Eric THIERRY
Scribes: Marc LASSON – Emmanuel LASALLE

1 Adapter une structure de donnée connue

1.1 Les arbres Rouge-Noir

Définition 1.1 *Un arbre binaire de recherche est une structure de donnée en forme d'arbre binaire dont les clefs sont des éléments d'un ensemble totalement ordonné (typiquement $\mathbb{N}$) et dont les noeuds vérifient l'invariant suivant : La clef de chaque noeud est plus grande que toutes les clefs de son sous-arbre gauche et plus petite que toutes les clefs de son sous-arbre droit.*

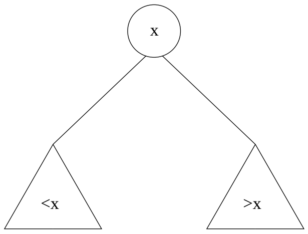

FIG. 1 – Un arbre binaire de recherche (à gauche) et un tas binaire (à droite).

La recherche d'un élément, du minimum, du maximum, d'un prédecesseur et d'un successeur, l'insertion et la suppression se font en $\mathcal{O}(h)$ où h est la hauteur de l'arbre.
Attention ! Ne pas confondre les arbres binaires de recherche avec les tas binaires. Les tas vérifient l'invariant : La clef de chaque noeud est plus grande que toutes les clefs de ses deux sous-arbres [1].

Définition 1.2 *Un arbre Rouge-Noir est un arbre binaire de recherche où*

1. *chaque noeud est soit Rouge, soit Noir,*
2. *chaque feuille est Noire,*
3. *si un noeud est Rouge, alors ses deux fils sont Noirs.*
4. *il y autant de noeuds Noirs sur chacun des chemins d'un noeud à ses feuilles.*

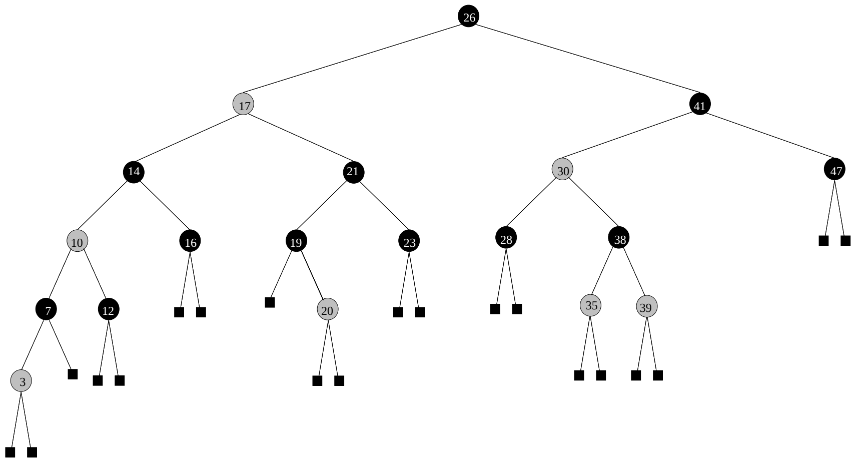

FIG. 2 – Un arbre Rouge-Noir.

[1]Pour se préparer aux tas de Fibonacci, il est conseillé de regarder l'implémentation des tas binomiaux (*Design and Analysis of Algorithms*, D. KOZEN, par exemple).

Les conditions imposées garantissent l'équilibre dans l'arbre (contrairement aux arbres binaires de recherche simples qui peuvent par exemple être des peignes).

Proposition 1.1 *La hauteur d'un arbre Rouge-Noir de n noeuds est majorée par* $2\log_2(n+1)$.

Démonstration: Soit N le nombre de noeuds Noirs par branche et h la hauteur de l'arbre.
- Les N premiers niveaux de l'arbre binaire sont complets et donc $n \geq 2^N - 1$.
- La condition (3) implique qu'il y a au plus N Rouges par branche et donc $h \leq 2N$ et donc que $n \geq 2^{h/2} - 1$ et que $h \leq 2\log_2(n+1)$.

1.2 Impémentation des opérations de base

L'opération de base est la rotation (*cf.* figure 3). Elle s'effectue en temps constant et préserve la propriété des arbres binaires de recherche.

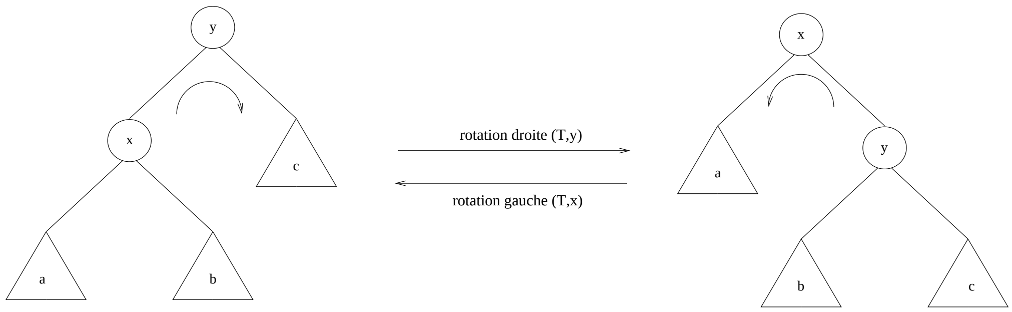

Fig. 3 – Rotations dans un arbre binaire.

Algorithme 1 Insertion dans un arbre Rouge-Noir

Entrée: Un arbre Rouge-Noir T et une clef x.

1: Insérer x dans T comme dans un arbre binaire de recherche classique et colorier le nouveau noeud en Rouge.

2: Réquilibrer en faisant des rotations et des changements de couleurs qui font remonter vers la racine tout motif violant la condition (3) jusqu'à disparition des problèmes (*cf.* le paragraphe suivant).

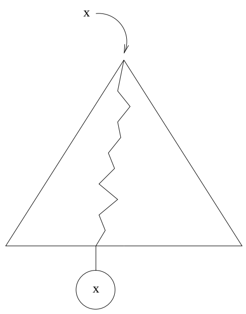

Fig. 4 – Insertion dans un arbre binaire.

Etude des problèmes lors de l'insertion :

Soit x un endroit potentiellement problématique. Aux symétries gauche/droite près, on se ramène à trois cas.

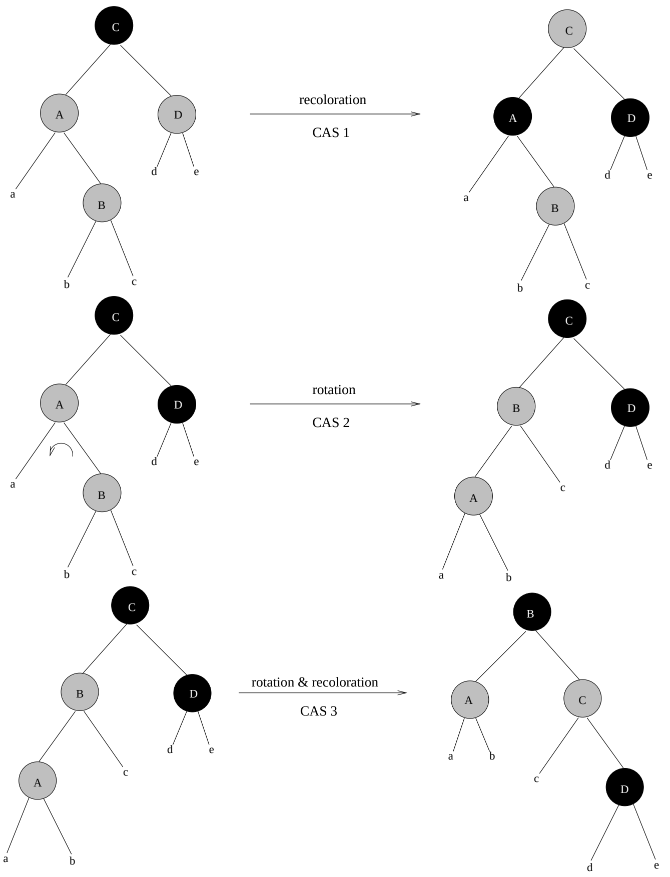

Fig. 5 – Les trois cas problématiques.

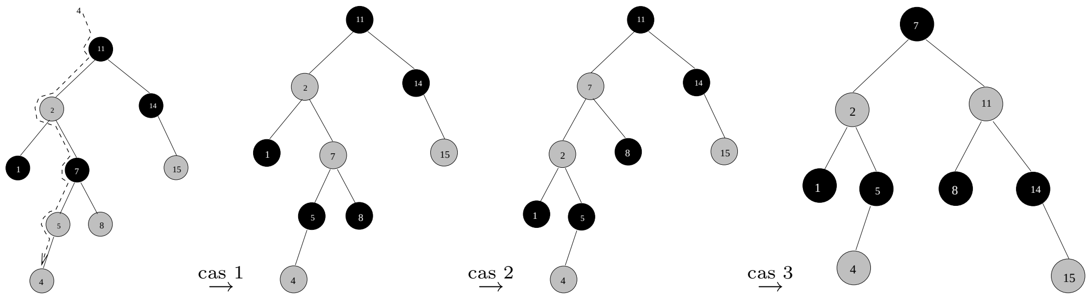

Fig. 6 – Exemple d'insertion dans un arbre Rouge-Noir faisant intervenir les trois cas.

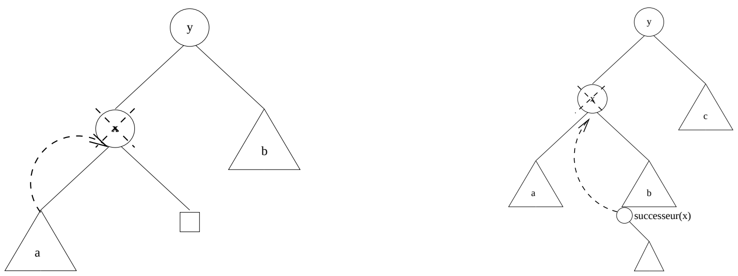

Fig. 7 – Suppression dans un arbre Rouge Noir.

Algorithme 2 Suppression dans un arbre Rouge-Noir

Entrée: Un arbre Rouge-Noir T et une clef $x \in T$.

On supprime x de T comme dans un arbre binaire de recherche classique.
si x a 0 ou 1 fils **alors**,
    on fait remonter ses sous-arbres (Fig. 7 à gauche).
sinon
    on remplace x par son successeur(x) qui a au plus 1 fils (droit) et on se ramène au cas précédent pour la suppression de successeur(x) (Fig. 7 à droite).
On élimine les problèmes sur les conditions en ouvrant son Cormen.

1.3 Exemple d'extension d'une structure de donnée

On suppose qu'on dispose d'un ensemble $S = \{x_0, \dots, x_{n-1}\}$ de clefs d'un ensemble totalement ordonné. On suppose $x_0 < \cdots < x_{n-1}$.

Définition 1.3

$$\begin{aligned} \operatorname{rang}(x_i) &= i \\ \operatorname{rang}^{-1}(i) &= x_i \end{aligned}$$

Remarque 1.1 *Le minimum de S est* $\operatorname{rang}^{-1}(1)$ *et le maximum* $\operatorname{rang}^{-1}(n)$.

On cherche à implémenter une structure de donnée pour gérer cet ensemble de clefs avec les opérations rechercher, predecesseur, successeur, inserer, supprimer + rang, rang^{-1}.

Idée 1 : Arbre Rouge-Noir avec un champs «rang» à chaque noeud.

- On dispose des opérations rang en $\mathcal{O}(1)$ et rang^{-1} en $\mathcal{O}(\log n)$,
- mais on a quelques problèmes pour mettre à jour ces champs après inserer et supprimer.

Idée 2 : Arbre Rouge-Noir avec un champs «taille» à chaque noeud.

Le nouveau champs contient la somme des tailles des sous-arbres du noeud. Ce champs nous permet d'implémenter facilement le calcul de rang^{-1} en $\mathcal{O}(\log n)$.

$$\begin{aligned} \operatorname{rang}^{-1}(i) &= \operatorname{rang}^{-1}(\text{racine}, i) \\ \operatorname{rang}^{-1}(x, i) &= i\text{-ème clef dans sous-arbre de racine } x \\ &= \begin{cases} \text{clef}(x) \text{ si } i = r \\ \operatorname{rang}^{-1}(\text{gauche}(x), i) \text{ si } i < r \\ \operatorname{rang}^{-1}(\text{droit}(x), i - r) \text{ si } i > r \end{cases} \quad \text{où } r = \text{taille}(\text{gauche}(x)) + 1 \end{aligned}$$

Et l'algorithme 3 nous donne le rang également en $\mathcal{O}(\log n)$.

Algorithme 3 Calcul du rang dans cette structure de donnée

Entrée: Un arbre Rouge-Noir avec champs «taille» T et une clef x.
Sortie: Le rang rang(x) de x.

$r \leftarrow$ taille(gauche(x)) + 1
$y \leftarrow x$
tant que $y \neq$ racine **faire**,
    si y = droit(pere(y)) (y *est un fils droit*) **alors**,
        $r \leftarrow r$ + taille(gauche(pere(y))) + 1
        $y \leftarrow$ pere(y)
retourner r

Mise à jour du champs «taille» pendant inserer **et** supprimer**.**

- insertion :
  1. On insére x dans T à l'aide de l'algorithme 1 en incrémentant la taille le long de la branche parcourue ($\mathcal{O}(\log n)$).
  2. Lors du réquilibrage, seules les rotations changent le champs «taille» *cf.* figure 8 ($\mathcal{O}(1)$).
- suppression : Même type de mise à jour.

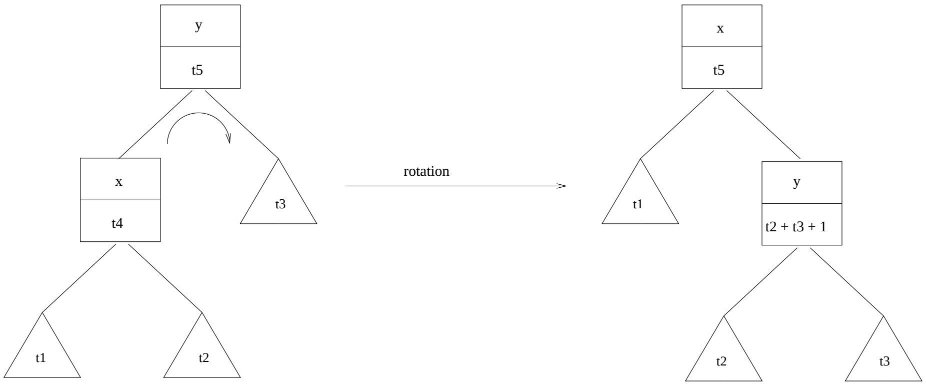

FIG. 8 – Mise à jour du champs «taille» lors des rotations.