Introduction aux arbres

1 Présentation et vocabulaire

1.1 Généralités

Un arbre est la donnée d'un *graphe connexe acyclique non orienté* et d'un sommet r de ce graphe appelé *racine* de l'arbre. Pour chaque sommet s du graphe, il existe alors un et un seul chemin de r à s.

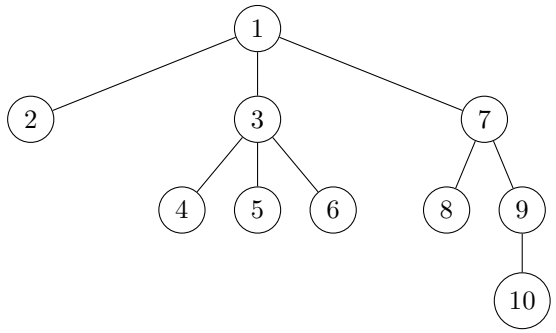


Figure 1 – Un arbre.

La racine induit une orientation implicite de l'arbre, du haut vers le bas.
Dans la figure 1, le sommet 1 est la racine de l'arbre.

Les sommets d'un arbre sont appelés des *nœuds*. Lorsqu'une arête mène du nœud i au nœud j, on dit que i est le *père* de j et que j est un *fils* de i.
La racine n'a pas de père, les autres nœuds possèdent un et un seul père. Un nœud peut avoir aucun, un seul ou plusieurs fils. L'*arité* d'un nœud est le nombre de fils de ce nœud.

Un nœud qui ne possède aucun fils est appelé une *feuille* de l'arbre. Un nœud qui possède au moins un fils est appelé un *nœud interne*.

La *profondeur* d'un nœud est sa distance par rapport à la racine. La *hauteur* d'un arbre est la profondeur maximale de ses nœuds.

Enfin, chaque nœud est la racine d'un arbre constitué de lui-même et de sa descendance : on parle alors de *sous-arbre* de l'arbre initial.

Exercice 1 *Pour l'arbre de la figure 1 :*

- *Les feuilles de l'arbre sont les nœuds* .....
- *L'arité du nœud* 1 *est .....*, *celle du nœud* 7 *est .....*
- *La profondeur du nœud* 3 *est .....*, *celle du nœud* 9 *est .....*
- *La hauteur de l'arbre est : .....*
- *Le sous-arbre associé au noeud* 3 *est :*

1.2 Arbres binaires

On appelle *arbre binaire strict* un arbre dans lequel l'arité de tous les nœuds internes est exactement 2. Chaque nœud interne possède alors un *fils gauche* et un *fils droit.*

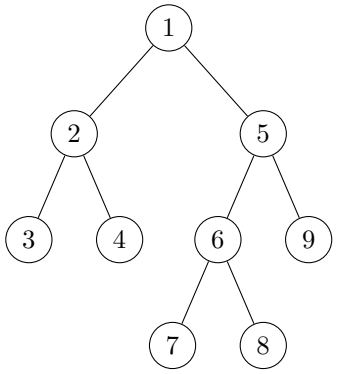


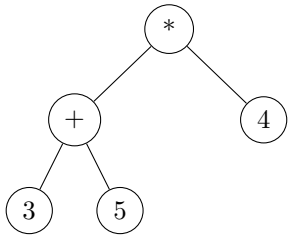
Figure 2 – Un arbre binaire strict.

Un arbre est alors soit une feuille, soit un nœud possédant un fils gauche et un fils droit.

En étiquetant les nœuds internes et les feuilles avec des étiquettes de types différents, on en déduit le type suivant :

```
type ('a, 'b) arbre =
  | Feuille of 'a
  | Noeud of 'b * ('a, 'b) arbre * ('a, 'b) arbre
;;
```

Exercice 2 *Représenter à l'aide du type précédent l'arbre :*



Plus généralement, un *arbre binaire* est un arbre dans lequel l'arité de chaque nœud est 0, 1 ou 2. Pour définir la structure de donnée associée, il est pratique de considérer qu'un arbre peut être vide, ce qui permet par exemple de définir le type suivant :

```
type 'a arbre =
  | Nil
  | Noeud of 'a * 'a arbre * 'a arbre
;;
```

Remarque :

- Les feuilles correspondent alors aux nœuds dont les deux fils sont vides.
- Ici, les étiquettes des feuilles et des nœuds internes sont de même type.

2 Quelques résultats sur les arbres binaires

On travaillera dans la suite avec le type défini juste au-dessus.

2.1 Nombre de nœuds et de feuilles

Exercice 3 *Écrire une fonction* `nb_noeuds : 'a arbre -> int` *et une fonction* `nb_feuilles : 'a arbre -> int` *prenant en argument un arbre et renvoyant respectivement le nombre de nœuds et le nombre de feuilles de l'arbre.*

Dans le cas d'un arbre binaire strict, on dispose du résultat suivant :

Théorème 1 *Soit $\mathcal{A}$ un arbre binaire strict (non vide). Si $\mathcal{A}$ possède n nœuds internes, alors $\mathcal{A}$ possède $n+1$ feuilles.*

Démonstration
Par récurrence forte sur n.

- **Initialisation.** Si $n = 0$, alors $\mathcal{A}$ ne comporte aucun nœud interne, donc la racine de $\mathcal{A}$ est une feuille. Par conséquent, $\mathcal{A}$ ne comporte qu'une feuille, donc $\mathcal{A}$ possède $n+1$ feuille.
- **Hérédité.** Soit $n \in \mathbb{N}$. Supposons la propriété vérifiée par les arbres comportant au plus n nœuds internes.

  Si $\mathcal{A}$ comporte $n+1$ nœuds internes, alors sa racine r est un nœud interne. Soient n_1 le nombre de nœuds internes du fils gauche de r et n_2 le nombre de nœuds internes du fils droit de n, alors $n+1 = n_1 + n_2 + 1$, donc $n_1 \leqslant n$ et $n_2 \leqslant n$. Par hypothèse de récurrence, le fils gauche de r comporte $n_1 + 1$ feuilles et le fils droit de r comporte $n_2 + 1$ feuilles, donc le nombre de feuilles de $\mathcal{A}$ est $n_1 + 1 + n_2 + 1 = (n+1) + 1$.

2.2 Hauteur d'un arbre

Exercice 4 *Écrire une fonction* `hauteur : 'a arbre -> int` *prenant en argument un arbre et renvoyant sa hauteur.*

On pourra convenir que la hauteur de l'arbre vide est -1.

.....

.....

.....

.....

.....

Théorème 2 *Soit $\mathcal{A}$ un arbre binaire de hauteur h et comportant n nœuds.*
Alors $h+1 \leqslant n \leqslant 2^{h+1}-1$.

Démonstration
Par récurrence forte sur h.

- **Initialisation.** Si $\mathcal{A}$ est de hauteur $h=0$, alors $\mathcal{A}$ ne contient que sa racine, donc son nombre de nœuds est $n=1$, et $h+1 \leqslant n \leqslant 2^{h+1}-1$.
- **Hérédité.** Soit $h \in \mathbb{N}$. Supposons que la propriété est vérifiée pour tout arbre de hauteur inférieure ou égale à h.

  Si $\mathcal{A}$ est de hauteur $h+1$, alors ses fils gauche et droite $\mathcal{A}_g$ et $\mathcal{A}_d$ sont de hauteur au plus h, donc comportent au plus $2^{h+1}-1$ nœuds. Par conséquent, $\mathcal{A}$ comporte au plus $2\times(2^{h+1}-1)+1$ i.e $2^{h+2}-1$ nœuds.

  De plus, l'un des deux sous-arbres $\mathcal{A}_g$ et $\mathcal{A}_d$ est de hauteur exactement h donc comporte au moins $h+1$ nœuds, donc $\mathcal{A}$ comporte au moins $h+2$ nœuds.

Théorème 3 *Soit $\mathcal{A}$ un arbre binaire de hauteur h et comportant f feuilles. Alors $f \leqslant 2^h$, donc $h \geqslant \lceil \log_2 f \rceil$.*

Démonstration
Même principe.

Remarque : Un arbre binaire de hauteur h comportant exactement $2^{h+1}-1$ nœuds est appelé un *arbre binaire complet*. On peut prouver par récurrence qu'un arbre binaire complet est strict et que ses feuilles ont toutes la même profondeur.

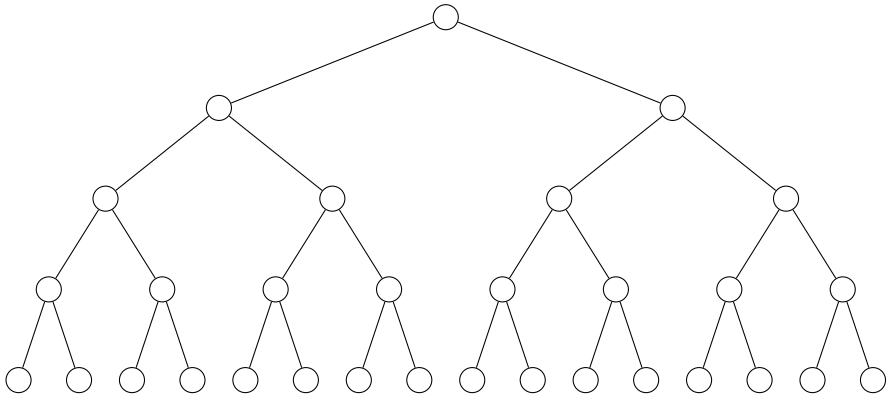


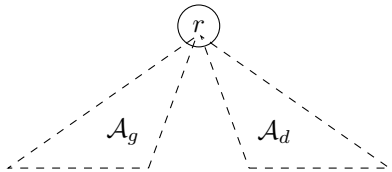
FIGURE 3 – Un arbre binaire complet.

3 Parcours d'arbres binaires

On souhaite énumérer les nœuds d'un arbre (pour stocker leurs étiquettes dans une liste ou encore pour les afficher). On distingue deux types de parcours : les parcours en *profondeur* et les parcours *en largeur*.

3.1 Parcours en profondeur

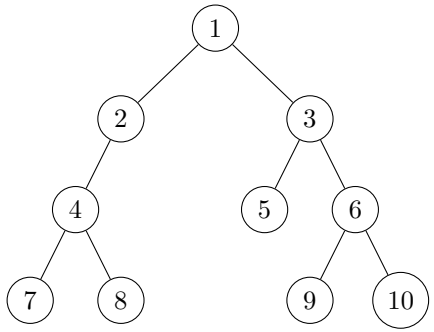
Lors d'un *parcours en profondeur* d'un arbre binaire non vide, comportant donc une racine et deux sous-arbres, chacun de ces sous-arbres est parcouru en entier avant de parcourir le sous-arbre suivant.



On distingue trois types de parcours en profondeur :

- le *parcours préfixe* : on parcourt la racine r, puis son fils gauche $\mathcal{A}_g$, puis son fils droit $\mathcal{A}_d$;
- le *parcours infixe* : on parcourt le fils gauche, puis r, puis le fils droit ;
- le *parcours postfixe* : on parcourt le fils gauche, puis le fils droit, puis r.

Exemple 1
Donner chacun des trois parcours en profondeur de l'arbre suivant :



- **Parcours préfixe :** .....
- **Parcours infixe :** .....
- **Parcours postfixe :** .....

Remarque : Dans le cas d'un arbre représentant une expression arithmétique, les parcours préfixe et postfixe correspondent respectivement aux représentations préfixe et postfixe [1] de l'expression.

Exercice 5 *Écrire des fonctions* `parcours_prefixe : 'a arbre -> 'a list`*,* `parcours_infixe : 'a arbre -> 'a list` *et* `parcours_postfixe : 'a arbre -> 'a list`*, prenant en argument un arbre et renvoyant la liste de ses étiquettes construite par chacun des trois parcours en profondeur.*

.....

.....

.....

.....

.....

.....

1. La notation postfixée est aussi appelée notation *polonaise inversée*

.....

.....

.....

3.2 Parcours en largeur

Le *parcours en largeur* consiste à parcourir les nœuds par profondeur croissante, de gauche à droite.

Dans le cas de l'arbre précédent, le parcours en largeur serait : 1-2-3-4-5-6-7-8-9-10.

Exercice 6 *Écrire des fonctions* `parcours_largeur : 'a arbre -> 'a list`*, prenant en argument un arbre et renvoyant la liste de ses étiquettes construite par un parcours en largeur.*

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

4 Exercices

Exercice 7 *On considère un arbre dont les nœuds sont étiquetés par des entiers.*

1. *Écrire une fonction prenant en argument un arbre et renvoyant le maximum de ses étiquettes.*
2. *On considère pour chaque branche de l'arbre (i.e un chemin de la racine à une feuille) la somme des étiquettes de ses nœds. Écrire une fonction donnant la somme maximale des branches d'un arbre.*

Exercice 8 *La numérotation de Sosa-Stradonitz d'un arbre binaire strict, utilisée pour identifier des individus dans une généalogie ascendante, consiste à numéroter les nœuds d'un arbre suivant les règles suivantes :*

- *La racine porte le numéro* 1 *;*
- *Si un nœud porte le numéro* n*, alors son fils gauche porte le numéro* $2n$ *et son fils droit le numéro* $2n+1$*.*

1. *Montrer que deux nœuds distincts ne peuvent pas porter le même numéro.*
2. *Montrer que les nœuds de profondeur* p *portent des numéros compris entre* 2^p *et* $2^{p+1}-1$*.*
3. *Déterminer la profondeur d'un nœud de numéro* n*.*
4. *Écrire une fonction* `arbre_binaire_complet : int -> int arbre` *prenant en argument un entier* h *et renvoyant un arbre binaire complet de hauteur* h *dont les étiquettes correspondent à la numérotation de Sosa-Stradonitz.*
5. *Écrire une fonction* `numeroter :'a arbre -> ('a * int) arbre` *qui prend en argument un arbre et qui numérote ses nœuds.*

Exercice 9 *On considère dans cet exercice des arbres avec des arités quelconques.*

1. *Proposer un type permettant de représenter ces arbres.*
2. *Réécrire les fonctions calculant le nombre de nœuds d'un arbre, son nombre de feuille, sa hauteur.*

Exercice 10 *Étant donné un arbre binaire* $\mathcal{A}$*, on appelle* diamètre *de* $\mathcal{A}$ *la longueur d'un plus long chemin dans* $\mathcal{A}$*.*

1. *Quel est la longueur d'un plus long chemin passant par la racine ?*
2. *Écrire une fonction* `diametre : 'a arbre -> int` *calculant le diamètre d'un arbre binaire ; on emploiera une technique « diviser pour régner »*

Exercice 11 *En choisissant un type d'arbres adapté, écrire une fonction prenant en argument un arbre d'expression arithmétique et renvoyant la valeur associée à l'expression.*

Exercice 12 (e3a 2017) *On suppose défini le type arbre de la manière suivante :*

```
type arbre =
   |Feuille of int
   |Noeud of arbre * arbre ;;
```

On dit qu'un arbre est un peigne si tous les noeuds à l'exception éventuelle de la racine ont au moins une feuille pour fils. On dit qu'un peigne est un strict si sa racine a au moins une feuille pour fils, ou s'il est réduit à une feuille. On dit qu'un peigne est rangé si le fils droit d'un noeud est toujours une feuille. Un arbre réduit à une feuille est un peigne rangé.

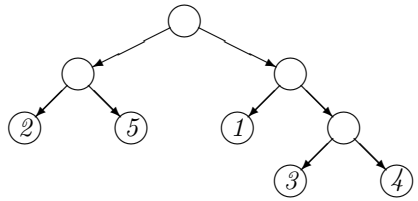


Figure 1 : un peigne à cinq feuilles

1. *Représenter un peigne rangé à* 5 *feuilles.*
2. *La hauteur d'un arbre est le nombre de noeuds maximal que l'on rencontre pour aller de la racine à une feuille (la hauteur d'une feuille seule est* 0*). Quelle est la hauteur d'un peigne rangé à* n *feuilles ? On justifiera la réponse.*
3. *Ecrire une fonction* `est_range : arbre → bool` *qui renvoie* `true` *si l'arbre donné en argument est un peigne rangé.*
4. *Ecrire une fonction* `est_peigne_strict : arbre → bool` *qui renvoie* `true` *si l'arbre donné en argument est un peigne strict. En déduire une fonction* `est_peigne : arbre → bool` *qui renvoie* `true` *si l'arbre donné en argument est un peigne.*
5. *On souhaite ranger un peigne donné. Supposons que le fils droit* N *de sa racine ne soit pas une feuille. Notons* A_1 *le sous-arbre gauche de la racine,* f *l'une des feuilles du noeud* N *et* A_2 *l'autre sous-arbre du noeud* N*. On va utiliser l'opération de rotation qui construit un nouveau peigne où*

- *le fils droit de la racine est le sous-arbre A_2 ;*
- *le fils gauche de la racine est un noeud de sous-arbre gauche A_1 et de sous-rabre droit la feuille f.*

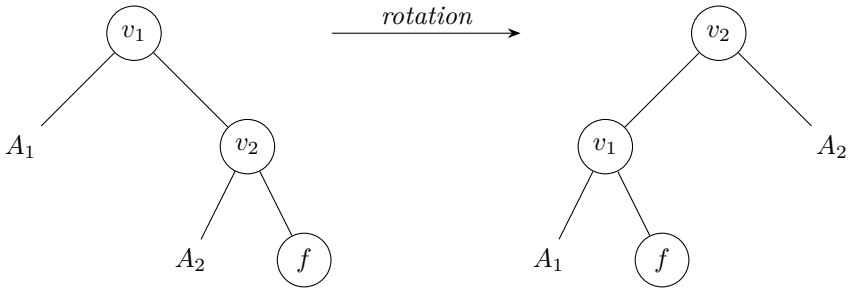


Figure 2 : une rotation

(a) Donner le résultat d'une rotation sur l'arbre de la figure 1.

(b) Ecrire une fonction `rotation : arbre` $\to$ `arbre` *qui effectue l'opération décrite ci-dessus. La fonction renverra l'arbre initial si une rotation n'est pas possible.*

(c) Ecrire une fonction `rangement : arbre` $\to$ `arbre` *qui range un peigne donné en argument, c'est à dire qui renvoie un peigne rangé ayant les mêmes feuilles que celui donné en argument. La fonction renverra l'arbre initial si celui-ci n'est pas un peigne.*