Travaux pratiques 2

Arbres A.V.L.

Option informatique MP1, MP2 & MP3

On se propose ici d'implémenter la structure d'arbre binaire de recherche en assurant une hauteur logarithmique. La structure choisie est celle définie par Georgy Adelson-Velsky and Evgenii Landis (d'où le nom AVL construit à partir des initiales) en 1962.
On ajoute à un nœud l'indication de sa hauteur et on équilibre l'arbre à chaque opération.

On rappelle qu'un arbre est **équilibré** si, pour tout nœud, la différence entre les hauteurs du fils droit et du fils gauche est au plus 1 en valeur absolue.
La suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_{n+2} = F_{n+1} + F_n$ pour $n \geq 0$.

Exercice 1 - Résultat théorique du cours de première année
Montrer que si a est un arbre équilibré de hauteur h alors
il contient au moins $F_{h+3} - 1$ nœuds.
En déduire que, pour un arbre équilibré de hauteur h et de taille n,
il existe un réel A tel que $h \leq A \log_2(n)$.

Ainsi la condition d'équilibre, garantit une hauteur logarithmique en la taille.

I Le type AVL

On considère un type d'arbre binaires que l'on a augmenté de l'indication de la hauteur (quatrième argument du constructeur `Noeud`).
Le type utilisé sera

```
type arbreAVL = Vide|Noeud of  arbreAVL * int * arbreAVL * int;;
```

Dans cette partie, on n'imposera pas de condition d'équilibre des hauteurs.
On imposera par contre que les valeurs (deuxième argument du constructeur `Noeud`) vérifient les conditions des arbres binaires de recherche.
Dans les questions suivantes on adaptera donc les fonctions du chapitre en ajoutant la gestion des hauteurs et sans utiliser la clé, on comparera directement les valeurs des clés.

Exercice 2 -
Écrire une fonction qui teste l'appartenance d'une valeur à un arbre.

```
chercher : int -> arbreAVL -> bool
```

Exercice 3 -
Écrire une fonction qui crée une feuille à partir d'une valeur du nœud.

```
feuille : int -> arbreAVL
```

Exercice 4 -
Écrire une fonction `ht` qui renvoie la hauteur d'un arbre.
La fonction doit renvoyer -1 dans le cas d'un arbre vide.

```
ht : arbreAVL -> int
```

Exercice 5 -
Écrire une fonction `cons g r d` avec r entier et g et d deux arbres AVL qui renvoie un arbre AVL construit avec ces paramètres (fils gauche, racine et fils droit respectivement).

```
cons : arbreAVL -> int -> arbreAVL -> arbreAVL
```

Exercice 6 -
En utilisant la fonction de construction ci-dessus, écrire une fonction d'insertion (aux feuilles) d'un entier dans un arbre AVL (de recherche).
Si la valeur existait déjà dans l'arbre on renvoie un arbre identique.

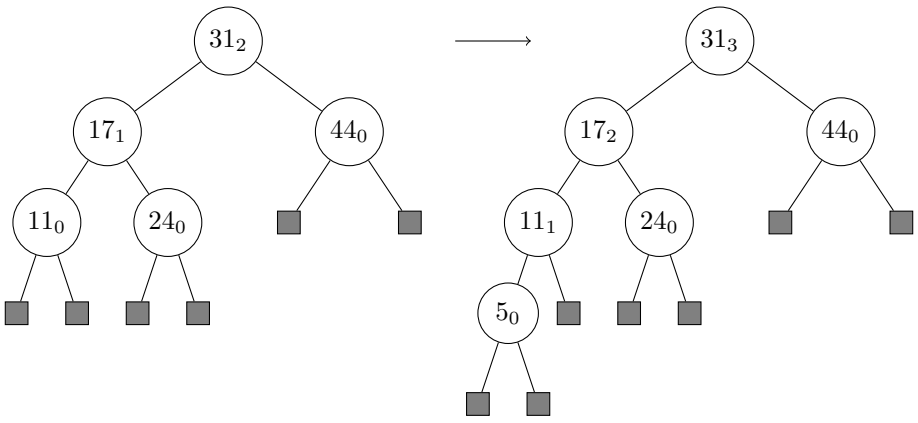
```
insertion : int -> arbreAVL  -> arbreAVL
```

Exercice 7 -
En adaptant les fonctions du cours écrire une fonction de suppression d'une valeur dans un arbre AVL.
Si la valeur n'existe pas dans l'arbre on renvoie un arbre identique.

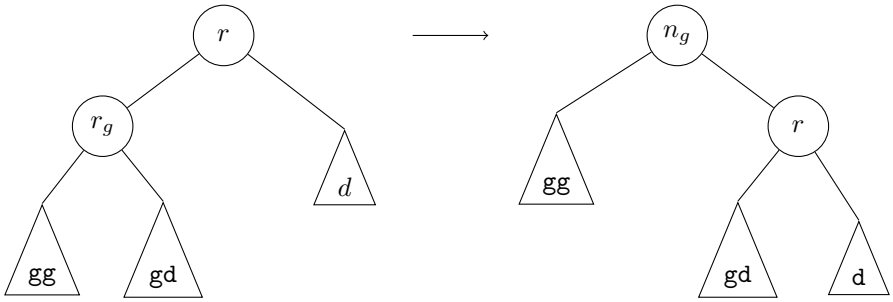
```
supression : int -> arbreAVL  -> arbreAVL
```

II Rotations

Pour l'instant l'introduction de la hauteur ne permet que de constater les déséquilibres.
Dans l'exemple ci-dessous, ou ajoute la valeur 5. La hauteur est en indice.



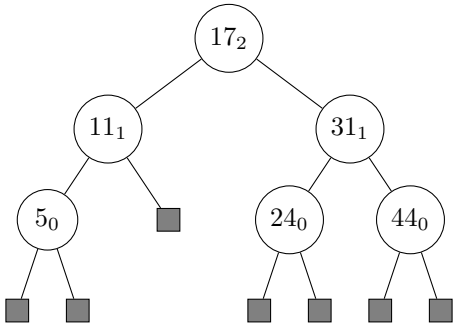
Pour en rétablir l'équilibre on va faire tourner la liaison entre la racine et le fils gauche en faisant glisser le fils droit du fils gauche le long de cette liaison pour qu'il devienne le fils gauche de l'ancienne racine devenue fils droit. Cette opération est nommée **rotation droite**.
On définit de même une rotation gauche.



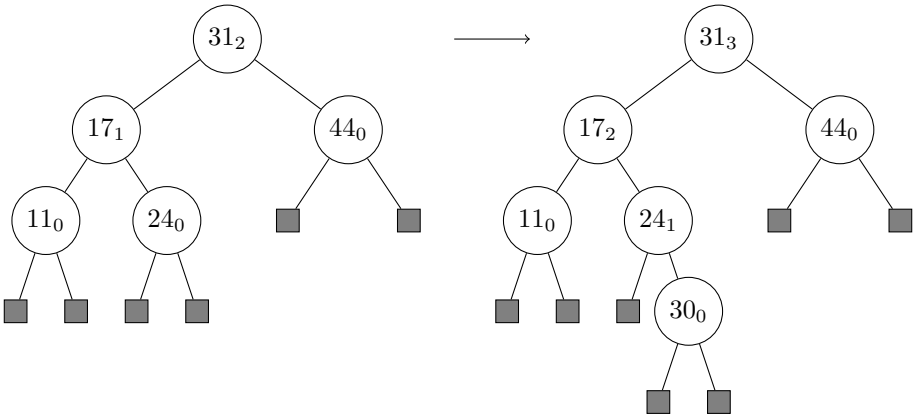
On suppose que l'arbre `Noeud(Noeud(gg, rg, gd, h+1), r, d, h+2)` est un arbre binaire de recherche mais n'est pas équilibré, uniquement à la racine avec `d` de hauteur $h-1$.
Un des deux fils du fils gauche est de hauteur h, l'autre est de hauteur h ou $h-1$.

Exercice 8 -
Si `gg` est de hauteur h et `gd` de hauteur $h' \in \{h-1, h\}$ alors le nœud obtenu par la rotation droite est un arbre binaire de recherche équilibré.

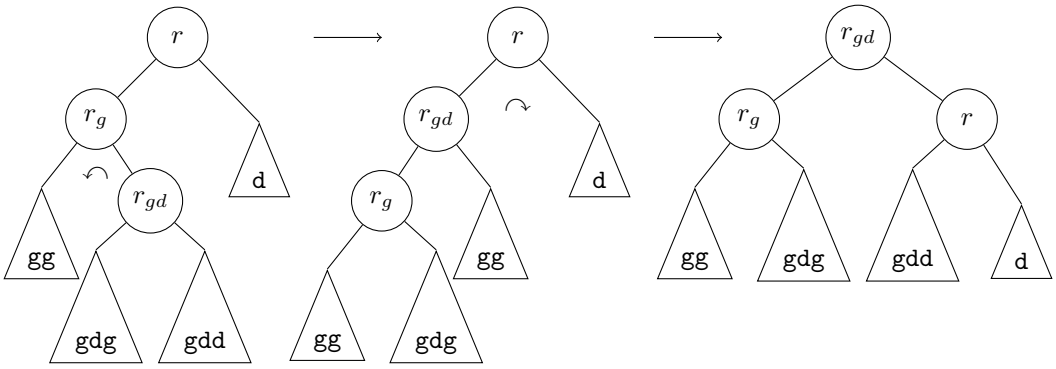
Cette situation est celle de l'exemple, la rotation le transforme en



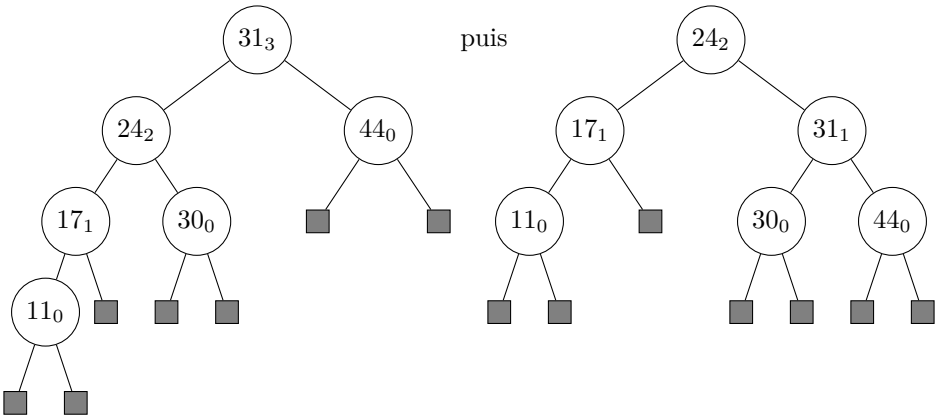
Si le déséquilibre vient de l'autre petit-fils cette solution ne convient pas. Par exemple si on ajoute la valeur 30, on obtient



Une rotation simple ne ferait que déplacer le déséquilibre de la gauche vers la droite.
La solution est d'effectuer une rotation gauche sur le fils gauche avant de réaliser la rotation droite.
La conclusien de l'exercice 8 reste valide si `d` est de hauteur h donc la composée des deux rotations maintient la structure d'arbre binaire de recherche et, comme la rotation gauche déplace la hauteur plus grande vers la gauche, on obtient un arbre équilibré.



On aboutit, avec l'exemple, à



Exercice 9 -
Modifier la fonction `cons` de la question 5 en une fonction `consAVL g n0 d` pour qu'elle équilibre les arbres qu'elle construit en utilisant les méthodes ci-dessus. Cette fonction pourra utiliser `cons`.

```
consAVL : arbreAVL -> int -> arbreAVL -> arbreAVL
```

Exercice 10 -
Qu'est-ce qui rend difficile l'insertion à la racine ?
Proposer une fonction.

Exercice 11 -
Faire des dessins

III Solutions

Solution de l'exercice 1 - Résultat théorique du cours de première année
On a $F_2 = 1$, $F_3 = 2$ et $F_4 = 3$.
On note $\mathcal{P}(h)$ la propriété :
tout arbre équilibré de hauteur h contient au moins $F_{h+3} - 1$ nœuds.
Un arbre de hauteur 0 est un nœud de fils vides donc contient 1 nœud et $F_3 - 1 = 1$.
Un arbre de hauteur 1 est un nœud dont au moins un fils est non vide : il contient au moins 2 nœuds et $F_4 - 1 = 2$. Ainsi $\mathcal{P}(0)$ et $\mathcal{P}(1)$ sont vraies.
On suppose que $\mathcal{P}(k)$ est vraie pour tout $k \leq h$ avec $h \geq 1$.
a est un arbre équilibré de hauteur $h+1$.
Ses deux fils sont de hauteur h au plus et au moins l'un des deux est de hauteur h.
Comme l'arbre est équilibré l'autre fils est de hauteur h ou $h-1$.
Les deux fils sont équilibrés donc l'un contient au moins $F_{h+3} - 1$ nœuds et l'autre contient au moins $F_{h-1+3} - 1$ nœuds.
On en déduit que la taille de a est minorée par $1 + F_{h+3} - 1 + F_{h+2} - 1 = F_{h+4} - 1$.
Ainsi $\mathcal{P}(h+1)$ est vrai donc la propriété est vraie pour tout arbre.

Les racines de $X^2 - X - 1$ sont $\alpha = \frac{1+\sqrt{5}}{2}$ et $\beta = \frac{1-\sqrt{5}}{2}$ et $F_n = \frac{1}{\sqrt{5}}\left(\alpha^{n+1} - \beta^{n+1}\right)$ donc, pour un arbre équilibré on a $n \geq \frac{1}{\sqrt{5}}\left(\alpha^{h+3} - \beta^{h+3}\right) - 1$. On en déduit

$$\frac{n}{\alpha^h} \geq \frac{\alpha^3}{\sqrt{5}} - \left(\frac{\beta}{\alpha}\right)^h \frac{\beta^3}{\sqrt{5}} - \frac{1}{\alpha^h} \geq \frac{\alpha^3}{\sqrt{5}} - \left(\frac{\beta}{\alpha}\right)^h \frac{\beta^3}{\sqrt{5}} - 1 = F_2 - 1 = 1$$

d'où $\alpha^h \leq n$ puis $h \leq \dfrac{1}{\log_2(\alpha)} \log_2(n)$.

Solution de l'exercice 2 -

```
let rec chercher n arbre =
   match arbre with
   |Vide -> false
   |Noeud(g, r, d, h) when (n = r) -> true
   |Noeud(g, r, d, h) when (n < r) -> chercher n g
   |Noeud(g, r, d, h) -> chercher n d;;
```

Solution de l'exercice 3 -

```
let feuille r =  Noeud(Vide, r, Vide, 0);;
```

Solution de l'exercice 4 -

```
let ht arbre =
   match arbre with
   |Vide -> -1
   |Noeud(g, r, d, h) -> h;;
```

Solution de l'exercice 5 -

```
let cons g r d =
   let hg = ht g in
   let hd = ht d in
   let h = (max hg hd) + 1 in
   Noeud(g, r, d, h);;
```

Solution de l'exercice 6 -

```
let rec insertion n arbre =
   match arbre with
   |Vide -> feuille n
   |Noeud(g, r, d, h) when n = r -> arbre
   |Noeud(g, r, d, h) when n < r
                      -> cons (insertion n g) r  d
   |Noeud(g, r, d, h) -> cons g r (insertion n d);;
```

Solution de l'exercice 7 -

```
let rec maxArbre arbre =
   match arbre with
   |Vide -> failwith "Arbre vide"
   |Noeud(_, r, Vide, _) -> r
   |Noeud(_, _, d, _) -> maxArbre d;;
```

```
let rec suppressionMax arbre =
   match arbre with
   |Vide -> failwith "Arbre vide"
   |Noeud(g, r, Vide, h) -> g
   |Noeud(g, r, d, h) -> cons g r (suppressionMax d);;
```

```
let suppressionRacine arbre =
   match arbre with
   |Vide -> Vide
   |Noeud(Vide, r, d, h) -> d
   |Noeud(g, r, d, h) ->  let p = maxArbre g in
                          cons (suppressionMax g) p d;;
```

```
let rec suppression n arbre =
   match arbre with
   |Vide -> Vide
   |Noeud(g, r, d, h) when n = r
                      -> suppressionRacine arbre
   |Noeud(g, r, d, h) when n < r
                      -> cons (suppression n g) r d
   |Noeud(g, r, d, h) -> cons g r (suppression n d);;
```

Solution de l'exercice 8 -
Le parcours infixe reste le même : parcours infixe de `gg`, r_g, parcours infixe de `gd`, r puis parcours infixe de `d`. Il reste donc croissant et l'arbre est bien un arbre binaire de recherche.
`d' = Noeud(gd, r, d, h'+1)` est bien équilibré car `gd` est de hauteur $h-1$ ou h et `d` est de hauteur $h-1$. De plus `gg` est de hauteur h et `d'` est de hauteur h ou $h+1$ donc l'arbre obtenu est bien équilibré.

Solution de l'exercice 9 -

```
let consAVL g r d =
  let hg = ht g in
  let hd = ht d in
  if hg - hd > 1
  then begin
     match g with
     |Noeud(gg, rg, gd, _) when ht gg >= ht gd
         -> cons gg rg (cons gd r d)
     |Noeud(gg, rg, Noeud(gdg, rgd, gdd, _), _)
         -> cons (cons gg rg gdg) rgd (cons gdd r d)
     | _ -> failwith "Ceci ne devrait pas arriver" end
  else if hg - hd < -1
       then begin
          match d with
          |Noeud(dg, rd, dd, _) when ht dd > ht dg
              -> cons (cons g r dg) rd dd
          |Noeud(Noeud(dgg, rdg, dgd, _), rd, dd, _)
              -> cons (cons g r dgg) rdg (cons dgd rd dd)
          | _ -> failwith "Ceci ne devrait pas arriver" end
       else cons g r d;;
```

Solution de l'exercice 10 -
Lors du découpage d'un arbre on peut obtenir deux arbres dont la différence de hauteur est supérieure à 2. Pour pouvoir équilibrer on peut remplacer la fonction `consAVL` en la rendant récursive.

```
let rec consAVL g n0 d =
  let hg = ht g in
  let hd = ht d in
  if (hg - hd) > 1
  then begin match g with
    |Noeud(gg, rg, gd, _) when ht gg >= ht gd
      -> consAVL gg rg (consAVLgd n0 d)
    |Noeud(gg, rg, Noeud(gdg, rgd, gdd, _), _)
      -> consAVL (consAVLgg rg gdg) rgd (consAVLgdd n0 d)
    | _ -> failwith "Ceci ne devrait pas arriver" end
  else if (hg - hd) < -1
  then begin match d with
    |Noeud(dg, rd, dd, _) when ht dd > ht dg
      -> consAVL (consAVL g n0 dg) rd dd
    |Noeud(Noeud(dgg, rdg, dgd, _), rd, dd, _)
      -> consAVL (consAVL g n0 dgg) rdg (consAVL dgd rd dd)
    | _ -> failwith "Ceci ne devrait pas arriver" end
  else cons g n0 d;;
```

Solution de l'exercice 11 -
On a bien sur modifié l'insertion :

```
let rec insertion n arbre =
   match arbre with
   |Vide -> feuille n
   |Noeud(g, r, d, h) when n = r -> arbre
   |Noeud(g, r, d, h) when n < r
                      -> consAVL (insertion n g) r  d
   |Noeud(g, r, d, h) -> consAVL g r (insertion n d);;
```

Chargement du graphisme

```
#load "graphics.cma";;
open Graphics;;
```

pour OCaml < 4.09

```
#use "topfind";;
#require "graphics";;
open Graphics;;
```

Les constantes

```
let x0 = 20;;
let y0 = 500;;
let pas_h = 30;;
let pas_v = 60;;
let cote = 16
let rayon = 15;;
```

Le dessin des nœuds

```
let dessinNoeud arbre x y =
(* Dessin de la racine d'un arbre à la position (x,y)*)
   match arbre with
   |Vide -> set_color black;
            fill_rect (x-cote/2) (y-cote/2) cote cote
   |Noeud(_, n, _ , _) ->  set_color black;
                           fill_circle x y rayon;
                           set_color white;
                           if n < 10 then moveto (x - 2) (y - 5)
                                     else moveto (x - 6) (y - 5);
                           draw_string (string_of_int n);;
```

Le dessin d'un arbre

```
let dessin arbre =
   let rec aux a x y =
   match a with
   |Vide -> (x, x)
   |Noeud(g, n, d, h) -> let yf = y - pas_v in (* ordonnée des
      fils*)
                         let (xg, xrg) = aux g x yf in
                         let (xd, xrd) = aux d (xg+2*pas_h) yf in
                         set_color black;
                         moveto xrg yf;
                         lineto (xg+pas_h) y;
                         lineto xrd yf;
                         dessinNoeud g xrg yf;
                         dessinNoeud d xrd yf;
                         (xd, xg+pas_h) in
   let max, xr = aux arbre x0 y0 in
   dessinNoeud arbre xr y0;
   let _ = wait_next_event [Button_down] in clear_graph ();;
```

Une visualisation pas-à-pas

```
let voirAVL liste =
   let rec aux arbre reste =
      match reste with
      |[] -> dessin arbre
      |t::q -> dessin arbre;
               aux (insertion t arbre) q in
   open_graph " 800x600";
   aux Vide liste;
   close_graph ();;
```

```
voirAVL [1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 11; 12];;
```