1 Arbres n-aires

1.1 Définition

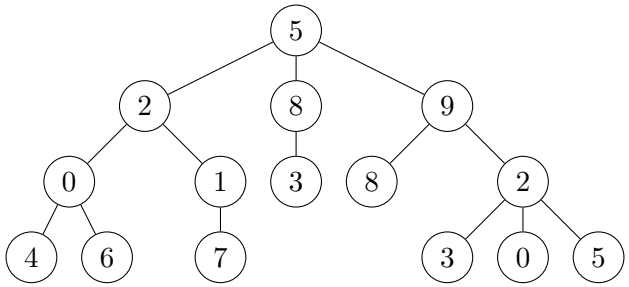
Soit E un ensemble.

On définit récursivement l'ensemble $\mathscr{A}(E)$ des arbres n-aires sur E :

— si $x \in E$, alors $(x, ())$ appartient à $\mathscr{A}(E)$;

— si $x \in E$, $k \in$ et $(A_0, A_1, \ldots, A_{k-1}) \in \mathscr{A}(E)^k$, alors $(x, (A_0, \ldots, A_k)) \in \mathscr{A}(E)$.

Avec cette définition, il n'y a pas d'arbre vide. Tout arbre n-aire est de la forme $A = (x, (A_0, \ldots, A_{k-1}))$: x est appelé l'étiquette de l'arbre, $A_0, \ldots, A_{k-1}$ sont les fils de x. Éventuellement, on a $k = 0$: A n'a pas de fils, on dit que A est une feuille.



1.2 Feuilles et nœuds

On définit récursivement les feuilles $F(A)$ et les nœuds (internes) $N(A)$ d'un arbre A (listes car il peut y avoir des répétitions) :

si $A = (x, (A_0, \ldots, A_{k-1}))$, alors

— si $k = 0$, alors $F(A) = [A]$ et $N(A) = [\,]$;

— sinon $F(A) = \bigcirc_{i=0}^{k-1} F(A_i)$ et $N(B) = [A] \circ \bigcirc_{i=0}^{k-1} N(A_i)$.

($\circ$ désigne la concaténation)

Les feuilles et les nœuds sont en particulier des arbres. Un sommet de l'arbre est soit une feuille, soit un nœud (interne). Les feuilles sont donc les sommets qui n'ont pas de fils, les nœuds (internes) sont les sommets qui ont au moins un fils.

Définition. On appelle arité d'un nœud $(x, (A_0, \ldots, A_{k-1}))$ l'entier k, nombre de ses fils. L'objet x est appelé l'étiquette du nœud.

Un arbre est donc binaire si tous ses nœuds sont d'arité au plus 2. Plus généralement, un arbre est dit N-aire si tous ses nœuds sont d'arité au plus N (l'arbre de l'exemple ci-dessus est donc ternaire).

Remarque.

— La terminologie est relativement mouvante : certains auteurs appellent nœud ce qui est appelé ici sommet, les nœuds étant internes ou externes (= feuilles). En fonction du contexte, il est facile de s'adapter…

— En général, lorsqu'on parle des feuilles ou des nœuds, on identifie le sommet avec son étiquette. Avec l'arbre donné en exemple, on se permettra de parler de la feuille 7, ou du nœud 9. En cas d'ambiguïté, on ajoute les informations nécessaires comme la position du sommet dans l'arbre.

1.3 Hauteur

Définition. On définit là encore récursivement la hauteur d'un arbre A : si $A = (x, (A_0, \ldots, A_{k-1}))$, alors

— si $k = 0$, alors $h(A) = 0$;

— sinon $h(A) = 1 + \max_{0 \leqslant i \leqslant k-1} (h(A_i))$.

La hauteur d'un arbre est donc le nombre d'arêtes du plus long chemin menant de la racine à une feuille.
On parle aussi de la profondeur d'un sommet (ou d'un nœud) comme la longueur du chemin menant de la racine à ce sommet.
On peut généraliser le résultat sur le nombre de sommets et la hauteur, vu dans le cas des arbres binaires.

Proposition 1 *Si A est un arbre de hauteur h, possédant n sommets et d'arité maximale a, alors*

$$h+1 \leqslant n \leqslant \frac{a^{h+1}-1}{a-1}$$

ou encore

$$\lfloor \log_a((a-1)n) \rfloor \leqslant h \leqslant n-1$$

Remarque. On suppose bien sûr que $a > 1$, sinon A n'est plus vraiment un arbre, mais une liste, qui est un arbre unaire.

2 Arbres en CAML

Les arbres ne font pas partie des types natifs de CAML. Mais il est facile de les créer en utilisant les types abstraits.

2.1 Implémentation homogène

Tous les sommets contiennent le même type d'objet.
Sans distinguer les feuilles des nœuds (internes) :

```
type 'a arbre = S of 'a * 'a arbre list;;

let ar = S(5, [
           S(2,[]);
           S(8, [S(4,[])])
         ]);;

let rec conc ll =
  match ll with
  | [] -> []
  | l :: ql -> l @ conc ql;;

let rec feuilles ar =
  match ar with
  | S(a,[]) -> [a]
  | S(a,l) -> conc (map feuilles l);;


let feuilles ar =
  match ar with
  | S(a,l) -> list_it (prefix @) (map feuilles l) [];;
```

En distinguant les feuilles des nœuds (internes) :

```
type 'a arbre = F of 'a | N of 'a * 'a arbre list;;

let ar = N(5, [F(2); F(3); N(0,[F(1); F(8)]) ]);;

let feuilles ar =
  match ar with
  | F(a) -> [a]
  | N(a,l) -> conc (map feuilles l);;
```

2.2 Implémentation hétérogène

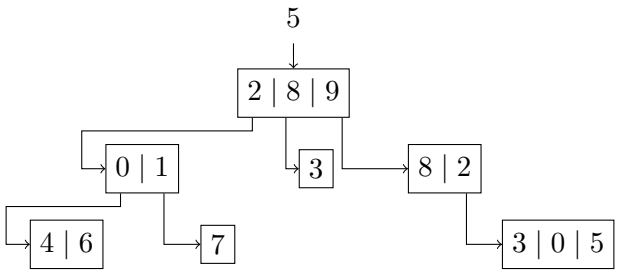
Pour les arbres « hétérogènes », on a deux solutions :
- soit on crée un type somme *ad hoc* pour réunir les types différents en un seul type ;
- soit on crée un type arbre qui prend en compte l'hétérogénéité des types.

```
type ('f,'n) arbre = F of 'f | N of 'n * ('f,'n) arbre list;;

let a = N("+", [F(2); F(7); N("-", [F(10); F(4)]); F(5) ]);;
```

3 Parcours d'arbres

Dans beaucoup d'algorithmes on doit parcourir récursivement les arbres n-aires. Il peut être intéressant d'avoir une autre représentation de tels arbres :



3.1 Parcours en profondeur

Il y a deux parcours récursifs de l'arbre, suivant l'ordre dans lequel on traite chaque sommet :
- parcours préfixe : racine, puis les fils dans l'ordre ;
- parcours postfixe (ou suffixe) : les fils dans l'ordre, puis racine.

La notion de parcours préfixe n'est pas définie dans le cas d'arbres non binaires.

Sur l'exemple du début du chapitre, cela donne :
- préfixe :
- postfixe :

3.2 Parcours en largeur

Cela consiste à parcourir l'arbre du haut vers le bas et sur chaque niveau, de gauche à droite.

Sur l'exemple précédent, cela donne :