Corrigé du TP sur les tas binomiaux

Partie I : les arbres non binaires

```
type 'a foret = Vide | Noeud of 'a *'a foret *'a foret ;;

let rec longueur = function
   | Vide -> 0
   | Noeud(_,_,f) -> 1 + longueur f ;;


let premier = function
   Vide -> failwith "la forêt est vide"
   | Noeud(a,f,_)-> Noeud(a,f,Vide) ;;
```

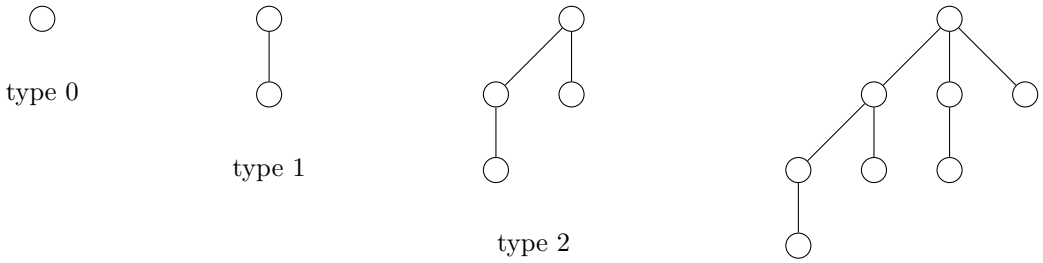
La forêt donnée en exemple se définit facilement (en écrivant les lignes qui suivent de bas en haut !) :

```
let f =
   let f17 = Noeud(17, Vide, Vide) in
   let f30 = Noeud(30, f17, Vide) in
   let f15 = Noeud(15, Vide, f30) in
   let f19 = Noeud(19, f15, Vide) in
   let f49 = Noeud(49, Vide, Vide) in
   let f6 = Noeud(6, Vide, Vide) in
   let f1 = Noeud(1, Vide, f6) in
   let f7 = Noeud(7, f1, f49) in
   let f3 = Noeud(3, Vide, f7) in
   Noeud(2,f3,f19) ;


let fils = function
   | Vide -> failwith "l'arbre est vide"
   | Noeud(a,f,_) -> f ;;


let rec taille = function
   | Vide -> 0
   | Noeud(_,f1,f2) -> 1+taille f1 + taille f2 ;;


let rec hauteur = function
   | Vide -> -1
   | Noeud(_,f1,f2) -> max (1+hauteur f1) (hauteur f2) ;;

let ajouter t f = match t with
```

```
  | Vide -> failwith "l'arbre est vide"
  | Noeud(a,f1,Vide) -> Noeud(a,f1,f)
  | _ -> failwith "l'argument n'est pas un arbre";;
```

Partie II : les arbres et les tas binomiaux

1) Nous avons directement :



Un arbre binomial de type k est de hauteur k, de taille 2^k et possèdent exactement $\binom{k}{i}$ nœuds au niveau i, pour tout i compris entre 0 et k, d'où la dénomination "d'arbre binomial". On peut également remarquer que les fils d'un arbre binomial de type k forment une forêts d'arbres binomiaux de type $k-1$, $k-2$, …, 0.

Il y a de nombreuses façons de calculer l'ordre k d'un arbre binomial, en un temps proportionnel à k. Par exemple, cet ordre est le nombre de fils de la racine, ce qui donne :

```
let ordre t = longueur(fils t) ;;
```

2) Si un tas binomial est formé d'arbres binomiaux de type $t_1 < t_2 < \cdots < t_p$, il contient $2^{t_1} + 2^{t_2} + \cdots + 2^{t_p}$ nœuds : pour que ce nombre soit égal à un entier n donné, il faut et il suffit que les t_i soient les positions des chiffres 1 dans l'écriture binaire de n. Pour un entier n donné, la suite t_i est donc imposée et les tas binomiaux permettant de stocker n clés seront donc tous géométriquement identiques. On peut également remarquer que $n \geq 2^{t_p} \geq 2^{p-1}$: un tas binomial stockant n clés contiendra donc au maximum $1 + \lfloor \ln_2 n \rfloor$ arbres.

3) Il suffit de traduire la définition, en mettant la plus petite valeur à la racine :

```
let fusion_arbre t1 t2 = match (t1,t2) with
  | (Noeud(a, f1, Vide), Noeud(b, f2, Vide)) when a<b -> Noeud(a, ajouter t2 f1, Vide)
  | (Noeud(a, f1, Vide), Noeud(b, f2, Vide)) -> Noeud(b, ajouter t1 f2, Vide)
  | _ -> failwith "les arguments doivent être des arbres binomiaux" ;;
```

Partie III : insertion d'une nouvelle clé et suppression de la plus petite clé dans un tas binomial

La fonction `union` prend en argument deux tas binomiaux $t1$ et $t2$ et renvoie la forêt obtenue en réunissant les deux tas, mais en conservant la croissance (au sens large) des ordres des arbres binomiaux. On utilise ici la fonction `ordre`, en remarquant qu'elle renvoie, quand on l'applique à un tas binomial non vide, l'ordre du premier arbre binomial du tas. L'analyse est élémentaire :

— si $t1$ (resp. $t2$) est vide, on renvoie $t2$ (resp $t1$) ;
— si le premier arbre de $t1$ est d'ordre strictement inférieur au premier arbre de $t2$, on applique récursivement la fonction `union` à $t1$ privé de son premier arbre et à $t2$, puis on ajoute le premier arbre de $t1$ en tête de la forêt ainsi obtenue.
— sinon, on effectue le même travail en échangeant les rôles de $t1$ et de $t2$.

```
let rec union t1 t2 = match (t1,t2) with
   | (Vide,_) -> t2
   | (_, Vide) -> t1
   | (Noeud(a, f1, f2), Noeud(b, f3, f4)) when ordre t1 < ordre t2
                        -> ajouter (Noeud(a, f1, Vide)) (union f2 t2)
   | (Noeud(a, f1, f2), Noeud(b, f3, f4)) -> ajouter (Noeud(b, f3, Vide)) (union t1 f4) ;;
```

Il reste ensuite à fusionner les arbres binomiaux de même ordre : cela se fait récursivement à l'aide de la fonction `fusion_forêt`. L'analyse de cette procédure est la suivante, $f = [T_1, T_2, \ldots, T_k]$ étant une forêt d'arbres binomiaux d'ordres croissants ne contenant pas plus de deux arbres de même ordre :

- si $k \leq 1$, on renvoie f ;
- si $k \geq 2$ et si l'ordre de T_1 est strictement inférieur à celui de T_2, on applique la fonction récursivement à la forêt $[T_2, T_3, \ldots, T_k]$ et on insère T_1 au début de la forêt obtenue ;
- sinon, plusieurs cas sont encore à considérer :
  — si $k = 2$, on renvoie l'arbre obtenu par fusion de T_1 et de T_2 ;
  — si $k \geq 3$ et si T_3 est de même ordre que T_1 et T_2, on fusionne T_2 et T_3 en un arbre T, on applique récursivement `fusion_foret` à $[T, T_4, \ldots, T_k]$ et on insère T_1 eu début de la forêt obtenue ;
  — sinon, on fusionne T_1 et T_2 en un arbre T et on applique la fonction récursivement à $[T, T_3, \ldots, T_k]$.

```
let rec fusion_foret f = match f with
   | Vide -> Vide
   | Noeud(_,_,Vide) -> f
   | Noeud(a,f1,Noeud(b,f2,f3)) ->
          let t1 = Noeud(a, f1, Vide) and t2 = Noeud(b, f2, Vide) in
          if (ordre t1) < (ordre t2) then
             ajouter t1 (fusion_foret (Noeud(b, f2, f3)))
          else
             match f3 with
                | Vide -> fusion_arbre t1 t2
                | Noeud(c,g,h) -> let t3 = noeud(c,g,Vide) in
                   if ordre t1 = ordre t3 then
                      ajouter t1 (fusion_foret (ajouter (fusion_arbre t2 t3) h))
                   else
```

```
                    fusion_foret (ajouter (fusion_arbre t1 t2) f3) ;;

let fusion f1 f2 = fusion_foret (union f1 f2) ;;
```

2) Pour insérer une nouvelle clé a dans un tas binomial f, il suffit de fusionner f et le tas formé par un unique arbre d'ordre 0 contenant a :

```
let inserer a f = fusion (Noeud(a, Vide, Vide)) f ;;
```

3) On commence par calculer α en comparant les racines des arbres, grâce à la fonction :

```
let rec minimum = function
   | Vide -> max_int
   | Noeud(a,_,f) -> min a (minimum f) ;;
```

Une fois α calculé, en posant $f = [T_1, T_2, \ldots, T_k]$ et en notant i l'indice de l'arbre contenant α à la racine, on calcul les forêts $f_1 =$ `fils` T_i et $f_2 = [T_1, \ldots, T_{i-1}, T_{i+1}, \ldots, T_k]$. Il reste un dernier piège : il faut inverser l'ordre des arbres de la forêt f_1 avant de la fusionner avec f_2.

```
let rec separer t alpha = match t with
   | Vide -> failwith "la valeur n'est pas correcte"
   | Noeud(a,f1,f2) when a = alpha -> f1, f2
   | Noeud(a,f1,f2) -> let (g1,g2) = separer f2 alpha in
                           g1, ajouter (Noeud(a, f1, Vide)) g2 ;;

let inverser f = let g = ref f and h = ref Vide in
   while !g <> Vide do
      match !g with
         | Noeud(a, g1, g2) -> h := ajouter (Noeud(a, g1, Vide)) !h ; g := g2
   done ;
   !h ;;

let supprimer_minimum t = let f1,f2 = separer t (minimum t) in
   fusion (inverser f1) f2 ;;
```