

Corrigé du TD sur les arbres rouge et noir

Question 1.

```
let rec verif3 arbre = match arbre with
Vide->true
|Noeud(Rouge,Noeud(Rouge,_,_,_),_,_)>false
|Noeud(Rouge,_,_,Noeud(Rouge,_,_,_))>false
|Noeud(_,g,_,d)>verif3 g && verif3 d;;
```

Question 2.

```
let rec hauteur_noire arbre = match arbre with
Vide->0
|Noeud(couleur,g,x,d)>let hg=hauteur_noire g and hd=hauteur_noire d
in
if hg<>hd then failwith "hauteur édifferente" else
    if couleur=Rouge then hd else hd+1;;
```

Question 3.

```
let verif arbre = let h=hauteur_noire arbre in verif3 arbre;;
```

Question 4.

```
let rotation_d arbre = match arbre with
Noeud(cy,Noeud(cx,alpha,x,beta),y,gamma)>Noeud(cx,alpha,x,Noeud(cy,beta,y,gamma))
|_>arbre;;

let rotation_g arbre = match arbre with
Noeud(cx,alpha,x,Noeud(cy,beta,y,gamma))>Noeud(cy,Noeud(cx,alpha,x,beta),y,gamma)
|_>arbre;;
```

Question 5.

```
let rotation_g_d arbre = match arbre with
Noeud(c,g,x,d)>rotation_d (Noeud(c,rotation_g g,x,d))
|_>arbre;;

let rotation_d_g arbre = match arbre with
Noeud(c,g,x,d)>rotation_g (Noeud(c,g,x,rotation_d d))
|_>arbre;;
```

Question 6.

```
let rec simple_insert arbre x = match arbre with
Vide->Noeud(Rouge,Vide,x,Vide)
|Noeud(c,g,y,d) when x<y->Noeud(c,simple_insert g x,y,d)
|Noeud(c,g,y,d)>Noeud(c,g,y,simple_insert d x);
```

Question 7.

Le deuxième cas se résout avec une rotation droite-gauche, le troisième avec une rotation gauche-droite et le dernier avec une rotation gauche. Il est ensuite nécessaire de colorier correctement les noeuds du sous-arbre obtenu : dans tous les cas, on aura un arbre de racine rouge (x_1 , x_2 ou x_3) ayant ses deux fils noirs.

Question 8.

```
let racine_noire arbre = match arbre with
Vide->Vide
|Noeud(_,g,x,d)>Noeud(Noir,g,x,d);;
```

Question 9.

Comme dans tous les cas de figure, le coloriage final consiste à avoir la racine en rouge et ses deux fils en noir, on écrit un programme auxiliaire réalisant ce coloriage, que l'on applique après avoir effectué la rotation appropriée, déterminée à l'aide d'un filtrage (un peu lourd...).

```
let conflit arbre =
let colorie a = match a with
| Vide->Vide
| Noeud(c,g,x,d)->Noeud(c,racine_noire g,x,racine_noire d)
in
match arbre with
Noeud(Noir,Noeud(Rouge,Noeud(Rouge,t1,x1,t2),x2,t3),x3,t4)-> colorie (rotation_d arbre)
| Noeud(Noir,t1,x1,Noeud(Rouge,Noeud(Rouge,t2,x2,t3),x3,t4))-> colorie (rotation_d_g arbre)
| Noeud(Noir,Noeud(Rouge,t1,x1,Noeud(Rouge,t2,x2,t3)),x3,t4)-> colorie (rotation_g_d arbre)
| Noeud(Noir,t1,x1,Noeud(Rouge,t2,x2,Noeud(Rouge,t3,x3,t4)))-> colorie (rotation_g arbre)
|_->arbre;;
```

Question 10.

On commence par placer le nouveau noeud à la bonne place, et on fait des appels à `conflit` à chaque noeud que l'on a parcouru pour arriver à la nouvelle feuille. Ces appels sont placés au tout début de l'instruction, et seront donc effectués en ordre inverse, de bas en haut, lors du dépilement récursif du programme.

```
let rec insert_aux arbre x = match arbre with
Vide->Noeud(Rouge,Vide,x,Vide)
| Noeud(c,g,y,d) when x<y->conflit (Noeud(c,simple_insert g x,y,d))
| Noeud(c,g,y,d)->conflit (Noeud(c,g,y,simple_insert d x));;

let insert arbre x = racine_noire (insert_aux arbre x);;
```

Question 11.

Il est nécessaire de savoir répondre à ce type de question de façon efficace et rigoureuse. Pour cela, il faut essayer de deviner la complexité en regardant l'allure du programme, puis faire une récurrence pour le prouver. Écrire la récurrence en gardant la complexité comme inconnue peut également permettre de la déterminer.

Montrons par récurrence sur la hauteur de `arbre` que l'appel `insert_aux arbre x` nécessite au plus $2(h+1)$ opérations élémentaires.

Si `arbre` est de hauteur nulle, donc vide, on insère directement le nouveau noeud `x` : une seule opération élémentaire, la propriété est donc vraie pour $h = 0$.

Supposons la propriété vraie pour tous les arbres de hauteur $h - 1$, et soit `arbre` un arbre de hauteur h . Alors, le programme `insert_aux` effectue un appel à `conflit`, un appel récursif sur un arbre de hauteur au plus $h - 1$, et un appel au constructeur `Noeud`, soit au plus $2 + 2h = 2(h + 1)$ opérations élémentaires, ce qui conclut la preuve par récurrence.

Question 12.

```
let rec trouve a x = match a with
Vide->>false
| Noeud(_,g,y,d) when x=y->>true
| Noeud(_,g,y,d)->trouve g x || trouve d x;;
```

Question 13.

Commençons par montrer par récurrence sur la hauteur du noeud x que le sous-arbre de racine x contient au moins $2^{hn(x)} - 1$ ($hn(x)$ désigne la hauteur noire du noeud x) noeuds internes. Si x est de hauteur nulle, alors x est une feuille, et l'arbre de racine x contient au moins $2^0 - 1 = 0$ noeuds internes.

Soit maintenant x un noeud interne ayant deux fils. Chaque fils a une hauteur noire égale à $hn(x)$ ou $hn(x) - 1$, suivant sa couleur. Par hypothèse de récurrence, chaque arbre fils a au moins $2^{hn(x)-1} - 1$ noeuds internes. Donc, l'arbre de racine x a au moins $(2^{hn(x)-1} - 1) + (2^{hn(x)-1} - 1) + 1 = 2^{hn(x)} - 1$ noeuds internes, ce qui termine la preuve par récurrence.

Soit h la hauteur de l'arbre. La propriété 3 montre qu'au moins la moitié des noeuds sur n'importe quel chemin de la racine à une feuille sont noirs. La hauteur noire de la racine est donc au moins $h/2$, d'où : $n \geq 2^{h/2} - 1$. En passant au logarithme, on obtient la preuve du lemme 1.

Question 14.

Il suffit de considérer une liste déjà triée : dans ce cas, l'arbre binaire de recherche créé à partir de cette liste aura pour hauteur la longueur de la liste, et la recherche dans cet arbre sera totalement inefficace.