Chapître IV.
Arbres rouge et noir

1. Arbres binaires de recherche

Les arbres binaires de recherche (ou arbres binaires ordonnés, abrev. ABO) sont des structures de données pouvant supporter nombre d'opérations sur les ensembles dynamiques (qui sont ordonnés totalement):

$RECHERCHER$, $INSERER$, $SUPPRIMER$
$MINIMUM$, $MAXIMUM$
$PREDECESSEUR$, $SUCCESSEUR$

Les clés d'un arbre binaire de recherche sont toujours stockées de manière à satisfaire la **propriété des arbres binaires de recherche**:

> Soit x un nœud d'un arbre binaire de recherche. Si y est un nœud du sous-arbre gauche de x, alors $cle[y] \leq cle[x]$. Si y est un nœud du sous-arbre droit de x, alors $cle[x] \leq cle[y]$.

Théorème. *Les opérations de base des ensembles dynamiques comme $RECHERCHER$, $INSERER$, $SUPPRIMER$, $MINIMUM$, $MAXIMUM$, $PREDECESSEUR$ et $SUCCESSEUR$ peuvent être réalisées dans un temps en $O(h)$ sur un arbre binaire de recherche de hauteur h.*

2. Arbres binaires de recherche équilibrés

Les opérations ensemblistes sont donc d'autant plus rapides que la hauteur de l'arbre binaire est petite ($O(\log n)$ ou $\Theta(\log n)$);
mais si cette hauteur est grande ($\Omega(n)$ ou $\Theta(n)$), leur performance peut ne pas être meilleure que pour une liste chaînée.

$\Rightarrow$ L'idée d'un arbre binaire de recherche de l'hauteur $O(\log n)$ se trouve souvent dans les publications:

-) arbres AVL (Adelson-Velski, Landis, 1962)
-) arbres 2 - 3 (Hopcroft, 1970)
-) arbre rouge et noir (Bayer, 1972)

-) arbres déployés (Sleator, Tarjan, 1983)

On a déjà considéré les arbres AVL et leur opérations, en particulier les rotations.

Dans un certain sens les arbres AVL sont les arbres binaires de recherche les plus équilibrés possible. Pour obtenir d'autres arbres binaires de recherche équilibrés il faut remplacer la définition d'arbres AVL par la suivante:

Définition. Une classe des arbres binaires de recherche est approximativement équilibrée si la hauteur d'un arbre est $O(\log n)$ pour chaque arbre comportant n nœuds (et qui permet ainsi aux opérations d'ensembles dynamiques de s'exécuter en temps $O(\log n)$ dans le pire des cas.)

3. Arbres rouge et noir

Un arbre rouge et noir est un arbre de recherche comportant un bit de stockage supplémentaire par nœud: sa couleur, qui peut valoir

soit $ROUGE$, soit $NOIR$.

Chaque nœud de l'arbre contient les champs $couleur$, cle, $gauche$ (fg), $droit$ (fd) et p.

Si un fils ou le père d'un nœud n'existe pas, le champs correspondant contient la valeur NIL.

Convention importante:

Nous considérons ces NIL comme des pointeurs sur des nœuds externes (feuilles) de l'arbre binaire de recherche, et les nœuds comportant des clés comme des nœuds internes de l'arbre.

Définition. Un arbre binaire de recherche est un **arbre rouge et noir** s'il satisfait les propriétés suivantes:

1. Chaque nœud est soit rouge, soit noir.
2. Chaque feuille (NIL) est noire.
3. Si un nœud est rouge, alors ses deux fils sont noirs.
4. Chaque chemin simple reliant un nœud à une feuille descendante contient le même nombre de nœuds noirs.

4. La hauteur d'un arbre rouge et noir

Les propriétés 3 et 4 d'un arbre rouge et noir garantissent qu'aucun chemin (simple) allant de la racine à une feuille n'est plus de deux fois plus long que n'importe quel autre, ce qui rend l'arbre approximativement équilibré.

Définition. On appèlle hauteur noire d'un nœud x, notée $hn(x)$, le nombre de nœuds noirs présents dans un chemin quelconque descendant du nœud x, nœud x non compris, à une feuille.

D'après la propriété 4, la notion de hauteur noire est bien définie.

Le lemme suivant montre pourquoi les arbres rouge et noir sont des bons arbres binaires de recherche.

Lemme. *Un arbre rouge et noir comportant* n *nœuds internes a une hauteur au plus égale à* $2 \cdot \log(n+1)$.

Démonstration: *Commençons par montrer l'affirmation que le sous-arbre enraciné à un nœud* x *quelconque contient au moins* $2^{hn(x)}-1$ *nœuds internes.*

Finalement on peut obtenir le lemme comme une conséquence de l'affirmation et de la propriété 3.

Appelons h la hauteur de l'arbre. D'après la propriété 3, au moins la moitié des nœuds d'un chemin simple reliant la racine r à une feuille, r non comprise, doivent être noirs.

$\Rightarrow \quad hn(r) \geq h/2$

Affirmation pour $r \Rightarrow \quad n \geq 2^{h/2} - 1$

Prenons le logarithme (des deux membres), on obtient $\log(n+1) \geq h/2$, ou $h \leq 2\log(n+1)$.

$\square$

Affirmation: Le sous-arbre enraciné à un nœud x quelconque contient au moins $2^{hn(x)}-1$ nœuds internes.

Cette affirmation peut se démontrer par induction sur la hauteur de x.

Si $h(x) = 0$ alors x est une feuille et le sous-arbre enraciné à x contient au moins $2^{hnx}-1 = 2^0 - 1 = 0$ nœds internes.

Pour l'étape inductive, on considère un nœud interne x de hauteur positive. Alors x a deux fils.

Chaque fils a une hauteur noire $hn(x) - 1$ ou $hn(x)$ (selon que sa couleur est rouge ou noire).

La hauteur d'un fils de x est inférieur à la hauteur de x.

L'hypothèse d'induction $\Rightarrow$
Le sous-arbre enraciné à un fils de x possède au moins $2^{hn(x)-1} - 1$ nœuds internes.

Donc, le sous-arbre enraciné à x possède au moins $(2^{hn(x)-1} - 1) + (2^{hn(x)-1} - 1) + 1 = 2^{hn(x)} - 1$ nœuds.

Ce qui démontre l'assertion.

Conséquence immédiate de ce lemme:

Tous les opérations de requête qui ne change pas l'arbre, comme $RECHERCHER$, $MAXIMUM$, $MINIMUM$, $PREDECESSEUR$ et $SUCCESSEUR$, peuvent être implémentées dans un temps en $O(\log n)$ sur les arbres rouge et noir,

-) puisqu'elles peuvent s'exécuter en $O(h)$ sur un arbre binaire de recherche de hauteur h et

-) puisqu'un arbre rouge et noir quelconque à n nœuds et un arbre de recherche de hauteur $O(\log n)$, en effet de hauteur au plus de $2\log(n+1)$.

Attention:

$INSERER$ et $SUPPRIMER$ ne sont pas directement supportées, car elles ne garantissent pas que l'arbre binaire de recherche modifié par elles restera un arbre rouge et noir.

5. Rotations

Les opérations $ARBRE{-}INSERER$ et $ARBRE{-}SUPPRIMER$ qui on utilise par un arbre binaires de recherche détruisent la structure d'un arbre rouge et noir d'une manière déjà connue des arbres AVL. En plus il y a un problème nouveau: il faut préserver les propriété 3 et 4 d'un arbre rouge et noir.

Les outils les plus importants ne change pas comparé à l'arbre AVL. Ces sont des **rotations**.

Une rotation est une opération locale dans un arbre de recherche, préservant l'ordre infixe des clés. Les deux procedures $ROTATION{-}GAUCHE$ et $ROTATION{-}DROIT$ s'exécutent en $O(1)$. En plus, les pointeurs sont les seuls champs du nœud modifiés par une rotation.

6. L'opération $INSERER$

Considérons l'insertion d'un nœud dans un arbre rouge et noir.

Etape 1: On utilise la procédure $ARBRE-INSERER$ pour insérer le nœud x dans l'arbre T comme si c'était un arbre binaire de recherche ordinaire.

Etape 2: On colorie le nœud x en rouge.

[parce que cette coloration préserve la propriété 4]

Etape 3: Malheureusement le nouveau nœud rouge a peut-être un père rouge, c'est-à-dire la propriété 3 est violée. On corrige ensuite l'arbre en recoloriant les nœuds et en effectuant des rotations.

Qu'est-ce qu'il faut faire à l'étape 3?

On peut constater que l'unique propriété, qui est peut-être violée après la coloration du nœud x, est la propriété 3.

Nous observerons que cela est un invariant de procédure $RN - INSERER$.

Il faut considérer six cas mais trois d'entre eux sont symétriques avec les trois autres. On se trouve toujours dans la situation où le nœud x et le père $p[x]$ sont rouges.

Cas 1: L'oncle y de x, c'est-à-dire $y = droit[p[p[x]]]$, est rouge. Alors $p[x]$ et y sont tous les deux rouges.

On colorie $p[x]$ et y noir. Pour conserver la propriété 4 il faut qu'on colorie $p[p[x]]$ qui était noir en rouge.

Peut-être propriété 3 est maintenant violée pour $p[p[x]]$ et son père, si le père est rouge. Alors on appelle x le nœud $p[p[x]]$ et on répète.

Constatons qu'on a progressé du nœud actuel à son grandpère.

Cas 2 et 3: L'oncle $y = droit[p[p[x]]]$ de x est noir.

Dans ces cas on utilise des rotations et recoloration.

Les deux cas se distinguent selon que x est un fils droit ou gauche de $p[x]$.

On transforme le cas 2 en cas 3 par une seule $ROTATION-GAUCHE(T,x)$.

Dans cas 3, on colorie $p[x]$ noir et $p[p[x]]$ rouge. Après on utilise une seule $ROTATION-DROITE(T,p[p[x]])$.

A cause de cette rotation droite $p[p[x]]$ ne reste pas le fils de son père $p[p[p[x]]]$. Le nouveau fils de $p[p[p[x]]]$, qui est peut-être rouge, devient $p[x]$ qui est certainement noir.

Analyse de temps

La hauteur d'un arbre rouge et noir à n nœuds étant en $O(\log n)$.

L'appel à $ARBRE-INSERER$ prend $O(\log n)$.

On répète uniquement dans le cas 1. Dans cas 2 ou 3 on arrête après au plus deux rotations.

Dans le cas 1 le nœud courant passe toujours à grandpère. Alors cette boucle peut être exécutée au total $O(\log n)$ fois.

Remarque. Chose intéressante, on n'effectue jamais plus de deux rotations.

7. L'opération $SUPPRIMER$

Comme les autres opérations primitives sur un arbre rouge et noir à n nœud, la suppression d'un nœud requiert un temps en $O(\log n)$.