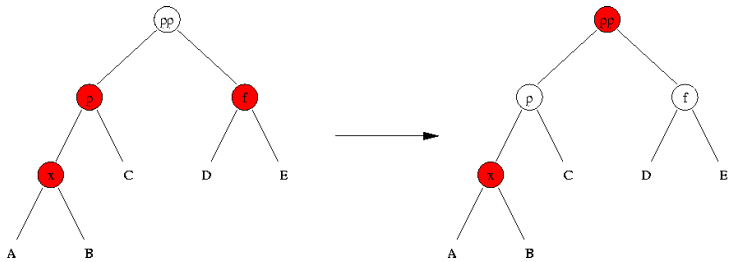
Cas 0 : le nœud père p est la racine de l'arbre

Le nœud père devient alors noir. La propriété (2) est maintenant vérifiée et la propriété (3) le reste. C'est le seul cas où la hauteur noire de l'arbre augmente.



Cas 1 : le frère f de p est rouge

Les nœuds p et f deviennent noirs et leur père pp devient rouge. La propriété (3) reste vérifiée mais la propriété ne l'est pas nécessairement. Si le père de pp est aussi rouge. Par contre, l'emplacement des deux nœuds rouges consécutifs s'est déplacé vers la racine.



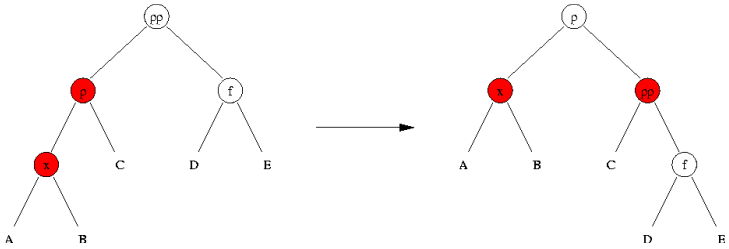
Cas 2 : le frère f de p est noir

Par symétrie on suppose que p est le fils gauche de son père. L'algorithme distingue à nouveau deux cas suivant que x est le fils gauche ou le fils droit de p.

Cas 2a : x est le fils gauche de p.

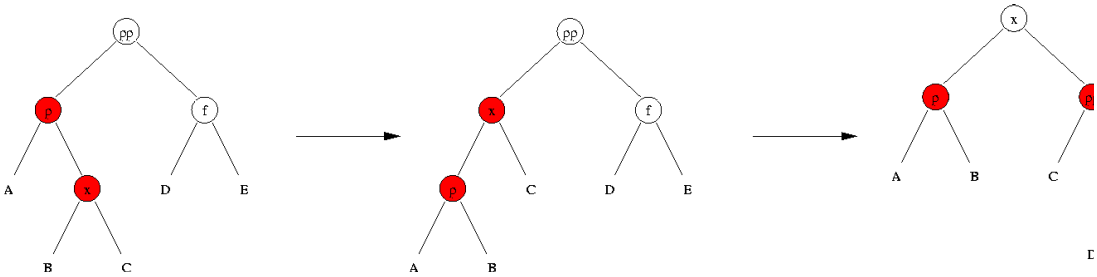
L'algorithme effectue une rotation droite entre p et pp. Ensuite le nœud p devient noir et le nœud pp devient rouge. L'algorithme s'arrête alors puisque les propriétés

(2) et (3) sont maintenant vérifiées.



Cas 2b : x est le fils droit de p.

L'algorithme effectue une rotation gauche entre x et p de sorte que p deviennent le fils gauche de x. On est ramené au cas précédent et l'algorithme effectue une rotation droite entre x et pp. Ensuite le nœud x devient noir et le nœud pp devient rouge. L'algorithme s'arrête alors puisque les propriétés (2) et (3) sont maintenant vérifiées.



Suppression d'une valeur

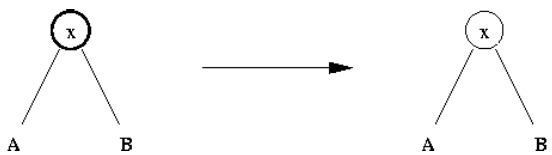
Comme pour l'insertion d'une valeur, la suppression d'une valeur dans un arbre rouge et noir commence par supprimer un nœud comme dans un arbre binaire de recherche. Si le nœud qui porte la valeur à supprimer a zéro ou un fils, c'est ce nœud qui supprimer et son éventuel fils prend sa place. Si au contraire, ce nœud a deux fils, il n'est pas supprimé. La valeur qu'il porte est remplacée par la valeur suivante dans l'ordre et c'est le nœud qui portait cette valeur suivante qui est supprimé. Ce nœud supprimé est le nœud au bout de la branche gauche du sous-arbre droit du nœud qui portait la valeur à supprimer. Ce nœud n'a pas de fils gauche.

Si le nœud supprimé est rouge, la propriété (3) reste vérifiée. Si le nœud supprimé est noir, on considère que le nœud qui remplace le nœud supprimé porte une couleur noire en plus. Ceci signifie qu'il devient noir s'il est rouge et qu'il devient doublement noir s'il est déjà noir. La propriété (3) reste ainsi vérifié mais il y a éventuellement un nœud qui est doublement noir.

Afin de supprimer ce nœud doublement noir, l'algorithme effectue des modifications dans l'arbre à l'aide de rotation. Soit x le nœud doublement noir.

Cas 0 : le nœud x est la racine de l'arbre

Le nœud x devient simplement noir. La propriété (2) est maintenant vérifiée et la propriété (3) le reste. C'est le seul cas où la hauteur noire de l'arbre diminue.



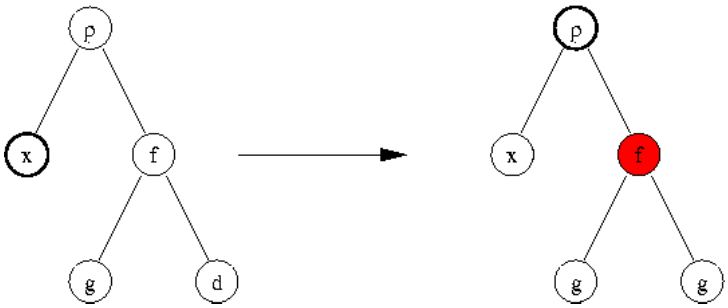
Cas 1 : le frère f de x est noir.

Par symétrie, on suppose que x est le fils gauche de son père p et que f est donc le fils droit de p. Soient g et d les fils gauche et droit de f. L'algorithme distingue à nouveau trois cas suivant les couleurs des nœuds g et d.

Cas 1a : les deux fils g et d de f sont noirs.

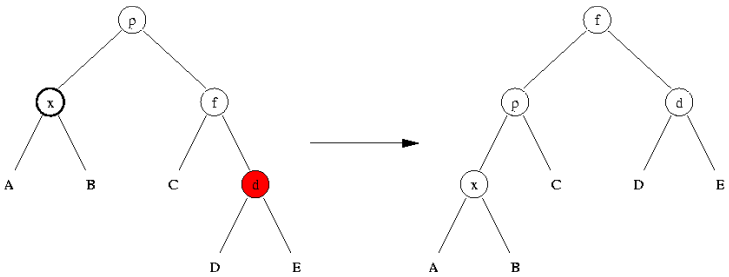
Le nœud x devient noir et le nœud f devient rouge. Le nœud p porte une couleur noire en plus. Il devient noir s'il est rouge et il devient doublement noir s'il est déjà

noir. Dans ce dernier cas, il reste encore un nœud doublement noir mais il s'est déplacé vers la racine de l'arbre. C'est ce dernier cas qui représenté à la figure ci-dessous.



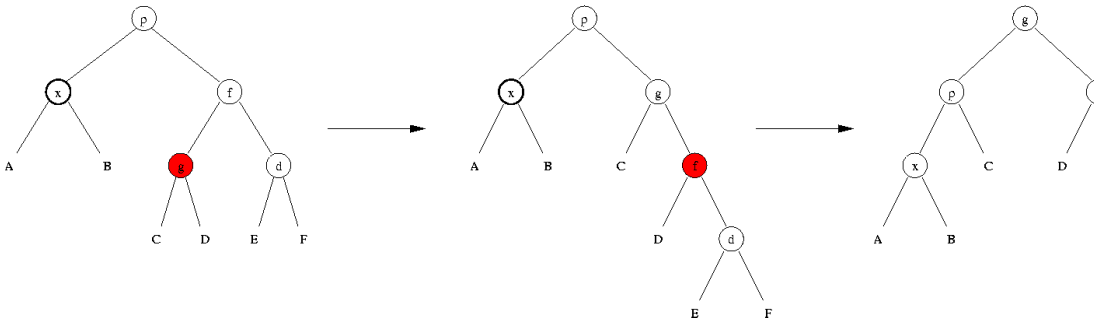
Cas 1b : le fils droit d de f est rouge.

L'algorithme effectue une rotation gauche entre p et f. Le nœud f prend la couleur du nœud p. Les nœuds x, p et d deviennent noirs et l'algorithme se termine.



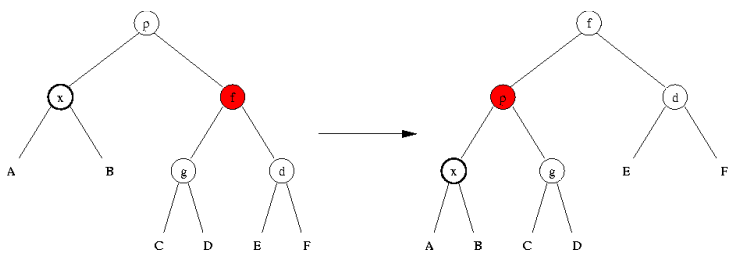
Cas 1c : le fils droit d est noir et le fils gauche g est rouge.

L'algorithme effectue une rotation droite entre f et g. Le nœud g devient noir et le nœud f devient rouge. Il n'y a pas deux nœuds rouges consécutifs puisque la racine du sous-arbre D est noire. On est ramené au cas précédent puisque maintenant, le frère de x est g qui est noir et les deux fils de g sont noir et rouge. L'algorithme effectue alors une rotation entre p et g. Le nœud f redevient noir et l'algorithme se termine.



Cas 2 : le frère f de x est rouge.

Par symétrie, on suppose que x est le fils gauche de son père p et que f est donc le fils droit de p. Puisque f est rouge, le père p de f ainsi que ses deux fils g et d sont noirs. L'algorithme effectue alors une rotation gauche entre p et f. Ensuite p devient rouge et f devient noir. Le nœud x reste doublement noir mais son frère est maintenant le nœud g qui est noir. On est donc ramené au cas 1.



Implémentation des arbres rouges et noirs

On présente ici une implémentation simpliste des arbres rouges et noirs. Comme les manipulations sur les arbres rouges et noirs utilisent comme les arbres AVL des rotations, l'implémentation tire partie de la programmation objet pour *factoriser* l'implémentation de ces deux types d'arbres. Les classes données ci-dessous se répartissent en trois catégories : les classes spécifiques aux arbres rouges et noirs, les classes spécifiques aux arbres AVL et les classes implémentant des parties communes.

- Classes communes
  Interface `SearchTree`
  : Cette interface définit les fonctionnalités d'un arbre de recherche : ajout et recherche d'une clé ainsi que les affichages.

  Classe abstraite `BinaryTree`
  : Implémentation par un arbre binaire d'un arbre de recherche. Les fonctionnalités non spéficiques aux arbres rouges et noirs ou aux AVL sont implémentés dans cette classe. C'est le cas des calculs de la hauteur et de la taille ainsi que des affichages.

  Classe `SimpleBinaryTree`
  : Implémentation d'un arbre binaire de recherche sans équilibrage.

  Classe `TestSearchTree`
  : Classe de tests.

  Interface `Node`

  Classe `InternalNode`
  : Nœud interne qui porte une valeur.

  Classe `EmptyNode`
  : Feuille qui ne porte pas de valeur.

  Classe abstraite `BalancedInternalNode`
  : Cette classe abstraite implémente les rotations gauche et droite.
- Classes spécifiques aux arbres rouges et noirs
  Classe `RbTree`
  : Arbre rouge et noir

  Interface `RbNode`

  Classe `RbInternalNode`
  : Nœud interne d'un arbre rouge et noir.

  Classe `RbEmptyNode`
  : Feuille d'un arbre rouge et noir.
- Classes spécifiques aux arbres AVL
  Classe `AvlTree`
  : Arbre AVL

  Classe `AvlInternalNode`
  : Nœud interne d'un arbre AVL.

  Classe `AvlEmptyNode`
  : Feuille d'un arbre AVL.