MP-MP* option info TP1

La structure de tas binomial

Stéphane Legros Année scolaire 2021-2022

Partie I : les arbres non binaires

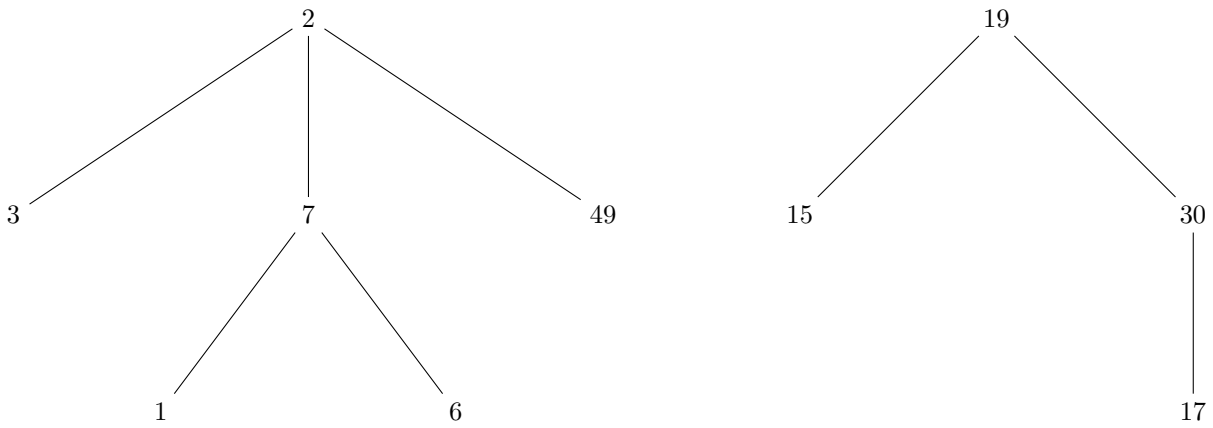
Nous souhaitons manipuler des arbres dont les nœuds, contrairement aux cas des arbres binaires, possèdent un nombre de fils non limité à 2. Nous appellerons *forêt* une liste $[T_1, T_2, \ldots, T_k]$ d'arbres non vides.

1) Expliquer comment le type

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

utilisé habituellement pour définir une structure d'arbre binaire permet aussi de gérer les forêts d'arbres quelconques, en convenant que `Noeud(`a`,`F_1`,`F_2`)` représente la forêt dont le premier arbre contient a à la racine et admet F_1 pour "forêt de fils" et F_2 pour "forêt de frères". En d'autres termes, on peut dire que F_1 "pointe" vers le fils aîné (i.e. vers le fils le plus à gauche) et que F_2 "pointe" vers le frère droit (T_2 étant alors considéré comme le frère droit de T_1). En particulier, cette forêt est réduite à un arbre si et seulement si F_2 est vide : le type `'a foret` permet donc de représenter à la fois les arbres et les forêts. Dans la suite, nous ne manipulerons que des arbres d'entiers, mais le problème pourrait être adapté à tout type de données totalement ordonnées.

2) Écrire des fonctions `longueur` et `premier` qui, appliquées à une forêt, renvoient le nombre d'arbres qui la constituent et, si elle est non vide, le premier de ces arbres. écrire une fonction `fils` qui, appliquée à un arbre, renvoie la forêt de ses fils. Appliquer vos fonctions à la forêt de longueur 2 :



3) Écrire des fonctions qui, appliquées à une forêt, renvoient respectivement sa taille (i.e. le nombre total de ses nœuds) et sa hauteur (i.e. le maximum des hauteurs des arbres qui la constituent).

4) Écrire une fonction `ajouter` qui, appliquée à une forêt $[T_1, T_2, \ldots, T_k]$ et à un arbre non vide T, renvoie la forêt $[T, T_1, T_2, \ldots, T_k]$.

Partie II : les arbres et les tas binomiaux

Les *arbres binomiaux* sont définis par induction : un arbre binomial d'ordre 0 est réduit à une feuille et un arbre binomial d'ordre $k+1$ (pour $k \geq 0$) est construit à l'aide de deux arbres binomiaux T_1 et T_2 d'ordre k, T_2 étant inséré dans T_1 comme fils aîné (T_2 est le fils aîné de la racine de T_1 et l'ancien fils aîné devient le frère droit de T_2).

Un tas binomial est une forêt d'arbres binomiaux telle que :

- les ordres des arbres constituant la forêt croissent strictement ;
- chaque nœud de chaque arbre a un contenu inférieur aux contenus de ses fils éventuels.

1) Représenter la structure géométrique d'un arbre binomial d'ordre k pour k compris entre 0 et 3. Quelles sont la taille et la hauteur d'un arbre binomial d'ordre k ? Pourquoi ces arbres sont-ils qualifiés de *binomiaux* ? écrire une procédure `ordre` qui calcule l'ordre d'un arbre supposé binomial.

2) Montrer qu'il existe une unique structure de tas binomial permettant de stocker un nombre fixé n de clés.

3) Écrire une procédure `fusion_arbre` qui fusionne deux arbres binomiaux de même taille en un unique arbre binomial, en respectant la condition d'ordre imposées aux clés.

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

1) Écrire une procédure `fusion` qui fusionne deux tas binomiaux en un unique tas binomial. On commencera par transformer les deux tas en une forêt d'arbres binomiaux d'ordres croissants au sens large. Pour tester cette fonction, on utilisera quelques tas binomiaux créés par le biais des fonctions :

```
let rec bino i = match i with
   | 0 -> Noeud(Random.int 10000, Vide, Vide)
   | _ -> fusion_arbre (bino (i-1)) (bino(i-1));;

let rec tas = function
   | [] -> Vide
   | n::q -> ajouter (bino n) (tas q);;
```

et on affichera la liste des ordres des arbres d'un tas grâce à la fonction :

```
let rec imprimer f=match f with
   | Vide -> print_char ' '
   | Noeud(a,t1,f1) -> print_int (ordre f); print_char ' '; imprimer f1;;
```

2) En déduire une fonction élémentaire permettant d'insérer une clé dans un tas binomial.

3) On cherche à supprimer la plus petite clé α d'un tas binomial. Où se trouve cette clé ? écrire une fonction qui, appliquée à un tas binomial non vide, renvoie un couple de tas binomiaux contenant toutes les clés du tas initial, exceptée la clé α. En déduire une fonction de suppression de la plus petite clé d'un tas binomial.