TP n°22 - Équilibrage des ABR Rouge/Noir

Un **arbre bicolore** ou **arbre rouge et noir** ou **arbre rouge-noir** est un arbre binaire de recherche **équilibré**, dont l'équilibrage est garanti par des contraintes sur la couleur des nœuds, qui peuvent être rouges ou noirs.
En contrôlant la manière dont les nœuds sont coloriés sur n'importe quel chemin allant de la racine à une feuille, les arbres rouge-noir garantissent qu'aucun de ces chemins n'est plus de deux fois plus long qu'un autre, ce qui rend l'arbre approximativement équilibré. Les opérations de recherche, insertion ou de suppression se font donc efficacement en $O(\log n)$ si n est le nombre d'éléments dans l'arbre, à condition toutefois d'être capable de conserver les propriétés d'arbre rouge-noir lors de ces opérations.

I. Arbres bicolores rouge-noir

Définition 1 (Arbre bicolore)

Un **arbre bicolore rouge-noir** est

- soit l'arbre vide E
- soit un arbre binaire t pour lequel chaque nœud est associé soit à la couleur **rouge**, soit à la couleur **noire**, et qui vérifie les propriétés suivantes :

  (P1) t est un ABR

  (P2) La **racine** est **noire**.

  (P3) Un nœud rouge ne peut pas avoir d'enfant rouge.

  (P4) Pour un nœud donné, tous les chemins de ce nœud à un sous arbre vide comportent le même nombre de nœuds noirs.

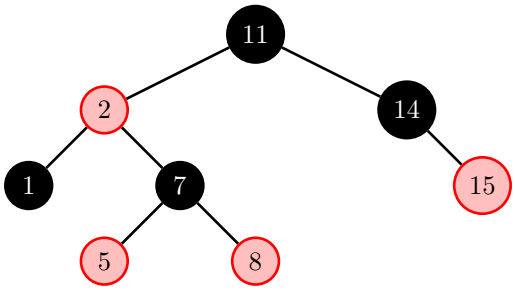

FIGURE 1 – Exemple d'arbre rouge-noir

Exercice 1 (Étude théorique : équilibrage des arbres bicolores).

1. Indiquer pourquoi chacun des arbres de la figure 2 n'est pas un arbre rouge-noir.
2. On considère le nombre de nœuds noirs d'un nœud vers n'importe quelle feuille, notion bien définie d'après **(P4)**.
   Quelle est ce nombre de nœuds noirs $b(t)$ de l'arbre rouge-noir de l'exemple 1.
3. Justifier que l'on peut remplacer la propriété **(P4)** par : **tout chemin de la racine à une feuille comporte le même nombre de nœuds noirs**.
4. Soit t un arbre vérifiant $(P3)$ et $(P4)$. On note $b(t)$ le nombre de nœuds noirs de la racine à une feuille de t. Cette valeur $b(t)$ sera appelée **hauteur noire** de l'arbre t. Montrer que l'on a : $2^{b(t)} \leq n(t)+1$, où $n(t)$ désigne le nombre de nœuds de l'arbre t.
5. Montrer que, pour tout arbre t vérifiant $(P3)$ et $(P4)$, la hauteur $h(t)$ de cet arbre vérifie : $h(t) \leq 2b(t)$
6. Montrer que les arbres bicolores forment une famille d'arbres équilibrés.
7. Comparer la famille des arbres rouge-noir et celle des arbres AVL en terme d'équilibrage, de complexité pour la mise en œuvre algorithmique...

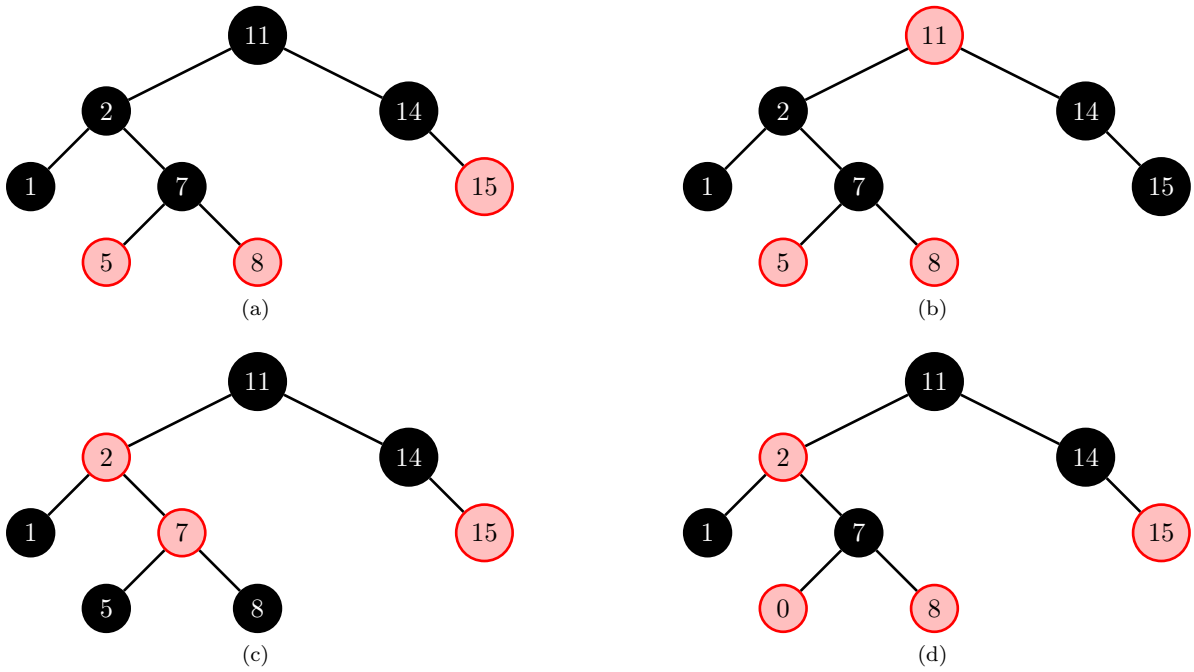

FIGURE 2 – Arbres qui ne sont pas des arbres rouge-noir.

Attention. La ≪ hauteur ≫ noire n'est pas une vraie hauteur, au sens des conventions que nous avons adopté précédemment dans ce cours. En effet, notre convention de hauteur avec $h(E) = -1$ revient à compter des arêtes, alors que la hauteur noire consiste à compter des noeuds noirs... Le mot hauteur doit donc être utilisé en gardant à l'esprit cette différence.

II. Insertion dans un arbre rouge-noir

La **recherche** d'un élément dans un arbre rouge-noir se fait exactement comme dans un arbre binaire de recherche. Il suffit d'ignorer la couleur des nœuds.
L'**insertion** d'un élément dans un arbre rouge-noir se fait également de la même manière : on recherche la position d'insertion et on y ajoute une feuille contenant l'élément à insérer.
On convient que l'on colorie cette nouvelle feuille en rouge.

Exercice 2 (Problème rouge-rouge).

1. Dessiner l'arbre obtenu après ajout de l'élément 13 dans l'arbre de l'exemple 1. L'arbre obtenu est-il toujours un arbre rouge-noir ?
2. Dessiner l'arbre obtenu après ajout de l'élément 3 dans l'arbre obtenu précédemment. Cette arbre est-il encore un arbre rouge-noir ?
3. Quelle(s) propriété(s) des arbres rouge-noir peuvent être violées par l'ajout d'une feuille rouge à la bonne position ? Justifier.

Les réponses de l'exercice précédent indiquent qu'il n'y a que deux violations possibles des propriétés d'arbre rouge-noir suite à l'insertion d'un nœud rouge :

a) La racine est rouge.

b) Un nœud rouge a un fils rouge ; ce nœud problématique est unique

Notons que la racine ne peut être rouge que si l'arbre était initialement vide, cependant ce premier cas pourra également apparaître lors des corrections que nous allons effectuer pour corriger le problème b). Dans le cas b) le nœud qui a un fils rouge est nécessairement le père de la feuille qui vient d'être insérée, mais on va faire **remonter** le problème vers la racine dans le processus de correction, qui pourra donc être déplacé à d'autres endroits par la suite. Le cas b) est dorénavant appelé un problème **rouge-rouge**.

Exercice 3 (Résolution du problème rouge-rouge).
On considère un arbre bicolore valide.
On y insère un nœud rouge : d'après l'exercice précédent, l'arbre obtenu vérifie toutes les propriétés des arbres bicolores sauf éventuellement a) et b) ci-dessus.

1. Justifier que si on est dans le cas a) alors il suffit de colorier la racine en noir pour obtenir un arbre rouge-noir correct.

2. On suppose désormais que l'on est dans le cas b) sans être dans le cas a). Montrer que le père du nœud rouge ayant un fils rouge existe et est noir.

3. Il y a alors quatre configurations possibles selon si les deux nœuds rouges du problème **rouge-rouge** sont fils gauches ou fils droits. La figure ci-après donne les deux configurations correspondant au cas où le nœud rouge ayant un fils rouge est fils gauche de son père noir. Les sous-arbres a, b, d et d sont quelconques mais vérifient les propriétés 1), 3) et 4) des arbres rouge-noir. Dessiner de même les deux configurations (3) et (4) symétriques, correspondant au cas où le nœud rouge ayant un fils rouge est fils droit de son père noir.

4. On se propose de corriger ces quatre cas en modifiant l'arbre comme indiqué sur la figure 3. Une telle transformation sera appelée **correction rouge** par la suite.

a. Justifier que l'arbre de la figure 3, et donc qu'une correction rouge, peut être obtenue par une rotation gauche, droite, gauche-droite ou droite-gauche à partir de chacune des quatre configurations, suivie d'une simple recoloration des nœuds. Préciser quelle rotation ou double rotation correspond à quel cas et quelle recoloration est effectuée.

b. Justifier qu'une correction rouge conserve les propriétés $(P1)$ et $(P4)$ d'arbre rouge-noir et corrige le problème b) correspondant à la propriété $(P3)$ pour le sous-arbre obtenu.

c. Cependant, on peut ainsi avoir créé un nouveau un problème **rouge-rouge** entre la nouvelle racine rouge y du nouveau sous-arbre et son éventuel père. Il suffit alors de recommencer avec une correction rouge plus haut. Justifier que ce processus termine.

d. Appliquer cette transformation complète à l'arbre de l'exercice précédent pour obtenir un arbre rouge-noir correct, en n'oubliant pas de recolorer la racine en noir à la fin si besoin pour garantir $(P2)$.

Remarque. Le cas de la suppression est plus délicat car la suppression d'un nœud noir peut changer la hauteur noire $b(t)$ de certains sous-arbres.

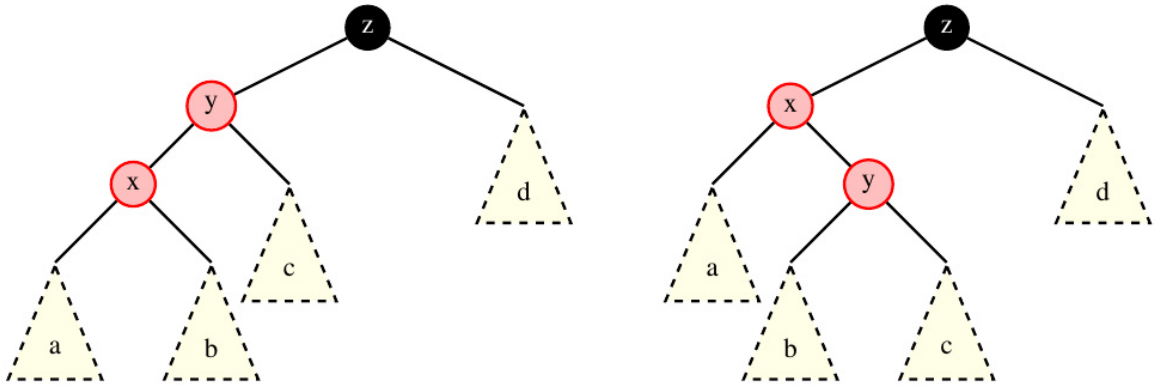

FIGURE 3 – Configurations (1) et (2).

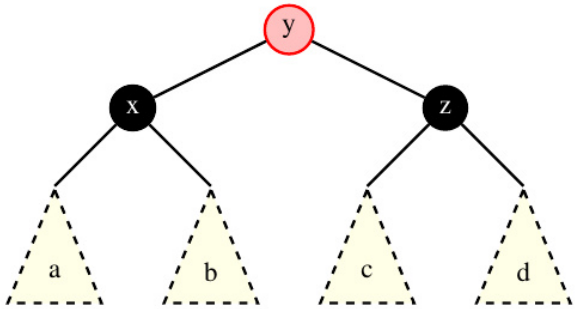

FIGURE 4 – Correction rouge proposée aux configurations (1), (2), (3) et (4).

FIGURE 3 – Configurations possibles de problèmes rouge-rouge (2 sur 4) lors d'une insertion et correction

III. Implémentation en OCaml

Remarque (Syntaxe de filtrage encore plus compacte en OCaml). En OCaml, si plusieurs cas de filtrages définissent *exactement* les mêmes variables, et si on souhaite apporter exactement la même réponse pour tous ces cas, on peut les regrouper avant une seule même flèche. Par exemple on peut écrire quelque chose comme :

```
match foo with
| A | B | C -> ......
| D x | E x | F (x, _) | G (_, x) -> .....
```

Il ne faut cependant pas en abuser.

On représente en OCaml un arbre rouge-noir par un arbre binaire avec un attribut supplémentaire correspondant à sa couleur.

Toutes les fonctions ci-dessous seront implémentées dans un fichier `NOM_bicolor.ml`.

Exercice 4 (Implémentation en OCaml).
On utilise les types suivants :

```
type color = R | B;;
type 'a bicolor = E | N of color * 'a bicolor * 'a * 'a bicolor;;
```

1. Écrire une fonction `height : 'a bicolor -> int` qui calcule la hauteur d'un arbre bicolore. Quelle est sa complexité ?
2. Écrire une fonction `black_height : 'a bicolor -> int` qui calcule la hauteur noire d'un arbre rouge-noir. Quelle est sa complexité ? *Indication : on suppose que l'arbre est rouge-noir ; ce ne doit pas être la même complexité que pour la hauteur !*
3. Écrire une fonction `is_bicolor : 'a bicolor -> bool` qui vérifie, en effectuant un seul parcours de l'arbre, si un arbre bicolore est effectivement un arbre rouge-noir. On suppose donc que l'arbre passé en argument est bien un arbre binaire de recherche (ou on le vérifiera à la fin de la fonction à l'aide de la fonction `is_bst` qui aura été adaptée pour les arbres bicolores). Remarquons que le type OCaml impose directement que la propriété 1) est nécessairement respectée.
4. Implémenter la fonction `red_corr: 'a -> 'a bicolor -> 'a bicolor`. Cette fonction effectue un filtrage sur la forme de l'arbre pour reconnaître éventuellement l'une des quatre configurations. Dans les quatre cas, l'arbre qui est alors obtenu est le même : celui de la figure 3.
5. Écrire une fonction `insert_balance : 'a -> 'a bicolor -> 'a bicolor` qui insère un élément dans un arbre rouge-noir et qui effectue les corrections nécessaires pour obtenir un arbre rouge-noir. Vérifier votre fonction sur l'exemple complet déroulé plus haut.

   Indication : lorsque l'on insère récursivement à gauche ou à droite, il faut effectuer une correction rouge si l'une des 4 configurations problématique apparaît. Il faut également remonter et vérifier toutes les configurations au dessus de l'insertion car une correction amène un nœud rouge comme racine du sous-arbre corrigé, et peut donc créer un problème rouge-rouge plus haut. Il ne faut pas oublier enfin de recolorer la racine tout en haut en noir car elle peut avoir été transformée en rouge toujours en lien avec la correction.
6. On insère dans l'ordre tous les entiers entre 1 et n. On doit obtenir les hauteurs et les hauteurs noires suivantes :

n(t)	10	10^2	10^3	10^4	10^5	10^6	10^7
h(t)	4	8	14	19	21	25	30
b(t)	3	6	9	13	16	19	23

   Vérifier que vous obtenez bien les mêmes hauteurs et hauteurs noires pour les arbres obtenus ainsi et que ce sont bien des arbres binaires rouge et noir.