Arbres rouge-noir

Présentation

Exercice 1. On appelle *arbre rouge-noir* un arbre binaire comportant un champ supplémentaire par nœud : sa couleur , qui peut être rouge ou noire, et qui vérifie les conditions suivantes :

P1 la racine est noire ;

P2 L'arbre vide est noir.

P3 Un nœud est soit rouge soit noir ;

P4 le parent d'un nœud rouge est noir ;

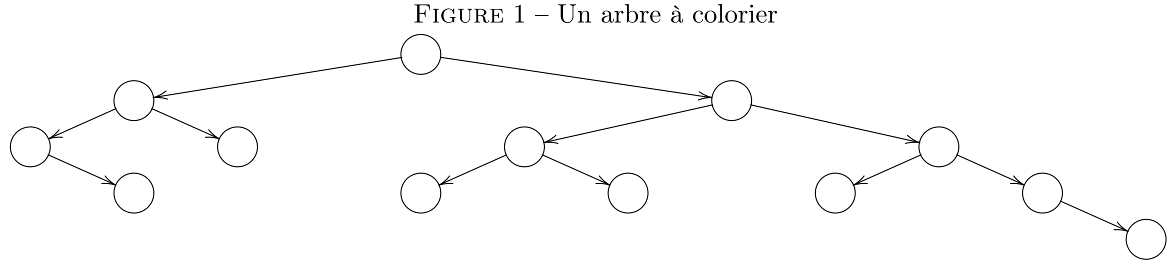
P5 pour chaque nœud, tous les chemins le reliant à contiennent le même nombre de nœuds noirs.il

Remarque. 1. Comme **Nil** est noir, P5 peut s'écrire : « pour chaque nœud, tous les chemins le reliant à des feuilles (mais pas à **Nil**) contiennent le même nombre de nœuds noirs. »

2. Dans les représentations on omet souvent de représenter l'arbre vide.
3. Un sous-arbre d'un arbre rouge-noir vérifie les propriétés P2 à P5 mais peut-être pas la propriété P1.
   Un arbre qui vérifie les propriétés P2 à P5 est appelé dans ce TP un *descendant licite* (sous-entendu : d'un arbre rouge-noir).

Q.1 Montrer que l'arbre 1 peut être muni d'une coloration rouge-noir. On n'a pas représenté les **Nil**.

Figure 1 – Un arbre à colorier



Q.2 Donner un exemple d'arbre qui ne puisse pas être muni d'une coloration rouge-noir.

Q.3 Étant donné un arbre rouge-noir A , on note $b(A)$ le nombre de nœuds noirs (non vides) que contient chacun des chemins de **Nil** à la racine (indépendant du choix de la feuille par définition). Par exemple, $b(\text{Nil}) = 0$. Montrer que

$$b(A) \leq h(A) + 1 \leq 2b(A) \text{ et } |A| \geq 2^{b(A)} - 1$$

En déduire que les arbres rouge-noir sont *équilibrés*, c'est à dire $h(A) = O(\log(|A|))$.

On définit les type :

```
type couleur = Rouge | Noir;;

type bicolore = Nil | Node of couleur * bicolore * int * bicolore;;
```

Q4 Écrire une fonction `hauteurnoire : bicolore -> int` qui détermine la *hauteur noire* d'un arbre supposé rouge-noir.

```
# let wa = let n6 = Node(Rouge,Nil,6,Nil) and n11 = Node(Noir,Nil,11,Nil) and
  n15=Node(Noir,Nil,15,Nil) and n22 =  Node(Rouge,Nil,22,Nil)
  and n27=  Node(Rouge,Nil,27,Nil) in let n1=Node(Noir,Nil,1,n6) and n25=Node(Noir,n22,25,n27
      )
  in let n8=Node(Rouge,n1,8,n11) and n17 =Node(Rouge,n15,17,n25) in Node(Noir,n8,13,n17);;
      val wa : bicolore =
  Node (Noir,
   Node (Rouge, Node (Noir, Nil, 1, Node (Rouge, Nil, 6, Nil)), 8,
    Node (Noir, Nil, 11, Nil)),
   13,
   Node (Rouge, Node (Noir, Nil, 15, Nil), 17,
    Node (Noir, Node (Rouge, Nil, 22, Nil), 25, Node (Rouge, Nil, 27, Nil))))
# hauteurnoire wa;;
- : int = 2
# hauteurnoire (Node(Rouge,Nil,6,Nil));;
- : int = 0
```

Q5 Écrire une fonction `check : bicolore -> bool` qui détermine si l'entrée est un arbre rouge-noir. L'algorithme choisi doit être de coût linéaire en $|A|$.

```
# check wa;;
- : bool = true
```

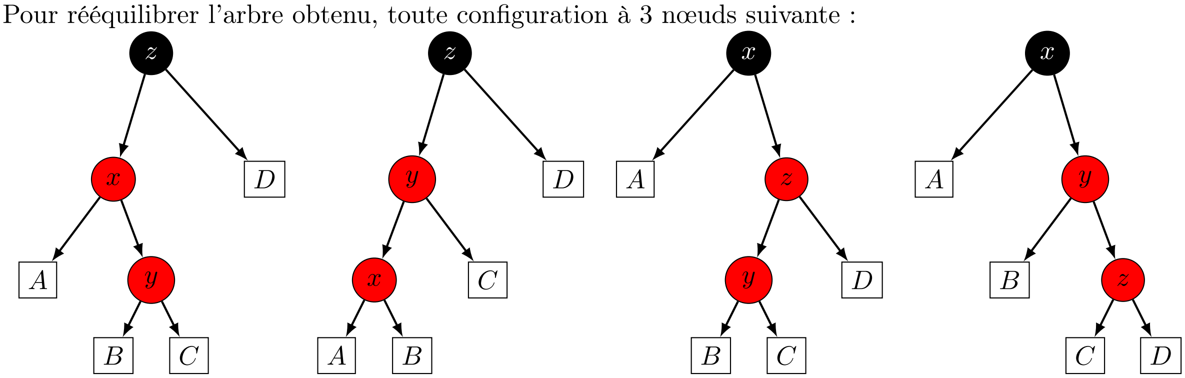
Insertion

Dans cette section les arbres rouge-noir sont également des ABR.

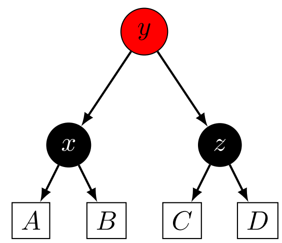
Exercice 2. **Q1** Rédiger une fonction insereABR de type `int -> bicolore -> bicolore` qui insère au niveau des feuilles un nouvel élément dans un arbre rouge-noir. Cette fonction devra préserver la structure d'ABR mais sans se soucier de la structure rouge-noir. On attribue arbitrairement la couleur rouge au nouveau nœud.

Q2 Lors de l'ajout via la fonction insereABR, quelle(s) propriété(s) des arbres rouge-noir peuvent être violées ?

Pour rééquilibrer l'arbre obtenu, toute configuration à 3 nœuds suivante :



est transformée en :



Nous appelons *correction rouge* cette transformation.

Q3 Analysons les transformations effectuées :

(a) La structure (pas la coloration) de l'arbre résultant est en fait obtenue par une ou deux rotations (voir cours) bien choisies.
Identifier ces rotations dans les 4 situations et en déduire que l'arbre résultant est un ABR.

(b) Pour la première des 4 situations à deux nœuds rouges consécutifs ci-dessus, on suppose que :

— Les 4 arbres A, B, C, D sont des ABR.

— A, B, C sont des arbres rouges noirs,

— D est un descendant licite.

— La propriété 5 (égalité du nombre de nœuds noirs pour toutes les branches issues de z) est vérifiée

Montrer que l'arbre résultant est un descendant licite, que sa hauteur noire est la même que l'arbre initial et que c'est un ABR.

Q4 Rédiger une fonction `correction_rouge` de type `bicolore -> bicolore` qui applique une correction rouge à un arbre de l'une des quatre formes présentées ci-dessus, et qui renvoie l'arbre inchangé dans les autres cas.

Q5 En déduire une nouvelle fonction d'insertion de type `int -> bicolore -> bicolore` baptisée `insereRN` qui insère un élément dans un arbre rouge-noir/ABR tout en préservant la structure rouge-noir/ABR.

Q6 Établir la correction de `insereRN` en présentant avec soin l'hypothèse d'induction.

Q7 Rédiger une fonction `test` de type `int -> bicolore` qui crée un arbre rougenoir en insérant successivement les entiers de 1 à n à l'aide de la fonction `insereRN`.
Vérifier que l'arbre obtenu pour $n = 32768$ est bien un arbre rouge-noir. Quelle est sa hauteur noire ?