Cours Informatique MP.

Arbres binaires de recherche équilibrés.

Antoine MOTEAU
antoine.moteau@wanadoo.fr

. . . /abrequ.tex (2003)
. . . /abrequ.tex Compilé le vendredi 09 mars 2018 à 09h 45m 14s avec LaTeX.
Compilable avec LaTeX, PDFLaTeX, LuaLaTeX, XeLaTeX.

Arbres binaires de recherche équilibrés.

Table des matières

Arbres binaires de recherche équilibrés.

1 Introduction

Dans les arbres binaires de recherche, de hauteur h, la complexité maximale de la recherche est un $O(h)$. On cherche donc à maintenir dans un arbre binaire de recherche une hauteur faible (à défaut de la plus faible possible) pour minimiser les coûts de recherche.
Dans un arbre binaire à n nœuds, de hauteur h, on sait que $h \leqslant n \leqslant 2^h - 1 \quad (\log_2(n+1) \leqslant h \leqslant n)$,

- le pire des cas ($h = n$) étant atteint par les arbres peignes,
- le meilleur des cas ($n = 2^h - 1$) étant atteint par les arbres binaires complets (voir ci-dessous).

1.1 Différentes notions d'équilibre (rappel)

Définition 1.1. Arbre binaire strictement équilibré, arbre binaire complet :

- Un arbre binaire de hauteur h est dit **strictement équilibré** (totalement équilibré) si tous les chemins menant de la racine à une feuille ont pour longueur h ou h-1.
- Si tous les chemins menant de la racine à une feuille ont pour longueur h, l'arbre est dit **complet** (tous les niveaux sont remplis).

Théorème 1.1. *Hauteur d'un arbre binaire strictement équilibré :*

Dans un arbre binaire strictement équilibré, de hauteur h, à n nœuds (et $p = n+1$ feuilles),

$$2^{h-1} \leqslant n \leqslant 2^h - 1 \qquad \textit{ou encore} \quad \log_2(n+1) \leqslant h \leqslant 1 + \log_2 n$$

(lorque l'arbre est complet, $n = 2^h - 1$).

Définition 1.2. Arbre binaire H-équilibré :

Un arbre (binaire) est dit **H-équilibré** (partiellement équilibré) si pour tout nœud, la différence de hauteur entre les arbres fils gauche et fils droit est au plus un.

Théorème 1.2. *hauteur d'un arbre binaire H-équilibré.*

Dans un arbre binaire H-équilibré, de hauteur h, à n nœuds,

$$h \leqslant 2.08 \ln(n+2) \quad (\textit{ ou } h \leqslant 1.45 \log_2(n+2) \qquad (\textit{plus précis que } h = O(\ln n))$$

1.2 Stratégies

La hauteur (donc le coût de recherche) est un $O(\ln n)$ dans les arbres strictement équilibrés ou H-équilibrés.

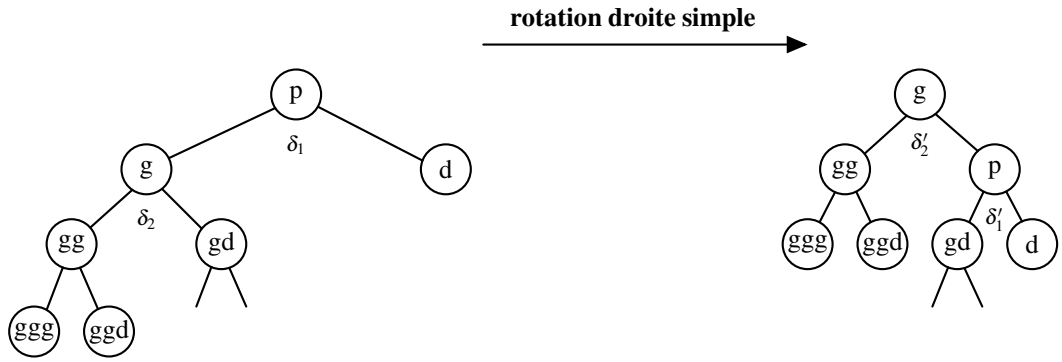
- Si un ABR est rarement modifié, on peut le reconstruire strictement équilibré à chaque modification.
- Si un ABR est fréquemment modifié, on préférera le maintenir seulement H-équilibré, ce qui n'oblige pas à reconstruire tout l'arbre à chaque modification et se fait à un coût moindre que la reconstruction complète de l'arbre.

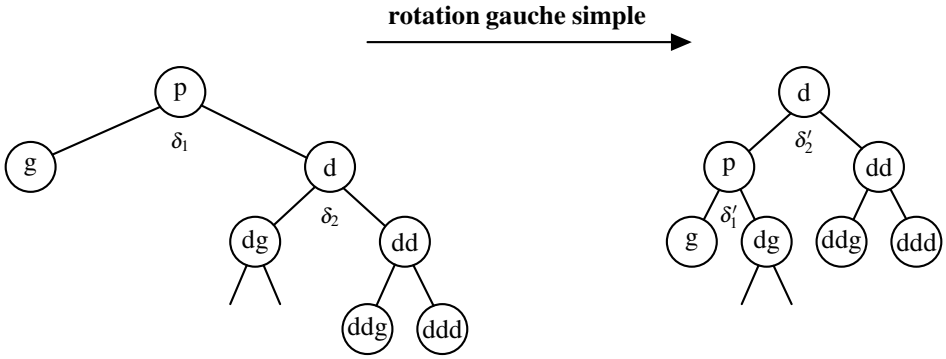
Exemples :

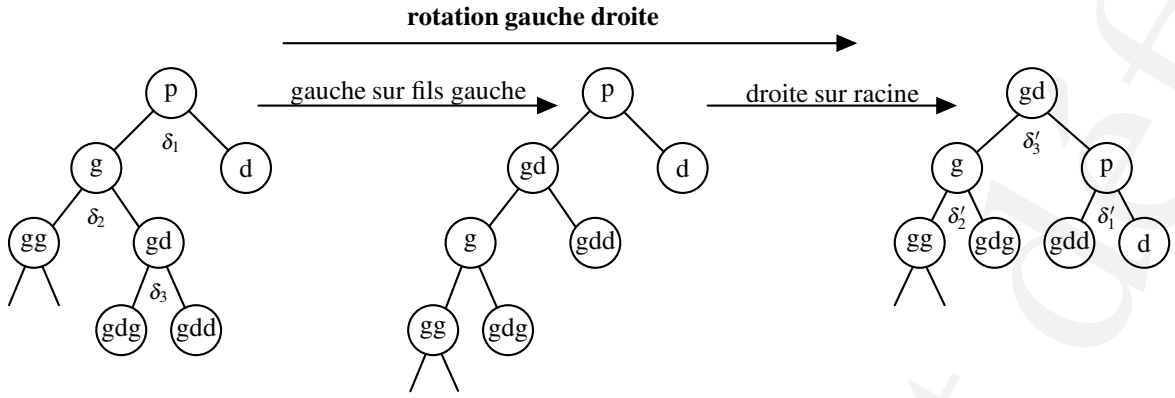
- Arbres AVL (Adel'son, Vel'skii, Landis, 1962).
  L'équilibre est maintenu par rotations et $O(\ln n)$ rotations peuvent être nécessaires pour maintenir l'équilibre après une suppression
- Arbres 2-3 (Hopcroft 1970). L'équilibre est maintenu en manipulant le degré des nœuds
- B-Arbres (Bayer, McCreight, 1972) généralisation des arbres 2-3
- Arbres Rouge et Noir (Bayer, Sleator, Tarjan)
- Arbres déployés (Sleator, Tarjan, 1983)
- ...

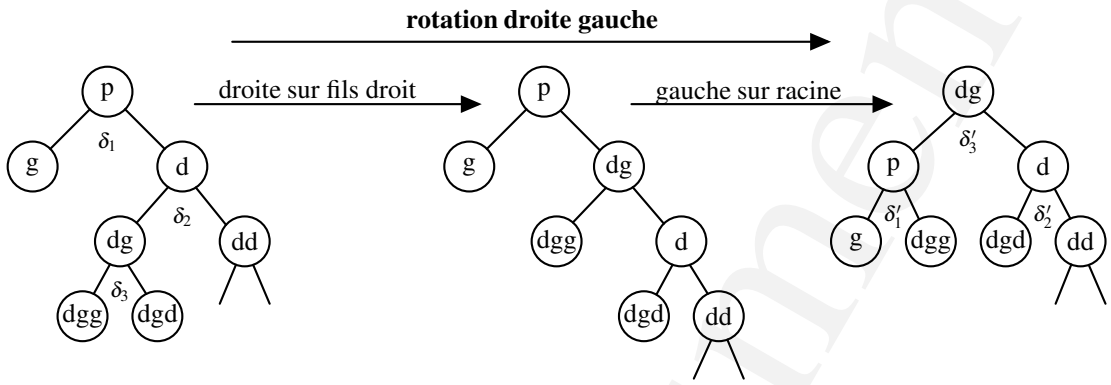
2 Rotations dans un ABR

Les rotations sont destinées à rétablir l'équilibre des ABR (après insertion ou suppression d'un nœud), en faisant passer des nœuds d'un fils à l'autre dans les sous-arbres, en conservant la structure d'ABR.









Les rotations ne concernent que les arbres (partie gauche des figures) qui possèdent la structure adéquate.

2.1 Rotations, sans équilibrage, dans un ABR défini par union

Pour modifier un arbre construit par union, il faut systématiquement en construire un nouveau (All operations over the type are purely applicative (no side-effects)).
Pour les illustrations, on utilisera un type d'arbre simple :

```
type 'n arbre =
 | F                                  (* ou Vide ou Rien ou V ou ... *)
 | N of 'n arbre * 'n * 'n arbre      (* N pour Noeud *)
;;
```

où une étiquette e est un triplet (clé, valeur associée, information d'équilibre) :

```
let cle (c,v,i) = c and valeur (c,v,i) = v and equilibre (c,v,i) = i;;
```

les clés étant ordonnées par la relation d'ordre générique (ordre <= par défaut) sur le type des clés.

2.1.1 Rotations simples : la racine passe à droite ou à gauche

```
let rotation_droite = function                    (* sans gestion d'équilibre *)
   | N( N(gg,g,gd), p, d) -> N(gg, g, N(gd,p,d))
   | a                    -> a
and rotation_gauche = function                    (* sans gestion d'équilibre *)
   | N(g, p, N(dg,d,dd)) -> N( N(g,p,dg), d, fd)
   | a                   -> a
;;
(* rotation_droite : 'a arbre -> 'a arbre = <fun> *)
(* rotation_gauche : 'a arbre -> 'a arbre = <fun> *)
```

On vérifie aisément que l'on transforme un ABR en un autre ABR.

2.1.2 Rotations composées, doubles : deux éléments passent à droite ou à gauche

1. **Rotation gauche-droite**
   Rotation gauche sur le fils gauche, suivie d'une rotation droite sur la racine de l'arbre global obtenu :

   ```
   let rotation_gauche_droite = function             (* sans gestion d'équilibre *)
     | F        -> F
     | N(g,p,d) -> rotation_droite (N(rotation_gauche g, p, d))
   ;;
   (* rotation_gauche_droite : 'a arbre -> 'a arbre = <fun> *)
   ```

2. **Rotation droite-gauche**
   Rotation droite sur le fils droit, suivie d'une rotation gauche sur la racine de l'arbre global obtenu :

   ```
   let rotation_droite_gauche = function             (* sans gestion d'équilibre *)
     | F        -> F
     | N(g,p,d) -> rotation_gauche (N(g, p, rotation_droite d))
   ;;
   (* rotation_droite_gauche : 'a arbre -> 'a arbre = <fun> *)
   ```

Les rotations doubles, compositions de rotations simples, transforment un ABR en un autre ABR.

2.2 Rotations dans un ABR défini par enregistrement à champs mutables

Lorsqu'un arbre est défini par **enregistrement** (ensemble de **champs nommés**) avec champs **mutables**, les modifications peuvent être effectuées en place, ce qui exige quelques précautions : il faut veiller à bien assurer le lien entre un père et ses fils (en particulier lorsque l'on échange les étiquettes de deux nœuds).

Par exemple, lorsque l'étiquette comprends une clé entière, une valeur et une information d'équilibre (entière),

```
type 'n arbre = { cle : int; mutable valeur : 'n; mutable equilibre : int;
                  mutable gauche : 'n arbre; mutable droite : 'n arbre }
;;
```

Autre exemple : Voir Concours CCS 2000, portant sur les arbres ~~rouges~~ blancs et noirs, où les étiquettes, qui sont des enregistrement avec champs mutables, contiennent, en plus de la clé et de la valeur associée,

- une couleur (~~rouge~~ blanc ou noir), qui est utilisée comme information d'équilibre,
- un lien du fils vers le père, ce qui est inhabituel (et peu souhaitable) .

3 Arbres AVL

- **Arbres binaires de recherche décorés par le déséquilibre (ABRdd)** :

  Les arbres binaires de recherche **décorés par le déséquilibre** (ABRdd) sont des arbres binaires de recherche où l'information d'équilibre relative à un nœud (contenue ici dans l'étiquette) est la différence δ de hauteur entre le fils gauche (f_g) et le fils droit (f_d) : $\delta = h(f_g) - h(f_d)$.

- **AVL** (**A**del'son, **V**el'skii, **L**andis) :

  Les AVL sont des arbres binaires de recherche décorés par le déséquilibre, où pour tout nœud, la différence de hauteur entre le fils droit et le fils gauche soit dans $\{-1,0,1\}$ (arbres H-équilibrés).

Pour les exemples, on utilise la structure d'arbre définie par union, introduite précédemment, où l'étiquette est composée du triplet (clé, valeur associée, information d'équilibre), les clés étant ordonnées par la relation d'ordre générique (ordre <= par défaut) sur le type des clés.

```
let a = N(N(N(F,(2,"v2",1),F), (5,"v5",1),F), (10,"v10",2), F);; (* ABRdd, non AVL *)
```

L'insertion et la suppression dans un ABRdd se fait en mettant à jour l'information d'équilibre, et si en plus on veut conserver une qualité d'AVL, il faut modifier (par rotations) la structure de l'arbre de façon à ce que l'information d'équilibre soit ramenée dans $\{-1,0,1\}$.

3.1 Requêtes et parcours dans un ABRdd ou un AVL

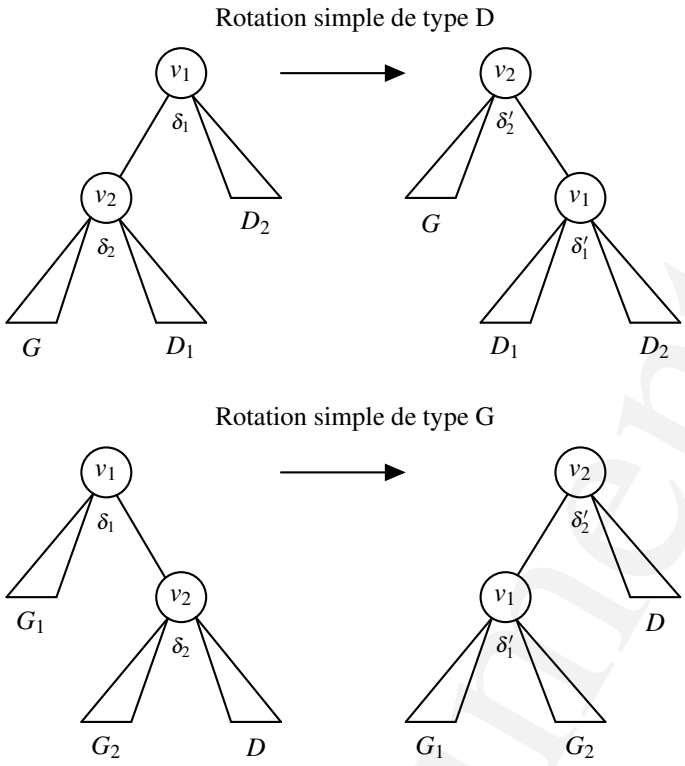
L'information d'équilibre, destinée à permettre le rétablissement de l'équilibre et la conservation d'une structure d'AVL après insertion et suppression, n'intervient pas dans le déroulement des requêtes ni des parcours ordinaires.

On utilisera les fonctions (adaptées à la forme de l'étiquette) définies dans le cas des ABR simples non décorés.

3.2 Rotations dans les ABR décorés (ABRdd)

Adaptation des rotations simples et doubles des ABR ordinaires (vues précédemment), complétées par la gestion des valeurs d'équilibre (inscrites sous les nœuds dans les figures).

3.2.1 Rotations simples dans un ABRdd



Les rotations simples ne concernent que les arbres (partie gauche de la figure) qui possèdent la structure adéquate.

– **Rotation S D** :
On appelle g le fils gauche de la racine de l'arbre initial et d' le fils droit de la racine de l'arbre après rotation.

- $\underline{\text{Si } \delta_2 > 0}$, $\begin{cases} |g| = 1 + |G| = 1 + \delta_2 + |D_1| \\ \delta_1 = |g| - |D_2| \end{cases}$ d'où $\begin{cases} |D_1| = |g| - 1 - \delta_2 \\ |D_2| = |g| - \delta_1 \end{cases}$ et $\delta'_1 = -1 + \delta_1 - \delta_2$.
  - $\underline{\text{Si } \delta'_1 \geqslant 0}$, $|d'| = 1 + |D_1|$ d'où $\delta'_2 = |G| - |d'| = |G| - 1 - |D_1|$, soit $\delta'_2 = -1 + \delta_2$
  - $\underline{\text{Si } \delta'_1 < 0}$, $\begin{cases} |d'| = 1 + |D_2| \\ |g| = 1 + |G| \end{cases}$ d'où $\delta'_2 = |G| - |d'| = |g| - 1 - 1 - |D_2|$, soit $\delta'_2 = -2 + \delta_1$
- $\underline{\text{Si } \delta_2 \leqslant 0}$, $\begin{cases} |g| = 1 + |D_1| \\ \delta_1 = |g| - |D_2| \end{cases}$ d'où $\begin{cases} |D_1| = |g| - 1 \\ |D_2| = |g| - \delta_1 \end{cases}$ et $\delta'_1 = -1 + \delta_1$.
  - $\underline{\text{Si } \delta'_1 \geqslant 0}$, $|d'| = 1 + |D_1|$ d'où $\delta'_2 = |G| - |d'| = |G| - 1 - |D_1|$, soit $\delta'_2 = -1 + \delta_2$
  - $\underline{\text{Si } \delta'_1 < 0}$, $\begin{cases} |d'| = 1 + |D_2| \\ |g| = 1 + |D_1| = 1 + |G| - \delta_2 \end{cases}$ d'où $\delta'_2 = |G| - |d'| = |g| - 1 + \delta_2 - 1 - |D_2|$, soit $\delta'_2 = -2 + \delta_1 + \delta_2$

En résumé :

$\underline{\text{Rotation S D}}$ (cas général des **ABRdd**) :

- $\underline{\text{Si } \delta_2 > 0}$, $\boxed{\delta'_1 = -1 + \delta_1 - \delta_2}$ $\begin{cases} \underline{\text{Si } \delta'_1 \geqslant 0}\,, & \boxed{\delta'_2 = -1 + \delta_2} \\ \underline{\text{Si } \delta'_1 < 0}\,, & \boxed{\delta'_2 = -2 + \delta_1} \end{cases}$
- $\underline{\text{Si } \delta_2 \leqslant 0}$, $\boxed{\delta'_1 = -1 + \delta_1}$ $\begin{cases} \underline{\text{Si } \delta'_1 \geqslant 0}\,, & \boxed{\delta'_2 = -1 + \delta_2} \\ \underline{\text{Si } \delta'_1 < 0}\,, & \boxed{\delta'_2 = -2 + \delta_1 + \delta_2} \end{cases}$

En réalité, pour les **AVL**, la **rotation S D** ne sera utilisée que lorsque $\begin{cases} \delta_1 > 0 \\ \delta_2 \in \{-1, 0, 1\} \end{cases}$,
ce qui supprime le dernier cas et permet d'obtenir une formule unique :

$\underline{\text{Rotation S D}}$ (cas réduit pour **AVL**) : $\begin{cases} \boxed{\delta'_1 = -1 + \delta_1 - \max(\delta_2, 0)} \\ \boxed{\delta'_2 = -1 + \delta_2 + \min(-1 + \delta_1 - \delta_2, 0)} \end{cases}$

– **Rotation S G** : on utilise la même argumentation, ce qui donne un résultat analogue (symétrique) :

$\underline{\text{Rotation S G}}$ (cas général des **ABRdd**) :

- $\underline{\text{Si } \delta_2 < 0}$, $\boxed{\delta'_1 = 1 + \delta_1 - \delta_2}$ $\begin{cases} \underline{\text{Si } \delta'_1 \leqslant 0}\,, & \boxed{\delta'_2 = 1 + \delta_2} \\ \underline{\text{Si } \delta'_1 > 0}\,, & \boxed{\delta'_2 = 2 + \delta_1} \end{cases}$
- $\underline{\text{Si } \delta_2 \geqslant 0}$, $\boxed{\delta'_1 = 1 + \delta_1}$ $\begin{cases} \underline{\text{Si } \delta'_1 \leqslant 0}\,, & \boxed{\delta'_2 = 1 + \delta_2} \\ \underline{\text{Si } \delta'_1 > 0}\,, & \boxed{\delta'_2 = 2 + \delta_1 + \delta_2} \end{cases}$

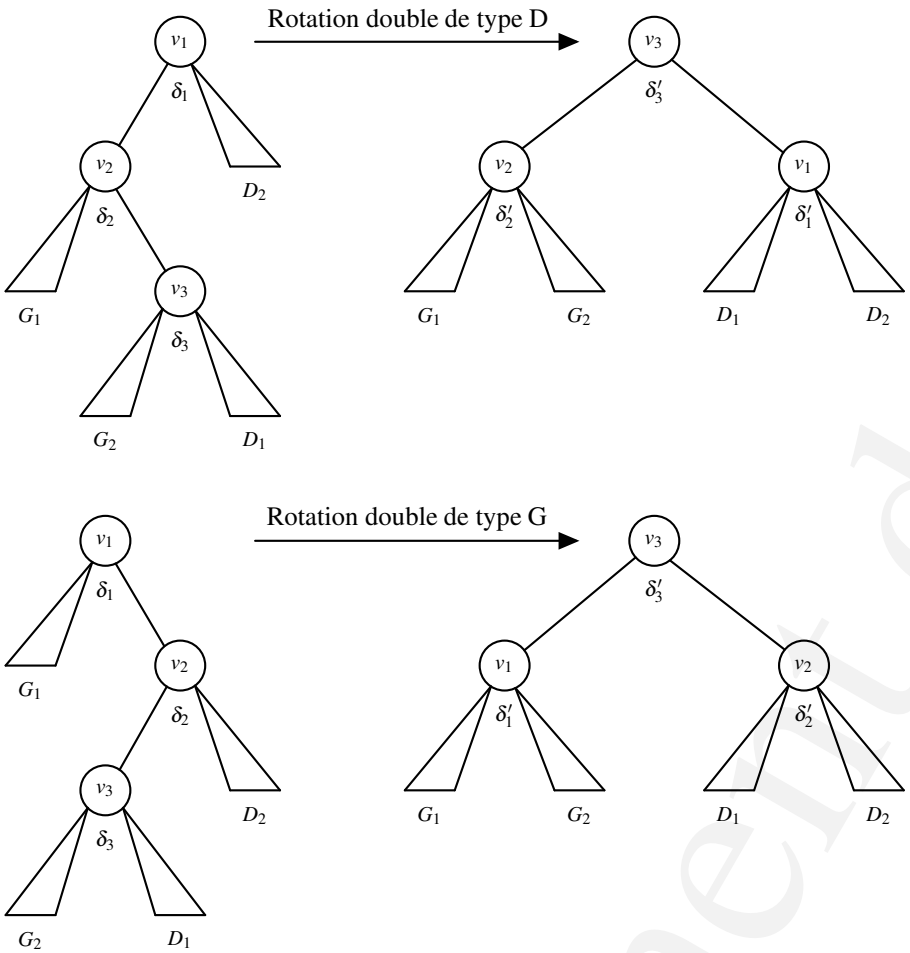
En réalité, pour les **AVL**, la **rotation S G** ne sera utilisée que lorsque $\begin{cases} \delta_1 < 0 \\ \delta_2 \in \{-1, 0, 1\} \end{cases}$,
ce qui supprime le dernier cas et permet d'obtenir une formule unique :

$\underline{\text{Rotation S G}}$ (cas réduit pour **AVL**) : $\begin{cases} \boxed{\delta'_1 = 1 + \delta_1 - \min(\delta_2, 0)} \\ \boxed{\delta'_2 = 1 + \delta_2 + \max(1 + \delta_1 - \delta_2, 0)} \end{cases}$

```
let rotationSD = function                (* cas général des ABRdd *)
  | F                  -> F
  | N ( F, _, _ ) as a -> a
  | N ( N(fG, (c2,v2,d2), fD1), (c1,v1,d1), fD2) ->
        let d1' = if d2 >= 0 then -1+d1-d2 else -1+d1
        in let d2' = if d1' >= 0 then -1+d2
                     else if d2 >= 0 then -2+d1 else -2+d1+d2
        in N(fG, (c2,v2,d2'), N(fD1, (c1,v1,d1'), fD2))

and rotationSG = function                (* cas général des ABRdd *)
  | F                  -> F
  | N ( _, _, F ) as a -> a
  | N ( fG1, (c1,v1,d1), N( fG2, (c2,v2,d2), fD)) ->
         let d1' = if d2 <= 0 then 1+d1-d2 else 1+d1
         in let d2' = if d1' <= 0 then 1+d2
                      else if d2 <= 0 then 2+d1 else 2+d1+d2
         in N( N( fG1, (c1,v1,d1'), fG2), (c2,v2,d2'), fD)
;;
(* rotationSD : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun>
   rotationSG : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun> *)
```

3.2.2 Rotations doubles dans un ABRdd



Les rotations doubles ne concernent que les arbres (partie gauche de la figure) qui possèdent la structure adéquate.

A cause du maintien de l'information d'équilibre, les rotations doubles dans les **ABRdd** ne réduisent pas à la composée de rotations simples d'**ABR**. Il faut adapter la valeur de δ_1 (valeur d'équilibre de la racine) après la première rotation simple (rotation droite simple du fils gauche ou rotation simple gauche du fils droit, voir étape intermédiaire dans la figure de la page 2).

– **Rotation D D** :

- On applique d'abord une **rotation S G** (cas général des **ABRdd**) sur le fils gauche.
- On calcule la nouvelle valeur de déséquilibre de la racine, en évaluant l'accroissement de hauteur du à la rotation **S G** du fils gauche (le fils droit étant inchangé) :
  - Si $\delta_2 \geqslant 0$ (G_1 est la fin d'une branche la plus longue), la hauteur du fils gauche a augmenté de 1, donc on remplace δ_1 par $\delta_1 + 1$
  - Si $\delta_2 < 0$
    - si $\delta_3 \geqslant 0$ (G_2 est la fin d'une branche la plus longue), et la hauteur du fils gauche est inchangée, donc δ_1 est inchangé.
    - si $\delta_3 < 0$ (D_1 est la fin de la <u>seule</u> branche la plus longue)
      - Si $\delta_2 < -1$, la hauteur du fils gauche a diminué de 1, donc on remplace δ_1 par $\delta_1 - 1$
      - si $\delta_2 = -1$, la hauteur du fils gauche est inchangée, donc δ_1 est inchangé.

En résumé :

• si $\delta_2 \geqslant 0$,	on incrémente δ_1,
• si $\delta_2 < -1$ et $\delta_3 < 0$,	on décrémente δ_1,
• sinon,	δ_1 reste inchangé.

- Puis, on applique la **rotation S D** (cas général des **ABRdd**) sur la racine.

<u>Rotation D D</u> (cas général des **ABRdd**) : composition, avec modification intermédiaire de δ_1 .

Remarque : Dans le cas des **AVL**, la **rotation D D** ne sera appliquée que lorsque $\delta_1 > 1$ ($\delta_1 = 2$) , $\delta_2 < 0$ ($\delta_2 = -1$). On utilisera donc les formules réduites de rotation simple obtenues précédemment
et, comme en plus on aura $\delta_3 \geqslant 0$, il n'y aura pas besoin de la modification intermédiaire de δ_1 !.

<u>Rotation D D</u> (cas réduit pour **AVL**) :
composition (cas réduit ?), sans modification intermédiaire de δ_1 .

– **Rotation D G** :

- On applique d'abord une **rotation S D** (cas général des **ABRdd**) sur le fils droit.
- On calcule la nouvelle valeur de déséquilibre de la racine, en évaluant l'accroissement de hauteur du à la rotation **S D** du fils droit (le fils gauche étant inchangé) :
  - Si $\delta_2 \leqslant 0$ (D_2 est la fin d'une branche la plus longue), la hauteur du fils droit a augmenté de 1, donc on remplace δ_1 par $\delta_1 - 1$
  - Si $\delta_2 > 0$
    - si $\delta_3 \leqslant 0$ (D_1 est la fin d'une branche la plus longue), et la hauteur du fils droit est inchangée, donc δ_1 est inchangé.
    - si $\delta_3 > 0$ (G_2 est la fin de la <u>seule</u> branche la plus longue)
      - si $\delta_2 > 1$, la hauteur du fils droit a diminué de 1, donc on remplace δ_1 par $\delta_1 + 1$
      - si $\delta_2 = 1$, la hauteur du fils droit est inchangée, donc δ_1 est inchangé.

En résumé :

• si $\delta_2 \leqslant 0$,	on décrémente δ_1,
• si $\delta_2 > 1$ et $\delta_3 > 0$,	on incrémente δ_1,
• sinon,	δ_1 reste inchangé.

- Puis, on applique la **rotation S G** (cas général des **ABRdd**) sur la racine.

<u>Rotation D G</u> (cas général des **ABRdd**) : composition, avec modification intermédiaire de δ_1 .

Remarque : Dans le cas des **AVL**, la **rotation D G** ne sera appliquée que lorsque $\delta_1 < -1$ ($\delta_1 = -2$) , $\delta_2 > 0$ ($\delta_2 = 1$). On utilisera donc les formules réduites de rotation simple obtenues précédemment et, comme en plus on aura $\delta_3 \leqslant 0$, il n'y aura pas besoin de la modification intermédiaire de δ_1 !.

<u>Rotation D G</u> (cas réduit pour **AVL**) :
composition (cas réduit ?), sans modification intermédiaire de δ_1 .

```
let rotationDD = function              (* composition, cas général ABRdd *)
 | F -> F
 | N (fg, (c1,v1,d1), fD2) as a -> match fg with
   | F -> a
   | N (_, (_,_,d2), fgD) -> match fgD with
     | F -> a
     | N (_, (_,_,d3), _) -> let nd1 = if d2 >= 0 then d1+1
                                      else if d2 < -1 & d3 < 0 then d1-1 else d1
                             in rotationSD ( N(rotationSG fg, (c1,v1,nd1), fD2) )

and rotationDG = function              (* composition, cas général ABRdd *)
 | F -> F
 | N (fG1, (c1,v1,d1), fd) as a -> match fd with
   | F -> a
   | N (fdG, (_,_,d2), _) -> match fdG with
     | F -> a
     | N (_, (_,_,d3), _) -> let nd1 = if d2 <= 0 then d1-1
                                      else if d2 > 1 & d3 > 0 then d1+1 else d1
                             in rotationSG ( N(fG1, (c1,v1,nd1), rotationSD fd) )
;;
(* rotationDD : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun>
   rotationDG : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun> *)
```

Si on se limite au cas des **AVL**, on pourra écrire :

```
let rotationDD = function               (* composition, cas réduit pour AVL *)
  F -> F | N (fg, e, fd) -> rotationSD (N ( rotationSG fg, e, fd))
and rotationDG = function               (* composition, cas réduit pour AVL *)
  F -> F | N (fg, e, fd) -> rotationSG (N (fg, e, rotationSD fd))
;;
(* rotationDD : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun>
   rotationDG : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun>  *)
```

3.3 Insertion en feuille dans un ABRdd, dans un AVL

L'insertion, comme feuille, d'un nouvel élément (clé, valeur associée) dans un **ABRdd**, de racine non vide peut augmenter (de $+1$) ou laisser inchangée la hauteur du fils dans lequel on réalise l'insertion.

- Si on se limite au cas d'un **ABRdd**, connaissant l'information d'accroissement de hauteur du fils ($dh = 0$ ou $dh = 1$), il suffit de l'ajouter (cas du fil gauche) ou de la retrancher (cas du fils droit) à l'information d'équilibre du nœud racine pour que l'arbre garde la qualité d'**ABRdd**.
- Si on exige en plus la qualité d'**AVL**, une fois la cohérence de l'information d'équilibre de la racine rétablie (qualité d'**ABRdd**), les fils ayant la qualité d'**AVL**, la valeur d'équilibre de la racine peut prendre les valeurs $-2, -1, 0, 1, +2$. Si la valeur d'équilibre est -2 ou $+2$, il faut effectuer une rotation pour rétablir la qualité d'**AVL** :
  - fils équilibré ou déséquilibré dans un sens favorable : rotation simple (gauche ou droite)
  - fils déséquilibré dans un sens défavorable : rotation double (gauche ou droite)

On rappelle qu'il y a deux stratégies pour l'insertion en feuille dans les **ABR** larges (la question ne se pose pas pour les **ABR** stricts) :

- Insertion en feuille réelle : On descends dans l'arbre jusqu'à ce que l'on rencontre une feuille où l'on place l'élément à insérer.
  Dans cette stratégie, le dernier entré peut être masqué par un élément plus ancien et de même clé, ce qui fait que, pour une même clé, les premiers visibles sont les plus anciens dans l'arbre.
- Insertion avec repoussement en feuille : Au cours de la descente dans l'arbre, en cas d'égalité de clé, l'élément nouveau prend la place de l'ancien et l'ancien est réinséré, selon le même principe, plus profondément, ce qui fait que, pour une même clé, les premiers visibles sont les plus récents dans l'arbre.

Dans la suite, on ne considère que l'insertion en feuille réelle (l'insertion avec repoussement en feuille s'en déduit par une très légère modification).

3.3.1 Insertion en feuille dans un ABRdd

On note $\delta(a)$ la valeur du déséquilibre de la racine d'un arbre a et, pour un arbre a non vide, $\varepsilon(a)$ l'étiquette de la racine de a, $\mathscr{G}_a$ le fils gauche de a et $\mathscr{D}_a$ le fils droit de a.

Soit $\Delta(v,a)$ ($\in\{0,1\}$) l'accroissement de hauteur résultant de l'insertion dans un arbre a du nouvel élément $x=(c_x,v_x)$ (de clé c_x et de valeur associée v_x).

- si $a=\emptyset$, $\quad\Delta(v,a)=1$
- si $c_x=$ `cle` $\varepsilon(a)$, $\quad a$ est inchangé, $\Delta(v,a)=0\quad$ (pour un **ABR** strict)
- si $c_x\leqslant$ `cle` $\varepsilon(a)$ (insertion de x à gauche, dans $\mathscr{G}_a$), $\begin{cases}\text{si } \delta(a)\geqslant 0, \Delta(v,a)=\Delta\big(v,\mathscr{G}(a)\big)\\ \text{si } \delta(a)<0, \Delta(v,a)=0\end{cases}$
- si $c_x>$ `cle` $\varepsilon(a)$ (insertion de x à droite, dans $\mathscr{D}_a$), $\begin{cases}\text{si } \delta(a)\leqslant 0, \Delta(v,a)=\Delta\big(v,\mathscr{D}(a)\big)\\ \text{si } \delta(a)>0, \Delta(v,a)=0\end{cases}$

On en déduit l'insertion d'un élément (clé, valeur) dans un **ABRdd**, à l'aide d'une fonction récursive auxiliaire qui renvoie l'arbre **ABRdd** obtenu par l'insertion et l'accroissement de hauteur du à l'insertion :

```
let inserer_ABRdd ((cx,vx) as x) a = fst (aux x a)
 where rec aux e = function
    F -> N(F, (cx,vx,0), F), 1
(*  | N(_, e, _) as a when cx = cle e   -> a, 0  *)   (* si ABR strict *)
  | N (fg, (c1,v1,d1), fd) when cx <= c1  ->
        let nfg, dh = aux x fg in N(nfg, (c1,v1,d1+dh), fd), (if d1 >= 0 then dh else 0)
  | N (fg, (c1,v1,d1), fd) (* c > c1 *)  ->
        let nfd, dh = aux x fd in N(fg, (c1,v1,d1-dh), nfd), (if d1 <= 0 then dh else 0)
;;
(* inserer_ABRdd : 'a * 'b -> ('a * 'b * int) arbre -> ('a * 'b * int) arbre =  <fun> *)
```

La complexité de l'insertion dans un **ABRdd** de hauteur h est un $O(h)$.

3.3.2 Rééquilibrage de la racine d'un AVL

On ne fait l'équilibrage que lorsque le déséquilibre δ_1 de la racine est en dehors des valeurs autorisées pour la qualité **AVL** ($\{-1,0,1\}$), c'est à dire, puisque les accroissements de hauteur seront de 0 ou $+1$ (pour l'insertion) ou de 0 ou -1 (pour la suppression), lorsque $\begin{cases}\delta_1=2 & \text{(rotation droite, simple ou double)}\\ \delta_1=-2 & \text{(rotation gauche, simple ou double)}\end{cases}$, les fils étant équilibrés (de qualité **AVL**).

Dans ces conditions, le choix d'une rotation simple ou double pour rétablir l'équilibre, dépends de la valeur (δ_2) de déséquilibre du fils **AVL** modifié :

- Si la plus grande hauteur est à "l'extérieur", soit $\begin{cases}\delta_1=2\\ \delta_2\geqslant 0\ (=+1)\end{cases}$ ou $\begin{cases}\delta_1=-2\\ \delta_2\leqslant 0\ (=-1)\end{cases}$, on fait une rotation simple gauche ou (resp.) droite.
- Si la plus grande hauteur est à "l'intérieur", soit $\begin{cases}\delta_1=2\\ \delta_2\leqslant 0\ (=-1)\end{cases}$ ou $\begin{cases}\delta_1=-2\\ \delta_2\geqslant 0\ (=+1)\end{cases}$, on fait une rotation double gauche ou (resp.) droite.

```
let equilibrage_AVL = function                                        (* fils AVL *)
  | (N (fg, (_,_,d1), _)) as a when d1 > 1 ->          (* RSD ou RDD *)
        begin match fg with
                | N (_, (_,_,d2), _) when d2 >= 0 -> rotationSD a
                | _                               -> rotationDD a
        end
  | (N (_, (_,_,d1), fd)) as a when d1 < -1  ->         (* RSG ou RDG *)
        begin match fd with
                | N (_, (_,_,d2),  _) when d2 <= 0 -> rotationSG a
                | _                                -> rotationDG a
        end
  | a -> a    (* non déséquilibré *)
;;
(* equilibrage_AVL : ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun> *)
```

3.3.3 Insertion en feuille dans un AVL

On reprend la structure de la fonction `inserer_ABRdd`, avec la fonction auxiliaire `aux` telle que `aux e a` renvoie l'arbre déduit de a par insertion de e dans a et l'accroissement de hauteur de a, modifiée ainsi :

Hors les cas élémentaires `Vide` (et $e = \varepsilon(a)$ pour un **ABR** strict), après insertion dans un fils,

- si le nouveau déséquilibre est hors limite (ce ne peut être que 2 ou -2), il y a eu accroissement de hauteur ($dh = +1$) du fils et le rééquilibrage permet de "gommer" cet accroissement de hauteur ;
- sinon, il n'y a pas à équilibrer, mais
  - si l'arbre penche strictement du coté de l'insertion, on reporte l'accroissement de hauteur ($dh = 0$ ou 1) du fils (le fils initial était la branche la plus longue)
  - sinon, il n'y a pas d'accroissement de hauteur (le fils initial n'était pas la branche la plus longue).

Lorsque a est de qualité **AVL**, `aux e a` renvoie maintenant le couple formé par l'arbre de qualité **AVL** obtenu par insertion de `e` = (clé, valeur) dans `a` et l'accroissement de hauteur qui en résulte.

```
let inserer_AVL ((cx,vx) as x) a = fst (aux x a)
 where rec aux x = function
    F -> N (F, (cx,vx,0), F), 1

(*  | N (_, e, _) as a when cx = cle e -> a, 0   (* si ABR strict *) *)

  | N (fg, (c,v,d), fd) when cx <= c ->
          let nfg, dh = aux x fg
          in let d' = d+dh
          in if d' = 2 then equilibrage_AVL (N (nfg, (c,v,d'), fd)), 0
                       else N (nfg, (c,v,d'), fd), if d' > 0 then dh else 0

  | N (fg, (c,v,d), fd) (* when cx > c *) ->
          let nfd, dh = aux x fd
          in let d' = d-dh
          in if d' = -2 then equilibrage_AVL (N (fg, (c,v,d'), nfd)), 0
                        else N (fg, (c,v,d'), nfd), if d' < 0 then dh else 0
;;
(* inserer_AVL : 'a * 'b -> ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun> *)
```

La complexité de l'insertion dans un **AVL** de hauteur h est un $O(h)$.

3.4 Insertion à la racine (dans un ABRdd)

L'insertion en feuille garde les informations anciennes près de la racine, ce qui fait que l'accès aux informations récentes est moins rapide que l'accès aux informations anciennes. Cela peut être, selon les cas, un avantage ou un inconvénient ....

L'insertion à la racine conduit au contraire à un accès plus rapide aux informations récentes et à un accès moins rapide aux informations anciennes, ce qui présente les avantages et inconvénients contraires de l'insertion en feuille.

L'insertion à la racine n'a pas de sens pour un **AVL**, et, pour un **ABRdd**, cela n'a pas beaucoup d'intérêt car cela risque de déséquilibrer fortement l'arbre, en particulier lorsque la clé à insérer n'a pas une valeur proche de la médiane des valeurs de clés déjà présentes dans l'arbre.

Si on veut à tout prix réaliser une insertion à la racine dans un **ABRdd**,

1. on part de l'insertion (classique) à la racine, par coupe, dans un **ABR** quelconque,
2. on adapte cette insertion au cas des **ABR** décorés par le déséquilibre (**ABRdd**) en faisant en sorte que la coupe renvoie à la fois des **ABRdd** et les informations de hauteur des coupes (en un seul parcours d'arbre), pour pouvoir redéfinir la valeur de déséquilibre de la nouvelle racine.

3.5 Suppression, selon une valeur de clé, dans un AVL

La suppression d'un nœud se fait selon le principe général de la suppression dans un **ABR** :

- suppression à gauche ou à droite de la racine d'un sous-arbre, selon la valeur de clé.
- suppression de la racine d'un sous arbre :
  - remplacement par l'autre fils si (au moins) un des fils est vide,
  - et si les deux fils ne sont pas vide, remplacement par le minimum supprimé du fils droit (ou par le maximum supprimé du fils gauche),

que l'on adapte au cas des **AVL**, comme on l'a fait pour l'insertion.

Cette méthode fait remonter à la racine des éléments qui sont enfouis profondément dans l'arbre !

La suppression dans un **AVL** utilise la fonction auxiliaire `aux` telle que `aux e a` renvoie l'arbre **AVL** déduit de `a` par suppression de `e` dans `a` et l'"accroissement" `dh` de hauteur qui en résulte (`dh` $\leqslant 0, = 0$ ou $= -1$) :

```
let supprimer_AVL cx a = fst (aux cx a)              (* dh <= 0 *)
 where rec aux cx = function
  | F                                -> F, 0
  | N (fg, (c,v,d), fd) when cx < c  -> let nfg, dh = aux cx fg
          in let d' = d+dh
          in if d' = -2 then equilibrage_AVL (N(nfg, (c,v,d'), fd)), dh
                        else N (nfg, (c,v,d'), fd), if d' < 0 then 0 else dh
  | N (fg, (c,v,d), fd) when cx > c  -> let nfd, dh = aux cx fd
          in let d' = d-dh
          in if d' = 2  then equilibrage_AVL (N (fg, (c,v,d'), nfd)), dh
                        else N (fg, (c,v,d'), nfd), if d' > 0 then 0 else dh
  | a  (* when cx = c *)              -> supprimer_racine_AVL a

where rec supprimer_racine_AVL = function
   | F                   -> F, 0
   | N (fg, _, F)        -> fg, -1
   | N (F, _, fd)        -> fd, -1

   | N (fg, (_,_,d), fd) -> let ((nc,nv,_), nfg, dh) = maxi_sousAVLg_dh fg
                  in let d' = d+dh
                  in if d' = -2 then equilibrage_AVL (N (nfg, (nc,nv,d'), fd)), dh
                     else N (nfg, (nc, nv, d'), fd), if d' < 0 then 0 else dh

(* | N (fg, (_,_,d), fd) -> let ((nc,nv,_), nfd, dh) = mini_sousAVLd_dh fd
                  in let d' = d-dh
                  in if d' = +2 then equilibrage_AVL (N (fg, (nc,nv,d'), nfd)), dh
                     else N (fg, (nc,nv,d'), nfd), if d' > 0 then 0 else dh
*)
and maxi_sousAVLg_dh = function     (* maxi, sous AVL gauche et variation hauteur (<= 0) *)
  | F                   -> failwith "y a comme un défaut"
  | N (fg, v, F)        -> (v, fg,-1)
  | N (fg, (c,v,d), fd) -> let (m, nfd, dh) = maxi_sousAVLg_dh fd
              in let d' = d-dh
              in if d' = 2 then (m, (equilibrage_AVL (N (fg, (c,v,d'), nfd))), dh)
              else (m, N (fg, (c,v,d'), nfd), (if d' > 0 then 0 else dh) )
and mini_sousAVLd_dh = function     (* mini, sous AVL droit et variation hauteur (<= 0) *)
  | F                   -> failwith "y a comme un défaut"
  | N (F, v, fd)        -> (v, fd,-1)
  | N (fg, (c,v,d), fd) -> let (m, nfg, dh) = mini_sousAVLd_dh fg
                           in let d' = d+dh
              in if d' = -2 then (m, (equilibrage_AVL (N (nfg, (c,v,d'), fd))), dh)
              else (m, N (nfg, (c,v,d'), fd), (if d' < 0 then 0 else dh) )
;;
(* supprimer_AVL : 'a -> ('a * 'b * int) arbre -> ('a * 'b * int) arbre = <fun> *)
```

La complexité de la suppression d'un élément dans **AVL** de hauteur h est encore un $O(h)$.

4 Arbres bicolores (information)

Un *arbre bicolore* (ABRB) est un arbre binaire de recherche dont les nœuds sont colorés d'une couleur parmi deux selon des règles particulières énoncées ci-dessous. Si le rouge et le noir sont classiquement utilisés, nous prendront ici le blanc et noir. On choisira d'autre part de ne pas étiqueter les feuilles.

Un arbre bicolore doit satisfaire aux contraintes suivantes (en plus d'être un arbre binaire de recherche) :

(A-1) Un nœud est soit blanc, soit noir.

(A-2) Une feuille est noire (convention)

(A-3) La racine est noire

(A-4) Le père d'un nœud blanc est noir.

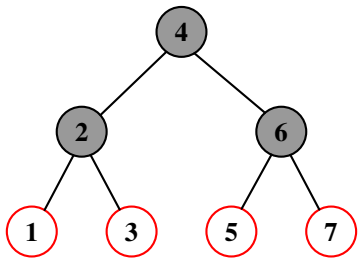
(A-5) Tous les chemins partant d'un nœud donné et se terminant à une feuille contiennent le même nombre de nœuds noir (sans prendre en compte le nœud de départ).

On nomme *hauteur noire* d'un nœud n appartenant à un arbre bicolore (on la note $hn(n)$) le nombre de nœuds noirs sur les chemins partant de ce nœud et aboutissant à une feuille (sans prendre en compte n). Cette fonction est bien définie grâce à (A-5). On nomme hauteur noire d'un arbre bicolore la hauteur noire de sa racine.

Les arbres bicolores sont des arbres binaires de recherche équilibrés dans lesquels les fonctions d'insertion et de suppression rétablissent les propriétés de coloriage, en effectuant des rotations.

4.1 Exemples

Les arbres ~~Rouges~~ Blancs et Noir peuvent être utilisés pour la représentation du type ensemble (à l'aide de l'ordre générique sur le type des éléments). Par exemple, $\{1,2,3,4,5,6,7\}$ se représente par l'arbre bicolore :



Arbre ~~rouge~~ blanc et noir (les feuilles, noires, ne sont pas représentées).

Exercice : Dessiner (à la main) un arbre bicolore dont les étiquettes sont les éléments de $\{1,2,3,4,5,6,7,8,9,10\}$ (on pourra dessiner un arbre binaire de recherche de hauteur minimale et le colorier ensuite).

Exemples de structure pour un arbre bicolore :

- Concours CCS 2000 : les nœuds sont des enregistrements à champs mutables, c'est compliqué ...

```
type Couleur = Blanc | Noir;;
type Noeud = { mutable etiquette : int;          mutable couleur : Couleur;
               mutable gauche : ArbreBinaire;    mutable droit : ArbreBinaire;
               mutable pere : ArbreBinaire ref }
and ArbreBinaire =   ArbreVide
                   | Pointeur of Noeud;;
```

- Concours CCP 2008 : beaucoup plus simple ...

```
type couleur = Blanc | Gris;;
type arbre = Vide | Noeud of couleur * arbre * int * arbre;;
```

4.2 Rotations simples et doubles dans un Arbre Rouge et Noir

4.3 Insertion (en feuille) dans un Arbre Rouge et Noir

4.4 Suppression, selon une valeur de clé, dans un Arbre Rouge et Noir

$< \mathcal{FIN} >$