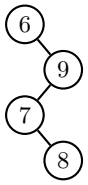
DS1 d'option informatique - Corrigé

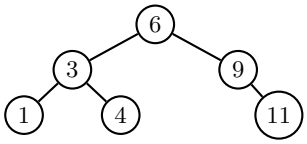
Question 1 : Si a est l'arbre vide $\emptyset$ alors sa profondeur est 0 sinon elle est égal au maximum de la profondeur de $\mathcal{G}(A)$ et de la profondeur de $\mathcal{D}(a)$ augmenté de 1.

$$|a| = \begin{cases} 0 & \text{si } a = \emptyset \\ 1 + \max(|\mathcal{G}(a)|, |\mathcal{D}(a))| & \text{sinon} \end{cases}$$

Question 2 : La profondeur de l'arbre est maximale si on a qu'un seul nœud à une profondeur k donné. Ainsi, au maximum un arbre ayant n noeuds sera de profondeur n.

La profondeur de l'arbre sera minimale si tous les niveaux de l'arbre sont pleins sauf éventuellement le dernier. On montre par récurrence que le niveau de profondeur k admet au plus 2^{k-1} nœuds et donc pour un arbre de profondeur minimal p, on a : $1 + 2 + 2^2 + \ldots + 2^{p-1} < n \leqslant 1 + 2 + 2^2 + \ldots + 2^p \iff 2^p \leqslant n \leqslant 2^{p+1} - 1 < 2^{p+1}$ d'où $p = \log_2(n)$.





Question 3 : La fonction auxiliaire a pour signature `aux : arbre -> bool * int` . La première composante du couple renvoyé est un booléen égal à `true` si l'arbre est valide et `false` sinon, et la deuxième composante est la hauteur de l'arbre.

```
let valider_decore a =
let rec aux ab = match ab with
  | Vide -> true,0
  | Noeud(d,fg,_,fd) -> let b1,h1 = aux fg
                        and b2,h2 = aux fd in
                              (b1 && b2 && d=h1-h2),(1+ max h1 h2)
in fst(aux a);;
```

La fonction ne parcourt qu'une fois l'arbre car pour chaque arbre, il y a juste un appel récursif à la fonction sur chacun des deux fils.

Question 4 : La fonction auxiliaire `aux` dans la fonction `valider_decore` est appelée autant de fois qu'il y a de nœuds donc la complexité est en $O(n)$ où n est le nombre de nœuds de l'arbre.

Question 5 : Par défintion, la profondeur d'un arbre qui n'est pas vide est égal au maximum des profondeurs des deux fils auquel on ajoute 1. Il suffit alors d'utiliser les décorations pour savoir quel est le fils le plus profond (celui de droite si le déséquilibre est négatif et celui de gauche sinon).

```
let profondeur a = match a
  | Vide -> 0
  | Noeud(d,fg,_,fd) -> if d > 0 then 1 + profondeur fg
                        else 1 + profondeur fd;;
```

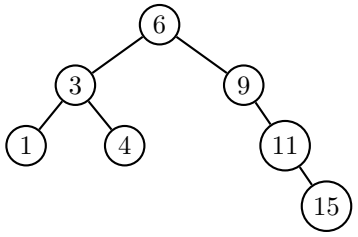
Question 6 : On utilise une fonction auxiliaire `aux : arbre -> bool * int * int` telle que `aux a` renvoie le triplet `b,min,max` où `b` est un booléen qui dit si l'arbre est un ABR, `min` est le minimum des étiquettes de l'arbre et `max` le maximum.

```
let rec valider_abr a =
let rec aux ab = match ab with
 Vide -> (true,0,0)
  | Noeud(_,Vide,e,Vide) -> true,e,e
  | Noeud(_,fg,e,Vide) -> let (b1,min1,max1) = aux fg in
      (b1 && (max1 < e)),min1, e
  | Noeud(_,Vide,e,fd) -> let (b2,min2,max2) = aux fd in
      (b2 && (min2 > e)), e, max2
  | Noeud(_,fg,e,fd) -> let (b1,min1,max1) = aux fg in
      let (b2,min2,max2) = aux fd in
      (b1 && b2 && (max1 < e) && (min2 > e)),min1, max2
in let b,_,_ = aux a in
  b;;
```

Question 7 : Question de cours

```
let rec rechercher_abr v a = match a with
 Vide -> false
  | Noeud(_,fg,e,fd) when v=e -> true
  | Noeud(_,fg,e,fd) -> if v < e then rechercher_abr v fg
        else rechercher_abr v fd;;
```

Question 8 : Dans le meilleur des cas, l'élément v cherché est à la racine de l'arbre, il n'y aura donc qu'un seul appel de la fonction `recherche_abr`. Dans le pire des cas, l'élément n'est pas dans l'arbre et il faut parcourir l'arbre de la racine jusqu'à une feuille de profondeur maximale. Dans ce cas le nombre d'appels à la fonction `recherche_abr` sera égal à la hauteur plus un.
Exemple : meilleur cas : rechercher 6 dans l'arbre suivant. cas le pire : rechercher 13 dans ce même arbre :



Question 9 :

$$\delta(v,a) = \begin{cases} \delta(v,\mathcal{G}(a)) & \text{si } v < \mathcal{E}(a) \text{ et } \Delta(a) \geqslant 0 \\ 0 & \text{si } v < \mathcal{E}(a) \text{ et } \Delta(a) < 0 \ (^*) \\ 0 & \text{si } v = \mathcal{E}(a) \\ \delta(v,\mathcal{D}(a)) & \text{si } v > \mathcal{E}(a) \text{ et } \Delta(a) \leqslant 0 \\ 0 & \text{si } v > \mathcal{E}(a) \text{ et } \Delta(a) > 0 \ (^*) \\ 1 & \text{si } a = \emptyset \end{cases}$$

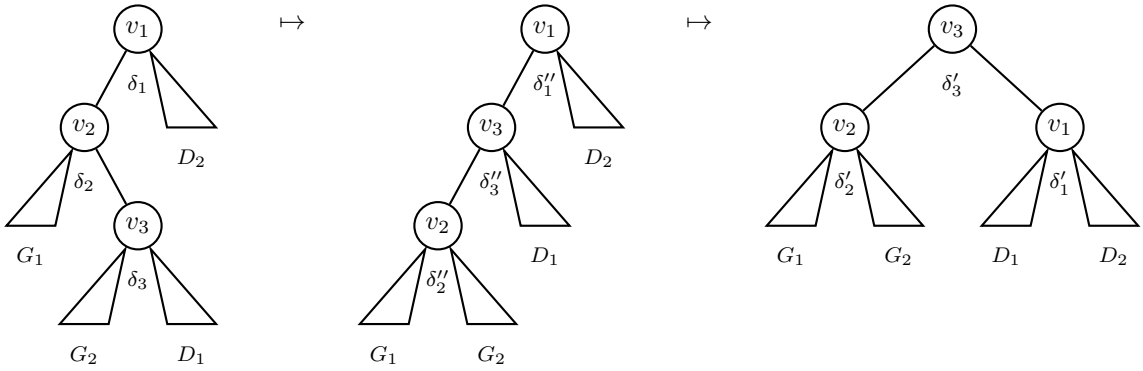
(*) Si l'arbre est déséquilibré vers la droite et que l'on insère à gauche, il n'y aura pas d'augmentation de hauteur, de même si le déséquilibre est à gauche et que l'on insère à droite.

Question 10 : On utilise une fonction auxiliaire `aux : arbre -> int * arbre` telle que `aux a` renvoie le couple `delta, ab` où `b` est l'arbre obtenu en insérant v dans `a` et `delta` est l'accroissement de la profondeur de l'arbre (comme défini à la question précédente). Notons que `v` est vu comme une variable globale par rapport à la fonction `aux`

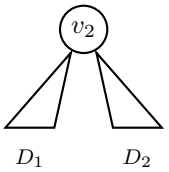
```
let inserer_abr v a =
let rec aux a = match a with
 Vide -> 1,Noeud(0,Vide,v,Vide)
  | (Noeud(d,fg,e,fd) as aa) when e=v -> 0,aa
  | Noeud(d,fg,e,fd) when v <e
                    -> let (d',fg') = aux fg in
                           if d >= 0 then d',Noeud(d+d',fg',e,fd)
                                     else 0,Noeud(d+d',fg',e,fd)
  | Noeud(d,fg,e,fd) (* when v>e *)
                    -> let (d',fd') = aux fd in
                           if d <= 0 then d',Noeud(d-d',fg,e,fd')
                                     else 0,Noeud(d-d',fg,e,fd')
in
    let delta,ab = aux a in
      ab;;
```

Question 11 : Dans le meilleur des cas, l'élément à insérer est à la racine de l'arbre, alors la fonction récursive `aux` est appelée une seule fois. Dans le pire des cas, l'élément à insérer n'est pas présent dans l'arbre et il faut accéder à la feuille de profondeur maximale pour insérer notre élément. Dans ce cas le nombre d'appels à la fonction récursive `aux` est égal à $(h+1)$ où h est la hauteur de l'arbre.

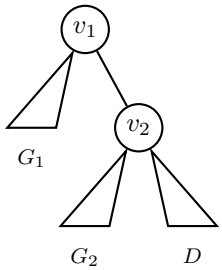
Question 12 : La rotation double de type D est la combinaison d'une rotation simple de type G du sous arbre gauche suivie d'un rotation simple de type D de l'arbre obtenu, comme l'illustre la figure suivante :



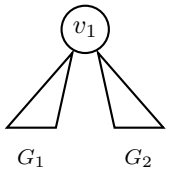
De la même façon, la rotation double de type G est la combinaison d'une rotation simple de type D du sous-arbre droit suivie d'une rotation de type G de l'arbre obtenue.

Question 13 : Pour la rotation simple de type D, on a, G, D_1 et D_2 qui sont des arbres binaires de recherche, les étiquettes de G sont plus petites que v_2, les étiquettes de D_1 sont plus grandes que v_2 mais plus petites que v_1 et les étiquettes de D_2 sont plus grandes que v_2. L'arbre 

est donc un arbre binaire de recherche dont toutes les étiquettes sont plus grandes que v_2.

Donc : 

est bien un arbre binaire de recherche.

De la même façon, pour la rotation de type G, les arbres G_1, G_2 et D sont des arbres binaires de recherche. Les étiquettes de G_1 sont plus petites que v_1 et les étiquettes de G_2 sont plus grandes que v_1 donc 

est un arbre binaire de recherche. Comme v_1, les étiquettes de G_1 et celle de G_2 sont plus petites que v_2 et les étiquettes de D sont plus grandes que v_2, l'arbre obtenu après rotation est bien un arbre binaire de recherche.

Pour les rotations doubles, comme ce sont des combinaisons de rotations simples et que les rotations simples conservent la structure d'arbre binaire de recherche, les rotations doubles vont aussi conserver la structure d'arbre binaire de recherche.

Question 14 : Pour la rotation simple de type D, on trouve :

$$\begin{aligned}
\delta'_1 &= h(D_1) - h(D_2) \\
&= -[1 + \max(h(G), h(D_1)) - h(D_1)] + 1 + \underbrace{\max(h(G), h(D_1)) - h(D_2)}_{\delta_1} \Rightarrow \boxed{\delta'_1 = \delta_1 - 1 - \max(\delta_2, 0)} \\
&= -[1 + \max(h(G) - h(D_1), 0)] + \delta_1 = \delta_1 - 1 - \max(\delta_2, 0) \\
\delta'_2 &= h(G) - [1 + \max(h(D_1), h(D_2))] \\
&= h(G) - 1 - [h(D_1) + \max(0, h(D_2) - h(D_1))] \Rightarrow \boxed{\delta'_2 = -1 - \delta_2 + \min(0, \delta'_1)} \\
&= -1 - (h(G) - h(D_1)) - \max(0, -\delta'_1) \\
&= -1 - \delta_2 + \min(0, \delta'_1)
\end{aligned}$$

De la même façon, pour la rotation simple de type G, on trouve :

$$\boxed{\delta'_1 = 1 + \delta_1 - \min(0, \delta_2)} \text{ et } \boxed{\delta'_2 = 1 + \delta_2 + \max(\delta'_1, 0)}$$

Pour la rotation double de type D :

$$\begin{aligned}
\delta'_2 &= h(G_1) - h(G_2) \\
&= \underbrace{h(G_1) - [1 + \max(h(G_2), h(D_1))]}_{=\delta_2} + [1 + \max(h(G_2), h(D_1)) - h(G_2)] \\
&\qquad \Rightarrow \boxed{\delta'_2 = 1 + \delta_2 + \min(0, \delta_3)} \\
&= \delta_2 + 1 + \max(0, h(D_1) - h(D_2)) \\
&= 1 + \delta_2 + \max(0, -\delta'_3) = 1 - \delta_2 + \min(0, \delta_3) \\
\delta'_1 &= h(D_1) - h(D_2) \\
&= h(D_1) - [1 + \max(h(G_1), 1 + \max(h(G_2), h(D_1))] - \big(\underbrace{h(D_2) - [1 + \max(h(G_1), 1 + \max(h(G_2), h(D_1))]}_{-\delta_1}\big) \\
&= h(D_1) - 1 - h(G_2) - \max(h(G_1) - h(G_2), 1 + \max(0, h(D_1) - h(G_2))) + \delta_1 \\
&= -\delta_3 - 1 - \max(\delta'_2, 1 + \max(0, \delta_3)) + \delta_1 \\
&\qquad \boxed{\delta'_1 = \delta_1 - \max(\delta'_2, 1 + \max(0, \delta_3)) - \delta_3 - 1} \\
\delta'_3 &= \max(h(G_1), h(G_2)) - \max(h(D_1), h(D_2)) \\
&= \max(\underbrace{h(G_1) - h(G_2)}_{=\delta'_2}, 0) + \underbrace{h(G_2) - h(D_1)}_{=\delta_3} - \max(0, \underbrace{h(D_2) - h(D_1)}_{=-\delta'_1}) \Rightarrow \boxed{\delta'_3 = \delta_3 + \max(\delta'_2, 0) + \min(\delta'_1, 0)}
\end{aligned}$$

De la même façon, pour la rotation double de type G, on trouve :

$$\boxed{\delta'_2 = \delta_2 - 1 - \max(\delta_3, 0)}$$

$$\boxed{\delta'_1 = \delta_1 + 1 + \max(1 + \max(\delta_3, 0), -\delta'_2) - \delta'_2}$$

$$\boxed{\delta'_3 = \max(\delta'_1, 0) + \delta_3 - \min(0, \delta'_2)}$$

Question 15 :

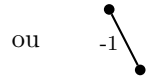
```
let rotationSD a = match a with
  Noeud(d1,Noeud(d2,G,v2,D1),v1,D2) -> let d'1 = d1-1-(max 0 d2) in
                                       let d'2 = (min 0 d'1)-1+d2 in
                      Noeud(d'2,G,v2,Noeud(d'1,D1,v1,D2))
  | a -> a;;
```

Question 16 :

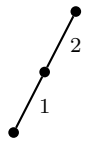
```
let rotationDD = function
  Noeud(d1,Noeud(d2,G1,v2,Noeud(d3,G2,v3,D1)),v1,D2) ->
          let d'2 = 1 + d2 + (min 0 d3) in
          let d'1 = d1 - (max d'2 (1+(max 0 d3))) - d3 -1 in
          let d'3 = d3+(max d'2 0)+(min d'1 0) in
                  Noeud(d'3,Noeud(d'2,G1,v2,G2),v3,Noeud(d'1,D1,v1,D2))
  | a -> a;;
```

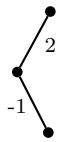
Notons que l'on ne peut pas se contenter d'écrire une fonction qui composerait les fonctions `rotationSD` et `rotationSG` car on n'obtient pas ainsi le nouveau déséquilibre δ_1' après la première rotation.

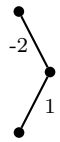
Question 17 : le nœud équilibré le plus profond atteint lors de l'insertion pouvant amener un déséquilibre, sera de la forme 

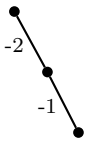
ou 

. Ils pourront amener aux arbres déséquilibrés suivants :

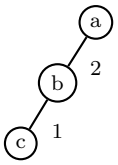


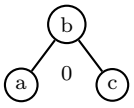
ou 

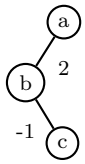
ou 

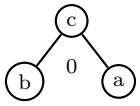
ou 

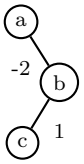
.

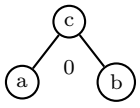
Question 18 : Un rotation à droite sur 

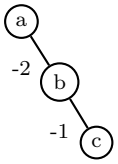
donne 

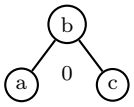
Une double rotation à droite sur 

donne 

Une double rotation à gauche sur 

donne 

Une rotation à gauche sur 

donne 

Question 19 :

```
let equilibrage a = match a with
 Noeud(2,Noeud(1,_,_,_),_,_)  -> rotationSD a
  | Noeud(2,Noeud(-1,_,_,_),_,_)  -> rotationDD a
  | Noeud(-2,_,_,Noeud(-1,_,_,_))  -> rotationSG a
  | Noeud(-2,_,_,Noeud(1,_,_,_))  -> rotationDG a
  | _ -> a;;
```

Question 21 :

```
let inserer_abr v A =
let rec aux = function
    Vide -> 1,Noeud(0,Vide,v,Vide)
  | Noeud(d,fg,e,fd) when v <e
    -> let (d',fg') = aux fg in
          if d+d' >= 2 then 0,equilibrage(Noeud(d+d',fg',e,fd))
                       else if d >= 0 then d',Noeud(d+d',fg',e,fd)
                                      else 0,Noeud(d+d',fg',e,fd)
   | Noeud(d,fg,e,fd) (* when v>e *)
     -> let (d',fd') = aux fd in
          if d-d' <= -2 then 0,equilibrage(Noeud(d-d',fg,e,fd'))
                        else if d <= 0 then d',Noeud(d-d',fg,e,fd')
                                       else 0,Noeud(d-d',fg,e,fd')
in
    let delta,A' = aux A in
    A';;
```

Question 22 : Pour un arbre équilibré de hauteur h, le cas le plus défavorable est quand l'un de ses sous-arbres est un arbre équilibré défavorable de hauteur $h-1$ et l'autre un arbre équilibré défavorable de hauteur $h-2$. On obtient la formule de récurrence : $\boxed{\mathcal{N}(h) = 1 + \mathcal{N}(h-1) + \mathcal{N}(h-2)}$.

Question 23 : On pose $\mathcal{F}(h) = \mathcal{N}(h) + 1$, on a alors :
$\mathcal{F}(h) = \mathcal{N}(h)+1 = 2+\mathcal{N}(h-1)+\mathcal{N}(h-2) = \mathcal{F}(h-1)+\mathcal{F}(h-2)$. On résout l'équation caractéristique $X^2 - X - 1 = 0$ et on trouve $\dfrac{1+\sqrt{5}}{2}$ et $\dfrac{1-\sqrt{5}}{2}$ comme racines. Les solutions de notre relation de récurrence sont les suites de la forme $u_n = \alpha.\left(\dfrac{1+\sqrt{5}}{2}\right)^n + \beta.\left(\dfrac{1-\sqrt{5}}{2}\right)^n$.
Comme $\mathcal{F}(1) = \mathcal{N}(h) + 1 = 2$ et $\mathcal{F}(2) = \mathcal{N}(h) + 1 = 2 + 1 = 3$ nous donne le système d'équation $\begin{cases} \alpha.\left(\frac{1+\sqrt{5}}{2}\right) + \beta.\left(\frac{1-\sqrt{5}}{2}\right) = 2 \\ \alpha.\left(\frac{1+\sqrt{5}}{2}\right)^2 + \beta.\left(\frac{1-\sqrt{5}}{2}\right)^2 = 3 \end{cases}$.
On trouve $\left\{\alpha = \dfrac{5+3\sqrt{5}}{10}, \beta = \dfrac{5-3\sqrt{5}}{10}\right\}$ et donc $\boxed{\mathcal{F}(h) = \left(\frac{5-3\sqrt{5}}{10}\right)\left(\frac{1-\sqrt{5}}{2}\right)^h + \left(\frac{5+3\sqrt{5}}{10}\right)\left(\frac{1+\sqrt{5}}{2}\right)^h}$.

Question 24 : Pour un arbre équilibré à n nœuds de profondeur h, on a, d'après les questions précédentes, $\boxed{\left(\frac{5-3\sqrt{5}}{10}\right)\left(\frac{1-\sqrt{5}}{2}\right)^h + \left(\frac{5+3\sqrt{5}}{10}\right)\left(\frac{1+\sqrt{5}}{2}\right)^h + 1 \leqslant n \leqslant 2^h - 1}$.

Le minorant étant équivalent à $\left(\frac{5+3\sqrt{5}}{10}\right)\left(\frac{1+\sqrt{5}}{2}\right)^h$, en passant au logarithme, on en déduit que $h = O(\ln n)$ et donc comme les opérations de recherche et d'insertion sont en $O(h)$, elles sont en $O(\ln n)$.