Corrigé T.D.4

Q.17 La propriété caractérisant le fait d'être un arbre d'entiers peut s'énoncer récursivement de la manière suivante : $\boxed{A(a) : (a = \emptyset) \text{ ou } [\mathcal{E}(a) \in \mathbb{N} \text{ et } (\forall\, b \in \mathcal{S}(a), A(b) \text{ et } b \neq \emptyset)]}$

Q.18 Voici une première réponse utilisant une fonction auxiliaire récursive calculant la taille d'une forêt (c'est-à-dire d'une liste d'arbres) :

```
let taille a =
   let rec aux f = match f with
      |Vide::g -> aux g
      |Noeud(_,s)::g -> 1 + aux s + aux g
      |_ -> 0
   in aux [a];;
```

Remarquons que le premier cas de filtrage est inutile si `a` est effectivement un arbre d'entiers. On peut également utiliser une récurrence croisée avec deux fonctions `taille` et `tailles` donnant respectivement la taille d'un arbre d'entiers et celle d'une liste d'arbres d'entiers

```
let rec taille a = match a with
  |Vide       -> 0
  |Noeud(_,b) -> 1 + tailles b
and tailles b = match b with
  |[]   -> 0
  |a::c -> taille a + tailles c;;
```

Une deuxième réponse possible repose sur l'idée suivante : la taille d'un arbre non vide et non réduit à une feuille est la somme de celle du fils aîné de sa racine et de celle de l'arbre amputé de son fils aîné ce qui donne :

```
let rec taille a = match a with
  |Vide          -> 0
  |Noeud(_,[])   -> 1
  |Noeud(e,f::q) -> taille f + taille Noeud(e,q);;
```

Q.19 Si `a=Vide`, sa hauteur n'est pas définie ; si `a=Noeud(x,[])` sa hauteur vaut 0. Sinon,

$$\boxed{\mathcal{H}(a) = 1 + \max_{b \in \mathcal{S}(a)} (\mathcal{H}(b))}$$

Q.20 Un arbre de taille n est de hauteur maximale si chacun de ces nœuds possède exactement un fils : il est alors de hauteur $n-1$.
Un arbre de taille n est de hauteur minimale si sa racine possède $n-1$ fils : il est alors de hauteur 1 si $n \geqslant 2$ et de hauteur 0 si $n = 1$.

Q.21 On définit tout d'abord une fonction auxiliaire donnant le maximum d'une liste d'entiers naturels si cette liste est non vide et -1 sinon.

```
let rec maxi l = match l with
  | []   -> -1
  | t::q -> max t (maxi q);;
```

On en déduit facilement la fonction `hauteur`, la cas d'un arbre réduit à une feuille étant traité correctement vu la définition de `Maxi`.Rappelons que l'instruction `List.map f l` transforme la liste `l=[a;b;...;z]` en `[f(a);f(b);...;f(z)]`.

```
let rec hauteur a = match a with
  |Vide       -> failwith "arbre vide"
  |Noeud(_,b) -> 1 + Maxi (List.map hauteur b);;
```

On peut également donner une variante utilisant la même idée que celle de la deuxième réponse à la question 18 :

```
let rec hauteur a = match a with
  |Vide -> failwith "arbre vide"
  |Noeud(_,[])->0
  |Noeud(e,f::q)->max (1+hauteur f) (hauteur (Noeud(x,q)));;
```

Q.22 Il est logique de considérer que l'arbre vide vérifie la propriété AT d'où, pour un arbre d'entiers a,

$$\boxed{AT(a) : (a = Vide) \quad \text{ou} \quad (\forall\, b \in \mathcal{S}(a),\ (\mathcal{E}(a) < \mathcal{E}(b) \quad \text{et} \quad AT(b))}$$

On notera l'importance d'avoir $AT(b)$ dans la dernière condition, afin de vérifier la condition sur les étiquettes à tous les niveaux et pas seulement au niveau des fils de la racine.

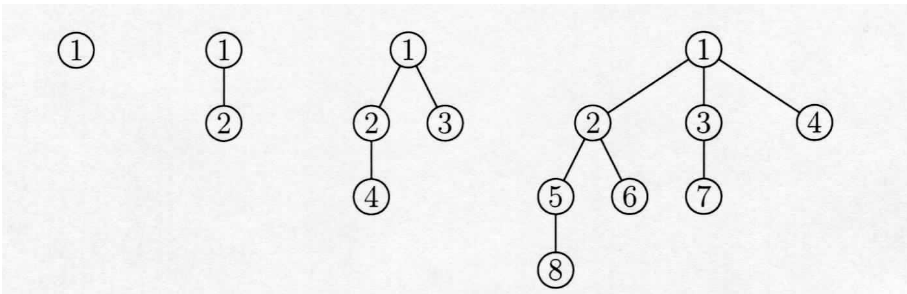
Q.23 On définit tout d'abord deux fonctions auxiliaires, la première retournant l'étiquette de la racine d'une arbre non vide et la seconde, indiquant si tous les éléments d'une liste vérifient une propriété :

```
let racine a = match a with
  | Vide      -> failwith "arbre vide"
  | Noeud(a,_)-> a;;
let rec pour_tout liste prop = match liste with
  | []   -> true
  | t::q -> (prop t) && (pour_tout q prop);;
```

On en déduit la fonction indiquant si un arbre est un arbre d'entiers en tas :

```
let rec validerAT arbre = match arbre with
  | Vide          -> true
  | Noeud(a,fils) -> pour_tout fils (fun b -> (racine b > a) && (validerAT b));;
```

Q.24 Exemples d'arbres binomiaux d'entiers en tas d'ordres 0, 1, 2 et 3 :



Q.25 Soit $k \in \mathbb{N}$ et `A=Noeud(x,F)` un arbre binomial d'entiers en tas d'ordre $k+1$. Alors F est de la forme $[F_k, F_{k-1}, \dots, F_0]$ où pour tout $i \in [\![0, k]\!]$, F_i est binomial en tas d'ordre i. Posons donc `C=`F_k et `B=Noeud(x,G)` où `G=`$[F_{k-1}, \dots, F_0]$.
Alors `B` et `C` sont clairement des arbres binomiaux d'entiers en tas d'ordre k et `A` est obtenu à partir de `B` et `C` selon la configuration décrite dans l'énoncé.
Remarque : on a clairement la réciproque de la propriété précédente, c'est-à-dire qu'à partir de deux arbres binomiaux en tas d'ordre k on peut fabriquer un arbre binomial en tas d'ordre $k+1$ (en prenant soin toutefois de choisir pour `B` celui dont l'étiquette de la racine est la plus petite) selon le schéma décrit par l'énoncé mais il n'était pas demandé de le démontrer.

Q.26 Montrons par récurrence sur $k \in \mathbb{N}$ qu'un arbre binomial d'ordre k est de taille 2^k.
⋄ La propriété est vraie pour $k = 0$ puisqu'un arbre binomial d'ordre 0 possède un unique nœud.
⋄ Supposons la propriété vraie à l'ordre k et soit A un arbre binomial d'ordre $k+1$. Alors, on peut lui associer deux arbres binomiaux B et C d'ordre k comme dans la question 9 donc $\mathcal{T}(A) = \mathcal{T}(B) + \mathcal{T}(C) = 2^k + 2^k = 2^{k+1}$ ce qui établit la propriété à l'ordre $k+1$ et achève la

récurrence.

Q.27 De même on montre par récurrence qu'un arbre binomial d'ordre k est de hauteur égale à k.

◇ La propriété est vraie pour les premières valeurs de k (*cf* question 8)

◇ Supposons la propriété vraie à l'ordre k et soit A arbre binomial d'ordre $k+1$. Considérons B et C comme dans la question 9 : alors $\mathcal{H}(A) = \max(\mathcal{H}(B), 1+\mathcal{H}(C)) = \max(k, k+1) = k+1$ ce qui établit la propriété à l'ordre $k+1$ et achève la récurrence.

Q.28 Notons pour $k \in \mathbb{N}$, $P(k)$ la propriété : Pour tout arbre binomial A d'ordre k et tout $p \in [\![0, k]\!]$, A possède exactement $\binom{k}{p}$ nœuds de profondeur p.

◇ La propriété est vraie pour $k \in [\![0, 3]\!]$ comme on le vérifie facilement (*cf* question 8)

◇ Supposons la propriété vraie à l'ordre k et soit A arbre binomial d'ordre $k+1$. Considérons B et C comme dans la question 9. Soit $p \in [\![0, k+1]\!]$.

- Si $p = 0$ il y a un unique nœud de A de profondeur 0 qui est la racine de A et $1 = \binom{k+1}{0}$.
- Si $p \in [\![1, k]\!]$, les nœuds de profondeur p de A sont obtenus à partir des nœuds de profondeur p de B ou de ceux de profondeur $p-1$ de C : il y en a donc $\binom{k}{p} + \binom{k}{p-1} = \binom{k+1}{p}$ d'après la formule du triangle de Pascal.
- Enfin si $p = k+1$, les nœuds de profondeur p de A sont ceux de profondeur $p-1 = k$ dans C. Il y en a $\binom{k}{k} = 1 = \binom{k+1}{k+1}$ ce qui achève la récurrence.

Q.29 On a pour $k = 0$,

$$ABT_0(a) : \mathcal{S}(a) = [\,]$$

et si $k \in \mathbb{N}$

$$ABT_{k+1}(a) : (\mathcal{E}(a) < \mathcal{E}(c)) \text{ et } ABT_k(c) \text{ et } ABT_k(b)$$

où $c = List.hd(\mathcal{S}(a))$ et $b = Noeud(\mathcal{E}(a), List.tl(\mathcal{S}(a)))$

Q.30 On peut par exemple traduire la propriété précédente de la manière suivante :

```
let rec validerABT k arbre = match arbre with
  | Vide          -> false
  | Noeud(_,[])  -> (k=0)
  | Noeud(x,c::f)-> (x < racine c) && (validerABT (k-1) c)
                                    && (validerABT (k-1) (Noeud(x,f)));;
```

Q.31 D'après la question 10, le tas $t = a_0, \ldots, a_l$, de signature $s_0, \ldots, s_l$ a pour taille

$$\mathcal{T}(t) = \sum_{i=0}^{l} s_i 2^i$$

Q.32 D'après la question précédente, la signature d'un tas binomial est la suite des chiffres de l'écriture en base 2 de sa taille : elle est donc entièrement déterminée par la taille de ce tas.

Q.33 Si on note $\wedge$ le « et » logique, on a pour $t = a_0, \ldots, a_l$,

$$TB(t) : ABT_l(a_l) \wedge \Big(\big(\bigwedge_{i=0}^{l-1} (a_i = \emptyset \text{ ou } ABT_i(a_i) \big) \Big)$$

Q.34
```
let validerTB tas =
  let rec valide k f = match f with
    | [] -> true
    | [a]-> validerABT k a
    | t::q -> (t=Vide || validerABT k t) && (valide (k+1) q)
  in valide 0 tas;;
```

La fonction auxiliaire `valide k f` teste si f est la « queue » $a_k, \ldots, a_l$ d'un tas binomial $a_0, \ldots, a_k$.

Q.35 D'après la question 16, la signature d'un tas binomial t' obtenu en ajoutant une valeur à un tas binomial t, est la décomposition binaire de l'entier décrit par la signature de t incrémenté de 1.

Considérons tout d'abord une fonction qui fabrique selon le procédé décrit en question 25 un arbre binomial d'ordre $k+1$ à partir de deux arbres binomiaux d'ordre k.

```
let combine b c = match b , c with
  | Noeud(x,fb) , Noeud(y,fc) when x < y -> Noeud(x,c::fb)
  | Noeud(x,fb) , Noeud(y,fc)            -> Noeud(y,b::fc)
  | _                                    -> failwith "erreur";;
```

On écrit ensuite une fonction qui s'inspire de l'addition avec retenue :

```
let rec insere r t = match t with
  | []      -> [r]
  | Vide::q -> r::q
  | a::q    -> let r' = combine a r in Vide::(insere r' q);;
```

On en déduit la fonction demandée :

```
let ajoute valeur tas = insere (Noeud(valeur,[])) tas;;
```

Q.36 Le résultat sur la signature résulte directement des questions 31 et 32.

```
let rec fusion t1 t2 =
  let rec fusionne r t1 t2 = match (t1,t2) with
    |[],_              -> insere r t2
    |_,[]              -> insere r t1
    |Vide::q1,Vide::q2 -> r ::(fusion q1 q2)
    |Vide::q1,a2::q2   -> if r=Vide then a2 :: (fusionne Vide q1 q2)
                          else Vide :: (fusionne (combine r a2) q1 q2)
    |a1::q1,Vide::a2   -> if r=Vide then a1 :: (fusionne Vide q1 q2)
                          else Vide :: (fusionne (combine r a1) q1 q2
    |a1::q1,a2::q2     -> r:: (fusionne (combine a1 a2) q1 q2)
  in
    fusionne Vide t1 t2;;
```