TP supplémentaire 1 - Arbres AVL - MP* - Correction

Question 1.a – Une structure de données est dite persistante si elle est statique et non mutable. "Statique" signifie qu'une fois que la structure de données a été créée, la mémoire utilisée pour stocker cette structure est de taille fixe. "Non mutable" signifie qu'après l'insertion d'une donnée dans la structure de données, cette donnée ne peut plus être modifiée.

Question 1.b – Oui, le type `dict` est une structure de données persistante.

Question 2 –

```
let lt s1 s2 =
  let i = ref 0 in
  while !i < String.length s1 && !i < String.length s2 && s1.[!i] = s2.[!i] do
    incr i
  done;
  !i < String.length s2 && (!i = String.length s1 || s1.[!i] < s2.[!i]);;
```

Question 3.a –

```
let create () =
  Vide;;
```

Question 3.b –

```
let is_empty abr = match abr with
  | Vide -> true
  | _ -> false;;
```

Question 4 –

```
let rec find abr c = match abr with
  | Vide -> failwith "La cle n'apparait pas dans le dictionnaire"
  | N(e, g, _) when e.cle > c -> find g c
  | N(e, _, d) when e.cle < c -> find d c
  | N(e, _, _) -> e.valeur;;
```

Question 5 –

```
let rec add abr c v = match abr with
  | Vide -> N({cle = c; valeur = v}, Vide, Vide)
  | N(e, g, d) when e.cle > c -> N(e, add g c v, d)
  | N(e, g, d) when e.cle < c -> N(e, g, add d c v)
  | N(e, _, _) -> failwith "La cle apparait deja dans le dictionnaire";;
```

Question 6 – Les fonctions `create` et `is_empty` s'exécutent en temps constant.
Pour les fonctions `find` et `add`, il y a au plus un appel récursif sur le sous-arbre gauche ou sur le sous-arbre droit. Ces fonctions s'exécutent donc en temps $\mathcal{O}(h)$ où h est la hauteur de l'arbre donné en entrée.

Question 7 – On le montre par récurrence forte h :

- Pour $h = -1$, l'arbre est vide, il possède donc 0 noeud et on a bien $0 \geqslant f_0 - 1 = 0$
- Soit $h \in \mathbb{N}$. On suppose la propriété vraie pour tout $h' < h$ et on la montre au rang h. Un arbre AVL de hauteur h est non vide, la racine possède donc un sous-arbre gauche G et un sous-arbre droit D. Soit n le nombre de noeuds de l'arbre, soient n_G et h_G le nombre de noeuds et la hauteur du sous-arbre gauche, et soient n_D et h_D le nombre de noeuds et la hauteur du sous-arbre droit. On a $h = 1 + \max(h_G, h_D)$, donc $h_G < h$ et $h_D < h$. Par l'hypothèse de récurrence :

$$n_G \geqslant f_{h_G+1} - 1 \qquad\qquad n_D \geqslant f_{h_D+1} - 1$$

  Puisque l'arbre est un AVL, on a trois cas :

- $h_G = h-1$ et $h_D = h-2$. Alors :

$$n = 1 + n_G + n_D \geqslant f_h + f_{h-1} - 1 = f_{h+1} - 1$$

- $h_G = h-2$ et $h_D = h-1$. Ce cas se traite de la même façon que le cas précédent.
- $h_G = h-1$ et $h_D = h-1$. On utilise alors le fait que la suite de Fibonacci est croissante :

$$n = 1 + n_G + n_D \geqslant f_h + f_h - 1 \geqslant f_h + f_{h-1} - 1 = f_{h+1} - 1$$

Question 8.a – On le montre par récurrence double sur n :

- Pour $n = 0$, on a bien $\varphi^{n-1} = 1/\varphi \leqslant 1 = f_{n-1}$
- Pour $n = 1$, on a bien $\varphi^{n-1} = 1 = f_{n-1}$
- Soit $n \geqslant 2$. On suppose la propriété vraie aux rangs $(n-1)$ et $(n-2)$, et on la montre au rang n. En utilisant l'hypothèse de récurrence et le fait que φ est racine du polynôme $X^2 - X - 1$, on obtient :

$$f_n = f_{n-1} + f_{n-2} \geqslant \varphi^{n-2} + \varphi^{n-3} = \varphi^{n-3}(\varphi + 1) = \varphi^{n-3}\varphi^2 = \varphi^{n-1}.$$

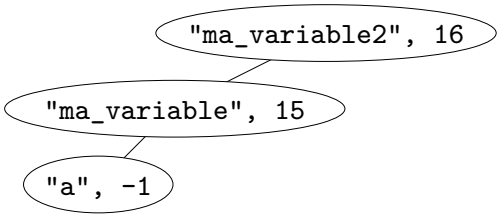
Question 8.b – D'après les questions précédentes :

$$n \geqslant f_{h+1} - 1 \geqslant \varphi^h - 1.$$

D'où :

$$h \leqslant \log_\varphi(n+1).$$

Question 8.c – La réciproque est fausse. Par exemple, l'arbre suivant n'est pas un AVL :



Pourtant, $h = 2 = \log_2(4) \leqslant \log_\varphi(4) = \log_\varphi(n+1)$.

Question 9 – D'après la question 8.b, $h = \mathcal{O}(\log n)$. D'après le cours, pour tout arbre binaire, on a $h \geqslant \log_2(n+1) - 1$. Donc $h = \Theta(\log n)$

Question 10 – Soient $Inf(A_1)$, $Inf(A_2)$ et $Inf(A_3)$ les parcours infixes des arbres A_1, A_2 et A_3. Dans l'arbre α, le parcours infixe du sous-arbre de racine x est :

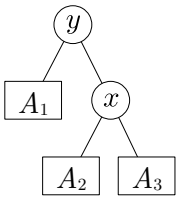
$$Inf(A_1) \texttt{ @ [}y\texttt{] @ } Inf(A_2) \texttt{ @ [}x\texttt{] @ } Inf(A_3)$$

En fait, c'est aussi le parcours infixe du sous-arbre de racine x dans l'arbre β. On en déduit que le parcours infixe de l'arbre α est égal au parcours infixe de l'arbre β.

Puisqu'un arbre binaire est de recherche si et seulement si son parcours infixe est trié, on conclut que si l'arbre α est un arbre binaire de recherche alors l'arbre β l'est aussi. Réciproquement, si l'arbre β est un arbre binaire de recherche alors l'arbre α l'est aussi.

Question 11.a – Dans les deux cas, le sous-arbre de racine y est de hauteur h donc le facteur d'équilibrage de la racine est $2 \notin \{-1, 0, 1\}$. Ainsi, l'arbre n'est pas AVL.

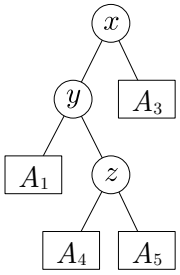
Question 11.b – Si on applique une rotation-droite à la racine, on obtient l'arbre :



D'après la question 10, l'arbre obtenu est un arbre binaire de recherche. L'énoncé fait l'hypothèse que A_1, A_2 et A_3 sont des AVL, il reste donc à montrer que les facteurs d'équilibrage de x et y sont -1, 0 ou 1.

Le facteur d'équilibrage de x est 0 et la hauteur du sous-arbre de racine x est $(h-1)$. Donc le facteur d'équilibrage de y est 0 et l'arbre est bien un AVL de hauteur h.

Question 11.c – Comme A_2 est de hauteur $h-1 \geqslant 0$, il est non vide. L'arbre est donc de la forme :



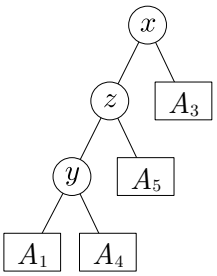
avec A_4 et A_5 des AVL. De plus, on se trouve dans l'un de ces cas :

(cas a) A_4 est de hauteur $(h-2)$ et A_5 est de hauteur $(h-2)$.

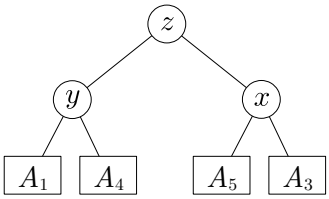
(cas b) A_4 est de hauteur $(h-2)$ et A_5 est de hauteur $(h-3)$.

(cas c) A_4 est de hauteur $(h-3)$ et A_5 est de hauteur $(h-2)$.

Lorsqu'on applique, une rotation gauche sur le noeud y, on obtient :



Lorsqu'on applique une rotation droite sur le noeud x, on obtient :



On vérifie facilement dans les cas a, b et c que les facteurs d'équilibrage sont -1, 0 ou 1, et que l'arbre obtenu est de hauteur h.

Question 12 – Pour $h=-1$, l'arbre initial est vide. En insérant un noeud, on obtient un arbre réduit à une feuille et aucune rotation n'est effectuée. L'arbre obtenu est bien un AVL de hauteur $0 = h+1$ et aucune rotation n'a été effectuée.

Question 13 – Si $h_{G'} = h_G$, alors le facteur d'équilibrage de la racine est le même qu'avant l'insertion. Aucune rotation n'est donc effectuée et l'arbre A' est un AVL. De plus, $h_G = h_{G'}$ et $h_D = h_{D'}$ donc $h' = h$.

Question 14.a – Si $h_{G'} = h_G + 1$ et $h_D = h_G$, alors le facteur d'équilibrage de la racine est $h_G + 1 - h_G = 1$. Il n'y a donc pas de rotation effectuée au niveau de la racine. De plus :

$$h' = 1 + \max(h_{G'}, h_D) = h_G + 2 \qquad h = 1 + \max(h_G, h_D) = h_G + 1$$

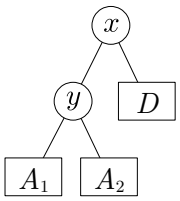
Donc $h' = h + 1$.

Question 14.b – Si $h_{G'} = h_G+1$ et $h_D = h_G+1$, alors le facteur d'équilibrage de la racine est $h_G+1-(h_G+1) = 0$. Il n'y a donc pas de rotation effectuée au niveau de la racine. De plus :

$$h' = 1 + \max(h_{G'}, h_D) = h_G + 2 \qquad h = 1 + \max(h_G, h_D) = h_G + 2$$

Donc $h' = h$.

Question 14.c – Si $h_{G'} = h_G + 1 \geqslant 0$ et $h_D = h_G - 1$, alors l'arbre A' est de la forme :



On a $h = 1 + \max(h_G, h_D) = h_G + 1$. Donc :

$$h_G = h - 1 \qquad h_{G'} = h \qquad h_D = h - 2$$

Puisque G' est de hauteur h, on fait une disjonction de cas sur les hauteurs de A_1 et A_2 :

- Si A_1 est de hauteur $(h-1)$ et A_2 est de hauteur $(h-2)$, alors on se trouve dans le cas 1 de la partie 2.2. Ainsi, une rotation est effectuée et l'arbre obtenu est un AVL de hauteur h.
- Si A_1 est de hauteur $(h-2)$ et A_2 est de hauteur $(h-1)$, alors on se trouve dans le cas 2 de la partie 2.2. Ainsi, deux rotations sont effectuées et l'arbre obtenu est un AVL de hauteur h.
- Sinon, A_1 est de hauteur $(h-1)$ et A_2 est de hauteur $(h-1)$. En fait, ce cas est impossible. Si c'était le cas, d'après l'égalité $h_{G'} = h_G + 1$ et l'hypothèse de récurrence, G' serait l'arbre G auquel le noeud z a été ajouté, mais aucune rotation n'a été effectuée. Sans perte de généralité, on peut supposer que z est insérer dans le sous-arbre gauche de G. Alors :
  - Le sous-arbre gauche de G est égal à A_1 dans lequel un noeud a été supprimé.
  - Le sous-arbre droit de G est égal à A_2.

  Donc G est de hauteur h ce qui contredit $h_G = h - 1$.

Question 15 –

```
let parc_inf arb0 =
  let rec aux acc arb = match arb with
    | Vide -> acc
    | N(e, g, d) ->
       let acc2 = aux acc d in
       aux (e.cle::acc2) g
  in
  aux [] arb0;;
```

```
let rec est_triee li = match li with
  | a::b::q -> a <= b && est_triee (b::q)
  | _ -> true;;
```

```
let est_ABR arb = est_triee (parc_inf arb);;
```

```
(* La fonction aux renvoie un booléen qui indique si l'arbre est équilibré ainsi
   que la hauteur de l'arbre *)
let est_equilibre arb0 =
  let rec aux  arb = match arb with
    | Vide -> true, -1
    | N(e, g, d) ->
       let bg, hg = aux g and
           bd, hd = aux d in
       bg && bd && abs(hg - hd) < 2, 1 + max hg hd
  in
  fst (aux arb0);;
```

```
let est_AVL arb = est_ABR arb && est_equilibre arb;;
```

Question 16 –

⋆ On commence par écrire des fonctions intermédiaires.

```
let hauteur arb = match arb with
  | Vide -> -1
  | N(e, _, _) -> e.hauteur;;
```

La fonction `maj_hauteur` met à jour la le champs `hauteur` de la racine de l'arbre.

```
let maj_hauteur arb = match arb with
  | Vide -> failwith "Ne devrait pas arriver"
  | N(e, g, d) ->
     let e2 = {
         valeur = e.valeur;
         cle = e.cle;
         hauteur = 1 + max (hauteur g) (hauteur d)
       } in
     N(e2, g, d);;
```

⋆ On équilibre l'arbre dans le cas où le facteur d'équilibrage de la racine est 2.

Les variables `x, y, a1, a2, a3` correspondent aux notations de l'énoncé.

```
let equilibrer_g_cas1 x y a1 a2 a3 =
  let d = maj_hauteur (N(x, a2, a3)) in
  maj_hauteur (N(y, a1, d));;
```

```
let equilibrer_g_cas2 x y a1 a2 a3 = match a2 with
  | N(z, a4, a5) -> let g = maj_hauteur (N(y, a1, a4)) in
                    let d = maj_hauteur (N(x, a5, a3)) in
                    maj_hauteur (N(z, g, d))
  | _ -> failwith "Ne devrait pas arriver";;
```

```
let equilibrer_g x g d =
  match g with
  | N(y, a1, a2) when hauteur a1 = hauteur a2 + 1 && hauteur a2 = hauteur d
    -> equilibrer_g_cas1 x y a1 a2 d
  | N(y, a1, a2) when hauteur a1 = hauteur a2 - 1 && hauteur a2 = hauteur d + 1
    -> equilibrer_g_cas2 x y a1 a2 d
  | _ -> failwith "Ne devrait pas arriver";;
```

⋆ Même principe lorsque le facteur d'équilibrage de la racine est -2.

```
let equilibrer_d_cas1 x y a1 a2 a3 =
  let g = maj_hauteur (N(x, a1, a2)) in
  maj_hauteur (N(y, g, a3));;
```

```
let equilibrer_d_cas2 x y a1 a2 a3 = match a2 with
  | N(z, a4, a5) -> let g = maj_hauteur (N(x, a1, a4)) in
                    let d = maj_hauteur (N(y, a5, a3)) in
                    maj_hauteur (N(z, g, d))
  | _ -> failwith "Ne devrait pas arriver";;
```

```
let equilibrer_d x g d =
  match d with
  | N(y, a2, a3) when hauteur a3 = hauteur a2 + 1 && hauteur a2 = hauteur g
    -> equilibrer_d_cas1 x y g a2 a3
  | N(y, a2, a3) when hauteur a3 = hauteur a2 - 1 && hauteur a2 = hauteur g + 1
    -> equilibrer_d_cas2 x y g a2 a3
  | _ -> failwith "Ne devrait pas arriver";;
```

⋆ On peut maintenant écrire la fonction principale

```
let equilibrer arb = match arb with
  | Vide -> failwith "Ne devrait pas arriver"
  | N(x, g, d) -> let fact_eq = hauteur g - (hauteur d) in
                  if fact_eq = -1 || fact_eq = 0 || fact_eq = 1 then
                    maj_hauteur (N(x, g, d))
                  else if fact_eq = 2 then equilibrer_g x g d
                  else if fact_eq = -2 then equilibrer_d x g d
                  else failwith "Ne devrait pas arriver";;
```

```
let rec addAVL avl c v = match avl with
  | Vide -> N({cle = c; valeur = v; hauteur = 0}, Vide, Vide)
  | N(e, g, d) when e.cle > c -> equilibrer (N(e, addAVL g c v, d))
  | N(e, g, d) when e.cle < c -> equilibrer (N(e, g, addAVL d c v))
  | N(e, _, _) -> failwith "La cle apparait deja dans le dictionnaire";;
```