Annexe G

Arbres rouges-noirs

Hugo Salou MPI*

Dernière mise à jour le 27 mars 2023

Un arbre rouge-noir est un cas particulier des arbres binaires de recherches. On l'utilise notamment pour représenter des ensembles, on veut donc réaliser deux opérations simples : l'insertion et le test d'appartenance. Initialement, on pense représenter un ensemble par une liste triée. Mais, on utilise plutôt un arbre binaire pour représenter des données avec une hauteur logarithmique, contrairement à une hauteur linéaire. Les arbres binaires de recherches sont des arbres dans lesquels ont peut réaliser une dichotomie.

```
type 'a btree = E | N of 'a * 'a btree * 'a btree

let rec mem (x: 'a) (t: 'a btree): bool =
  match t with
  | E -> false
  | N(y, g, d) ->
      if x < y       then mem x g
      else if x = y then true
      else               mem x d

let rec insere (x: 'a) (t: 'a btree): 'a btree =
  match t with
  | E -> false
  | N(y, g, d) ->
      if x < y       then N(y, insere x g, d)
      else if x = y then t
      else               N(y, g, insere x d)
```

Code 1 – Arbre binaire de recherche

La fonction `mem` permet de réaliser ce test d'appartenance et la fonction `insere` insère l'insertion dans l'arbre. Ainsi, on peut représenter un ensemble avec le type `'a btree`.

Mais, cet arbre peut être déséquilibré, et l'utilisation de la dichotomie ne donne pas de résultats très avantageux. On utilise donc un arbre *auto-équilibrant*, comme les AVL du 1er DM. Pour les AVL, la différence de hauteur est -1, 0 ou 1.

On introduit donc le concept d'arbre rouge-noir. Un arbre rouge-noir est un arbre parfait, qui a une certaine « élasticité. » On colorie chaque nœuds pour imposer des contraintes sur cette élasticité. Les branches de l'arbre a une longueur de rupture. Un arbre contenant uniquement des nœuds noirs est un arbre parfait. Et, entre deux nœuds noirs, on peut insérer un nœud rouge. Un arbre rouge-noir vérifie donc les trois propriétés suivantes :

(*a*) la racine est noire,

(*b*) le père d'un nœud rouge est noir,

(*c*) la hauteur noir de chaque feuille externe est constante. [important]

Une *feuille externe* est, dans le code OCaml, l'expression `E`; et, la *hauteur noir* d'une feuille externe est le nombre de nœuds noirs depuis la racine. On définit la *hauteur noir* d'un arbre comme la hauteur noir de chaque feuille externe (qui est constante).

Le problème est l'insertion d'un nœud. Insérer un nœud noir est, en général, plus dangereux car il modifie la hauteur noir de tout l'arbre. On préfère donc insérer un nœud rouge, sauf dans le cas de la racine.

On considère donc la propriété (*c*) comme invariante. En effet, corriger un arbre pour valider la propriété (*a*) ou la propriété (*b*) est bien plus simple.

On traite tous les cas dans le diaporama sur *cahier-de-prépa*, et on réalise l'exemple sur les lettres A, L, G, O, R, I, T, H, M, E.

Pour supprimer un nœud dans un arbre binaire classique, on peut le remplacer par le maximal de son sous-arbre droit, ou le minimum de son sous-arbre gauche. Si on supprime un nœud ayant un seul fils, on n'a qu'à re-brancher le sous-arbre. Pour les arbres rouges-noirs, c'est supprimer un nœud noir qui pose problème. Pour cela, on introduit les nœuds doublement noirs, qui comptent pour deux dans la hauteur noir. Ainsi, on supprime le nœud noir et on remplace les autres nœuds par des nœuds doublement noirs. L'algorithme n'a donc qu'à faire remonter le nœud doublement noir, jusqu'à la racine, où il sera transformé en nœud simplement noir.

Un arbre de hauteur h a une hauteur noir $\mathrm{bh} \geqslant h/2$. Et, la taille, *i.e.* le nombre de nœuds, est supérieure à $2^{\mathrm{bh}} - 1$. On conclut que $h \leqslant 2\log_2(\mathrm{taille}+1)$. De même par la propriété (c) permet de conclure que $h = \Theta(\log_2 \mathrm{taille})$.