Arbre binaire de recherche équilibré, arbre rouge-noir

Quentin Fortier

July 2, 2022

Binder

Arbre binaire de recherche (ABR)

On a écrit les fonctions suivantes sur un ABR de hauteur h :

- `has` : test d'appartenance en $O(h)$
- `add` : ajout d'un élément en $O(h)$
- `del` : suppression d'un élément en $O(h)$

Arbre binaire de recherche (ABR)

On a écrit les fonctions suivantes sur un ABR de hauteur h :

- `has` : test d'appartenance en $O(h)$
- `add` : ajout d'un élément en $O(h)$
- `del` : suppression d'un élément en $O(h)$

En mettant des couples (clé, valeur) sur chaque noeud et en utilisant seulement les clés pour les comparaisons, on obtient une implémentation de dictionnaire dont les opérations sont en $O(h)$.

Arbre binaire de recherche (ABR)

Si un ABR est équilibré (de hauteur $h = \mathrm{O}(\log(n))$), les opérations `add`, `del`, `has` sont en $\mathrm{O}(\log(n))$.

Opération	ABR	ABR équilibré	Table de hachage
ajouter	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne
supprimer	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne
rechercher	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne

Arbre binaire de recherche (ABR)

Si un ABR est équilibré (de hauteur $h = \mathrm{O}(\log(n))$), les opérations `add`, `del`, `has` sont en $\mathrm{O}(\log(n))$.

Opération	ABR	ABR équilibré	Table de hachage
ajouter	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne
supprimer	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne
rechercher	$\mathrm{O}(h)$	$\mathrm{O}(\log(n))$	$\mathrm{O}(1)$ en moyenne

Exemples d'ABR équilibrés :

1. ARN : au programme
2. AVL : utilisé par la stdlib d'OCaml pour implémenter le type `Map` (dictionnaire)
3. B-tree : utilisé dans les systèmes de fichier, par exemple de Linux (voir aussi le manuel *man*)

Arbre rouge-noir

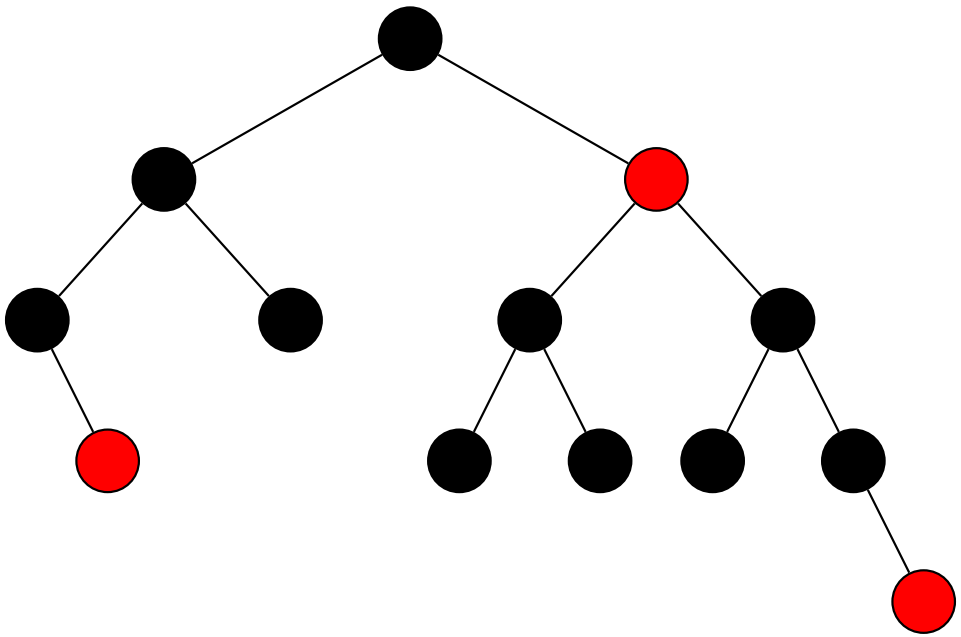
Un arbre rouge-noir (ARN) a est un ABR dont les noeuds sont coloriés en rouge ou noir, et vérifiant les conditions suivantes :

1. La racine est noire (non obligatoire, mais rend le code plus simple)
2. Si un sommet est rouge, ses éventuels fils sont noirs.
3. Le nombre de noeuds noirs le long de n'importe quel chemin de la racine **à un emplacement vide** (`V`) est le même, que l'on appelle *hauteur noire* et note $h_b(a)$.

Arbre rouge-noir

Un arbre rouge-noir (ARN) a est un ABR dont les noeuds sont coloriés en rouge ou noir, et vérifiant les conditions suivantes :

1. La racine est noire (non obligatoire, mais rend le code plus simple)
2. Si un sommet est rouge, ses éventuels fils sont noirs.
3. Le nombre de noeuds noirs le long de n'importe quel chemin de la racine **à un emplacement vide** (`V`) est le même, que l'on appelle *hauteur noire* et note $h_b(a)$.



Arbre rouge-noir

Un arbre rouge-noir (ARN) a est un ABR dont les noeuds sont coloriés en rouge ou noir, et vérifiant les conditions suivantes :

1. La racine est noire (non obligatoire, mais rend le code plus simple)
2. Si un sommet est rouge, ses éventuels fils sont noirs.
3. Le nombre de noeuds noirs le long de n'importe quel chemin de la racine **à un emplacement vide** (`V`) est le même, que l'on appelle *hauteur noire* et note $h_b(a)$.

Exercice

Donner un arbre binaire qui ne peut pas être colorié en ARN.

Arbre rouge-noir

Théorème

Soit a un ARN avec $n(a)$ noeuds et de hauteur $h(a)$. Alors :

$$h(a) \leq 2h_b(a)$$

Arbre rouge-noir

Théorème

Soit a un ARN avec $n(a)$ noeuds et de hauteur $h(a)$. Alors :

$$h(a) \leq 2h_b(a)$$

Preuve : Soit $\mathcal{C}$ un chemin de longueur $h(a)$ de la racine à une feuille.

Arbre rouge-noir

Théorème

Soit a un ARN avec $n(a)$ noeuds et de hauteur $h(a)$. Alors :

$$h(a) \leq 2h_b(a)$$

Preuve : Soit $\mathcal{C}$ un chemin de longueur $h(a)$ de la racine à une feuille. Alors $\mathcal{C}$ a au moins $\frac{h(a)}{2}$ noeuds noirs donc $h_b(a) \geq \frac{h(a)}{2}$.

Arbre rouge-noir

Théorème

Soit a un ARN avec $n(a)$ noeuds et de hauteur $h(a)$. Alors :

$$n(a) \geq 2^{h_b(a)} - 1$$

Arbre rouge-noir

Théorème

Soit a un ARN avec $n(a)$ noeuds et de hauteur $h(a)$. Alors :

$$n(a) \geq 2^{h_b(a)} - 1$$

Preuve : Par récurrence sur $n(a)$.

Arbre rouge-noir

Corollaire

Soit a un ARN avec n noeuds et de hauteur h. Alors :

$$h \leq 2\log_2(n+1)$$

On a donc aussi :

$$h = \mathsf{O}(\log_2(n)))$$

Preuve :

Arbre rouge-noir

Corollaire

Soit a un ARN avec n noeuds et de hauteur h. Alors :

$$h \leq 2\log_2(n+1)$$

On a donc aussi :

$$h = \mathsf{O}(\log_2(n)))$$

Preuve :On a montré $h \leq 2h_b(a)$ et $n \geq 2^{h_b(a)} - 1$.
D'où :

$$h \leq 2h_b(a) \leq \log_2(n+1)$$

Arbre rouge-noir : Fonctions utilitaires

```
type 'a rb_tree =
      E
    | R of 'a * 'a rb_tree * 'a rb_tree (* noeud rouge *)
    | B of 'a * 'a rb_tree * 'a rb_tree (* noeud noir *)

let red = function (* teste si la racine est rouge *)
    | E | B(_, _, _) -> false
    | R(_, _, _) -> true

let b = function (* rend la racine noire *)
    | R(r, g, d) -> B(r, g, d)
    | a -> a
```

Arbre rouge-noir : Appartenance

Le test d'appartenance est très similaire aux ABR (les contraintes d'arbre rouge-noir ne peuvent pas être violées) :

```
let rec has e = function
    | E -> false
    | R(r, g, d) | B(r, g, d) ->
        e = r || has e (if e < r then g else d)
```

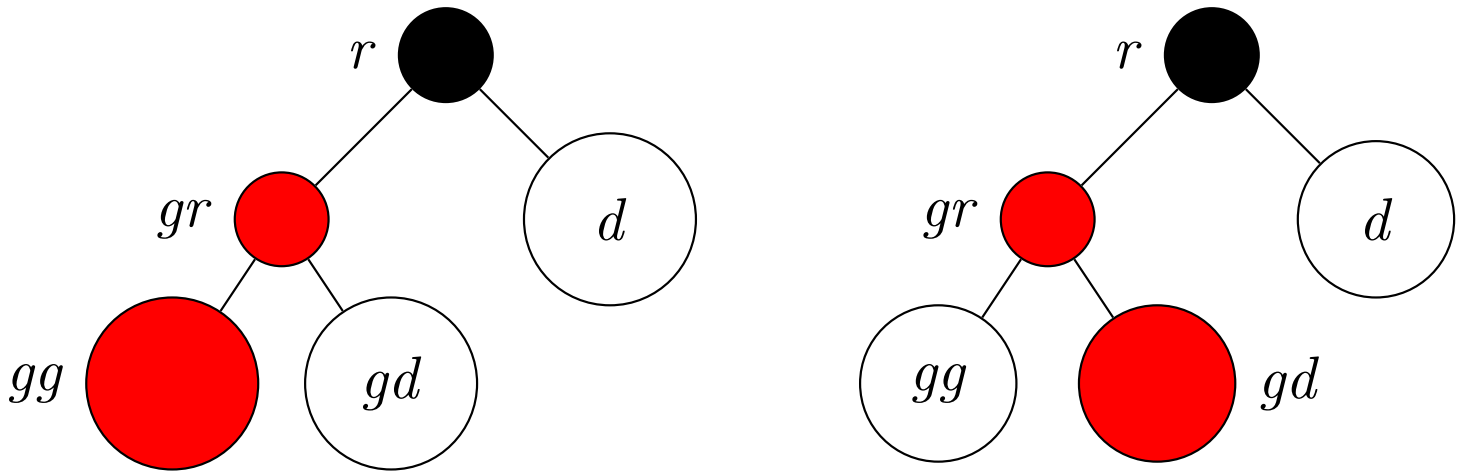
Ajout

On ajoute un élément en tant que feuille rouge, de la même façon que pour les ABR.

Ajout

On ajoute un élément en tant que feuille rouge, de la même façon que pour les ABR.

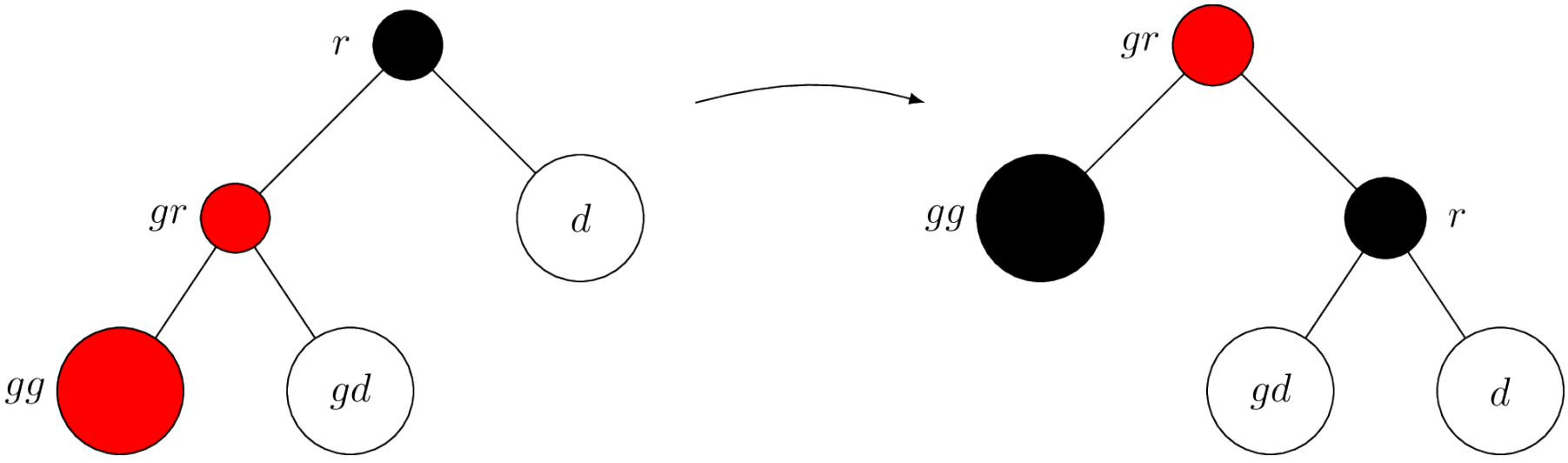
Cet ajout peut détruire la propriété d'ARN, en donnant un noeud rouge dont le fils est rouge. Il y a 4 situations possibles, dont les 2 suivants et les symétriques :



On suppose que gg, gd, d sont des ARN.

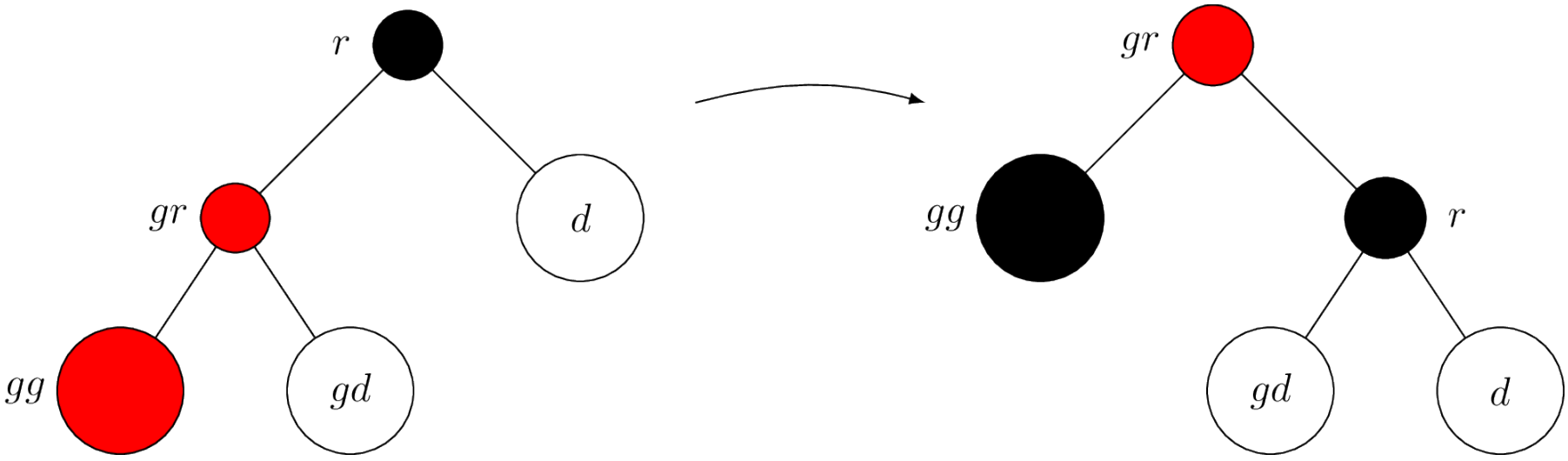
Ajout : Rotation droite

Dans le 1er cas, on utilise une rotation pour rétablir la condition d'ARN. Voici est une rotation droite (une rotation gauche fait la même chose, mais de droite à gauche) :



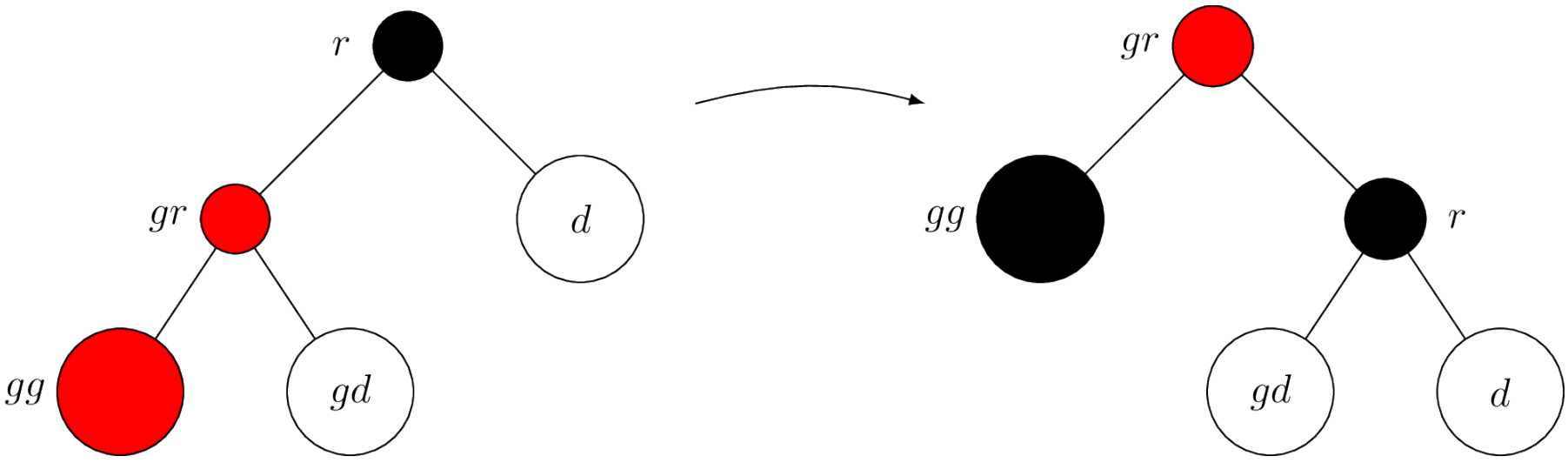
Ajout : Rotation droite

Dans le 1er cas, on utilise une rotation pour rétablir la condition d'ARN. Voici est une rotation droite (une rotation gauche fait la même chose, mais de droite à gauche) :



- La condition d'ABR est toujours respectée après rotation
- Les hauteurs noires ne changent pas
- Il n'y a plus de noeud rouge avec un père rouge, sauf peut-être pour la nouvelle racine avec son père : il faut faire des rotations récursivement

Ajout : Rotation droite

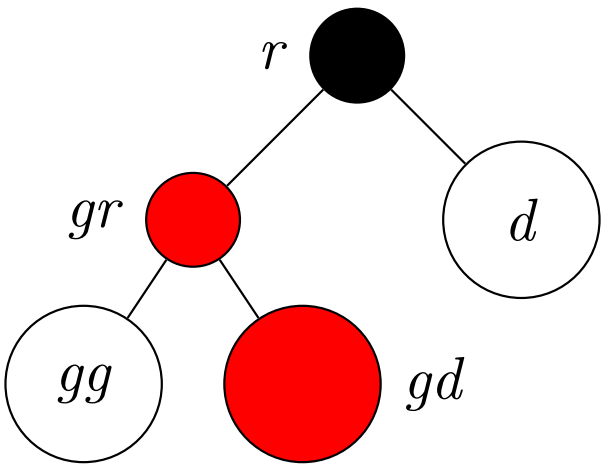



```
let rotd = function
    | B(r, R(gr, gg, gd), d) -> R(gr, b gg, B(r, gd, d))
```

Remarque : il y a en fait un cas symétrique qu'il faut ajouter à `rotd`.

Ajout : Rotation gauche-droite

Dans le cas suivant, on effectue d'abord une rotation gauche sur g pour se ramener au cas précédent :



Ajout : Rotation gauche-droite

La fonction suivante effectue 1 ou 2 rotations :

```
let rec balance t =
    let B(r, g, d) = t in match g, d with
  | R(gr, gg, gd), d when red gg -> rotd t
  | R(gr, gg, gd), d when red gd -> balance (B(r, rotg g, d))
  | g, R(dr, dg, dd) when red dd -> rotg t
  | g, R(dr, dg, dd) when red dg -> balance (B(r, g, rotd d))
  | _, _ -> t;;
```

`balance` est récursif seulement pour effectuer une 2ème rotation si besoin.

Ajout : add

On ajoute un noeud comme dans un ABR puis on restaure la propriété d'ARN avec `balance` :

```
let add t e =
    let rec aux = function
      | E -> R(e, E, E)
      | B(r, g, d) when e < r -> balance (B(r, aux g, d))
      | R(r, g, d) when e < r -> R(r, aux g, d)
      | B(r, g, d) -> balance (B(r, g, aux d))
      | R(r, g, d) -> R(r, g, aux d) in
    b (aux t) (* on rend la racine noire *)
```

Suppression

La suppression est compliquée...