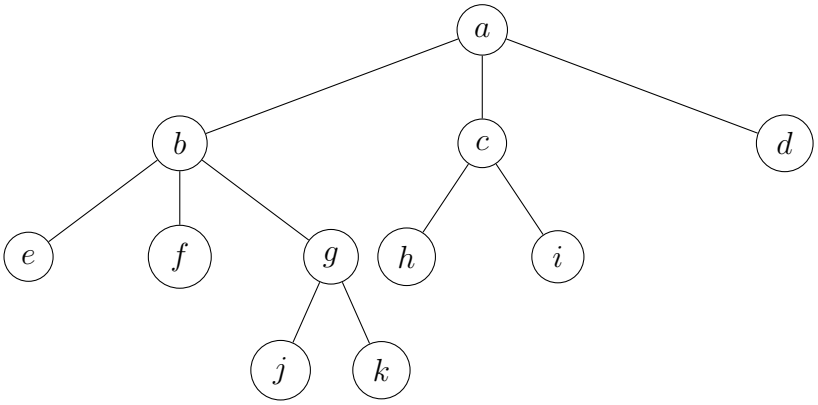
I - Notion mathématique d'arbre.

1) Introduction : « structure arborescente » et relation d'ordre partiel.

(a) « Structure arborescente » $\rightarrow$ relation d'ordre partiel.

Considérons un arbre A :



A définit une relation de précédence $<$: $a < b$, $a < c$, $b < e$, ...

Nous notons $\leq$ la clôture réflexive et transitive de $<$.

Par exemple : $a \leq b \leq g \leq k$.

Et $\leq$ est une relation d'ordre partiel sur A , autrement dit $(A, \leq)$ est un ensemble partiellement ordonné.

(b) Relation d'ordre partiel $\rightarrow$ « Structure arborescente » .

A désigne l'ensemble des entiers naturels inférieurs ou égaux à 10 et nous décidons de noter pour a et b appartenant à A :

$\Leftrightarrow a \,|\, b$.

Proposer alors une structure arborescente de l'ensemble $(A, |)$.

2) **Définition mathématique des arbres.**

Soit $(A, \leq)$ un ensemble partiellement ordonné.

a appartenant à A , nous notons :

i. $\text{Maj}(a) = \{x \in A / a \leq x\}$, ensemble des majorants de a ,

ii. $\text{Maj}^+(a) = \{x \in A / a \leq x \text{ et } x \neq a\}$, ensemble des majorants stricts de a ,

iii. $\text{Min}(a) = \{x \in A / x \leq a\}$, ensemble des minorants de a ,

iv. $\text{Min}^+(a) = \{x \in A / x \leq a \text{ et } x \neq a\}$, ensemble des minorants stricts de a .

Définition : On appelle **arbre** , tout ensemble partiellement ordonné $(A, \leq)$ tel que :

— A est un ensemble fini,
— A possède un plus petit élément $\min(A)$ appelé **racine de A** ,
— pour tout a appartenant à A , $\text{Min}(a)$ est une partie totalement ordonnée de A .

Remarque : Nous pouvons considérer que $\emptyset$ est ou non un arbre.

3) **Vocabulaire sur les arbres.**

Soit $(A, \leq)$ un arbre .

Par définition, **la racine de l'arbre** A est $\min(A)$.

(a) Sommets, feuilles et noeuds.

Quelques définitions :

— **sommet** : tout élément de A ,
— **feuille** : tout élément maximal de A , c'est-à-dire un élément a de A tel qu'aucun élément de A ne majore strictement a ,
— **noeud** : tout élément non-maximal de A .

(b) Ascendant, descendant ; père, fils.

— Soit a appartenant A distinct de la racine A ,
alors $\text{Min}^+(a)$ est une partie non vide de A et totalement ordonnée, donc elle admet un plus grand élément,
et **un ascendant de a** est un élément de $\text{Min}(a)$, **un ascendant strict de a** est un élément de $\text{Min}^+(a)$, **LE père de** a est $\max\left(\text{Min}^+(a)\right)$.
— Soit a appartenant A tel que a ne soit pas une feuille de A ,
alors $\text{Maj}^+(a)$ est une partie non vide de A et partiellement ordonnée, donc elle admet un nombre non nul d'éléments minimaux,
et **un descendant de a** est un élément de $\text{Maj}(a)$, **un descendant strict de a** est un élément de $\text{Maj}^+(a)$, **UN fils de** a est un élément minimal de $\text{Maj}^+(a)$.

Propriété :

i. Tout sommet autre que la racine est le fils de son père.

ii. Tout sommet autre qu'une feuille est le père de chacun de ses fils.

(c) Arêtes, chemins.

— **Arête :** couple de sommets en relation père-fils ,

— **Chemin entre deux sommets a et b :** séquence $s_0, s_1, \ldots, s_n$ de sommets tels que :

$$s_0 = a\,,\ s_n = b\,,\ \forall\, i \in [\![0, n-1]\!], (s_i, s_{i+1}) \text{ est une arête}\,,$$

n est la longueur du chemin.

— **Chemin direct entre deux sommets a et b :** chemin entre a et b ne passant pas deux fois par le même sommet (on remonte en fait à l'ascendant commun le plus proche et c'est le chemin le plus court).

(d) Forêts,sous-arbres et branches.

— **Forêt :** même définition qu'un arbre sans la condition d'existence d'un plus petit élément,
par exemple, $\mathrm{Maj}^+(racine)$ est une forêt.

— **Sous-arbre de A de racine b** : $\mathrm{Maj}(b)$.

— **Branches issues d'un sommet b** : c'est la forêt $\mathrm{Maj}^+(b)$.

(e) Hauteur, profondeur.

— **Hauteur d'un arbre A :** longueur maximale d'un chemin direct allant de la racine à une feuille.
La hauteur de l'arbre vide n'est pas définie.

— **Profondeur d'un sommet a** : longueur du chemin direct allant de la racine à a .

4) **Arité d'un sommet.**

Arité d'un sommet a : nombre de fils du sommet a ,

les feuilles sont d'arité nulle, les noeuds sont d'arité non nulle,

les sommets d'arité 1 sont dits **unaires**, les sommets d'arité 2 sont dits **binaires**.

n appartenant à $\mathbb{N}^*$, un **arbre n-aire** est un arbre dont tous les noeuds sont d'arité inférieure ou égale à n ,

un **arbre n-aire entier** est un arbre dont tous les noeuds sont d'arité égale à n .

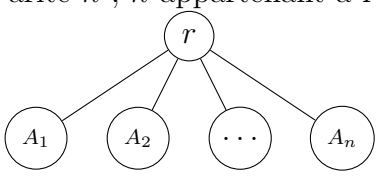
5) **Définition des arbres sur un ensemble Σ .**

Soit Σ un ensemble.

Définition récursive :

— L'arbre vide est un arbre sur Σ .

— r étant un élément de Σ d'arité n , n appartenant à $\mathbb{N}^*$, A_1 , A_2 , ... , A_n étant n arbres sur Σ, alors



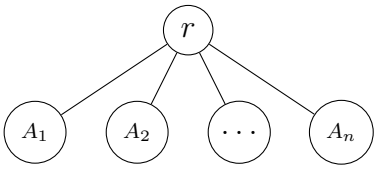
est un arbre sur Σ .

Remarque : La liste est un arbre unaire (entier).

Définition (récursive) de la hauteur $h(A)$ d'un arbre A :

— si A est l'arbre vide, alors : $h(A) = \ldots$

— si A est un arbre réduit à la racine, alors : $h(A) = \ldots$

— sinon A est la forme : 

où r est une racine d'arité non nulle n , alors :

$$h(A) = \dots$$

Remarque : De sa définition récursive, les objets définis sur un arbre et les preuves sur les arbres se feront par induction structurelle.

6) **Combinatoire sur les arbres.**

Un arbre non vide à n sommets possède ? arêtes.

Un arbre binaire entier non vide à n noeuds possède ? feuilles.

Un arbre binaire non vide à n noeuds et de hauteur h vérifie « fct de $h \leq n \leq$ fct de h » .

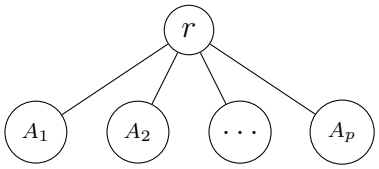
(a) Lien entre le nombre de sommets et le nombre d'arêtes.

Propriété : Soit n un entier naturel non nul.

Tout arbre (non vide) à n sommets possède ..... arêtes.

Démonstration : Par induction structurelle à partir de la définition récursive d'arbre.

Cas de base : Soit un arbre à 1 sommet, alors ...

Etape d'induction : Soit un arbre $A =$ 

,

r racine d'arité p , A_1 , A_2 , ... , A_p p arbres non vides vérifiant la propriété, alors ...

Par induction structurelle, le résultat est vrai. □

(b) Lien entre le nombre de noeuds et de feuilles d'un arbre binaire entier.

Propriété : Soit n un entier naturel.

Tout arbre binaire entier non vide à n noeuds possède ..... feuilles.

Démonstration : Par induction structurelle à partir de la définition récursive d'arbre.

(c) Lien entre le nombre de noeuds et la hauteur d'un arbre binaire.
Propriété : Soit n et h deux entiers naturels.
Tout arbre binaire non vide à n noeuds et de hauteur h vérifie :

$$h \leq n \leq 2^h - 1 \quad , \quad \lceil \mathrm{Log}_2(n+1) \rceil \leq h \leq n$$

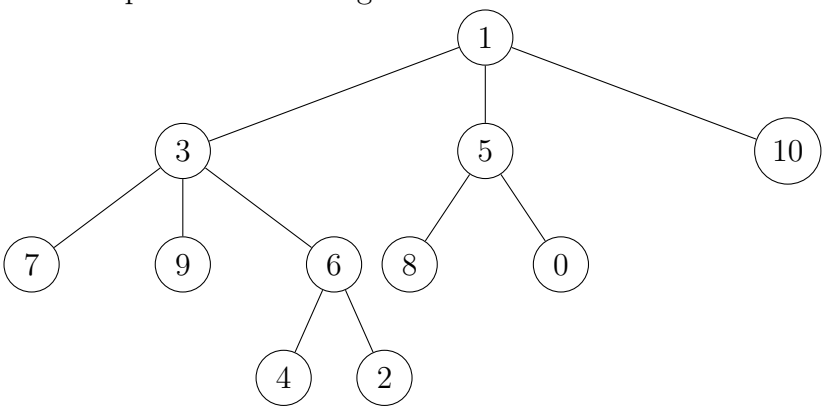
où x appartenant à $\mathbb{R}$, $\lceil x \rceil$ est la partie entière supérieure de x .
Démonstration : Par induction structurelle à partir de la définition récursive d'arbre.

Remarque : La complexité de nombreux algorithmes sur les arbres se calcule en fonction de la hauteur de ces arbres, d'où l'idée d'avoir la hauteur la plus petite possible.
Dans le cas des arbres binaires :

— configuration la pire : $n = h$, configuration de « peigne » nous sommes en fait ramenés à une liste,
— configuration la meilleure : $n = 2^h - 1$, configuration d'arbre binaire « complet » , configuration idéale mais réalisable que pour certaines valeurs de n .

7) **Applications particulières.**

(a) Etiquettes.
Nous pouvons stocker des informations soit au niveau des noeuds, des feuilles ou des sommets.
Lorsque les informations sur les noeuds et les feuilles sont de même type, nous parlons d'**arbre homogène**.
Lorsque les informations sur les noeuds et les feuilles ne sont pas de même type, nous parlons d'**arbre hétérogène**.
Un exemple d'arbre homogène :



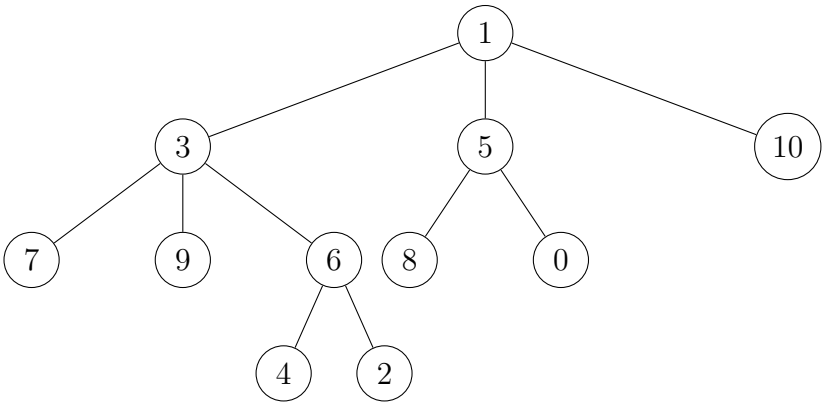
Un exemple d'arbre hétérogène pour représenter la formule mathématique : $\sin(3.0 + 4.0) \times \ln(2.0 \times \pi)$

(b) Positions.

Nous cherchons à référencer les sommets d'un arbre. La relation $\leq$ de la définition initiale est souvent oubliée voire inconnue.

Les sommets d'un arbre A sont référencés par des mots dont les lettres appartiennent à l'alphabet $[\![1, M]\!]$ ou $[\![0, M-1]\!]$ où M désigne le maximum des arités des noeuds de A .

Par exemple, dans l'arbre



6 est en position 1|3 ou 3|1 , 0 est en position .....

8) **Parcours d'arbres.**

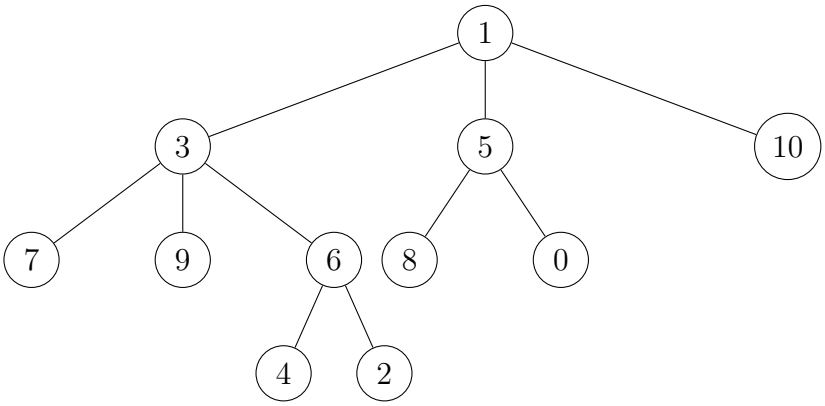
i. Parcours en profondeur.

Chaque sommet est visité un nombre de fois égal à son arité +1 .

Nous avons le parcours en profondeur en mode préfixe, traitement du sommet à la première visite.

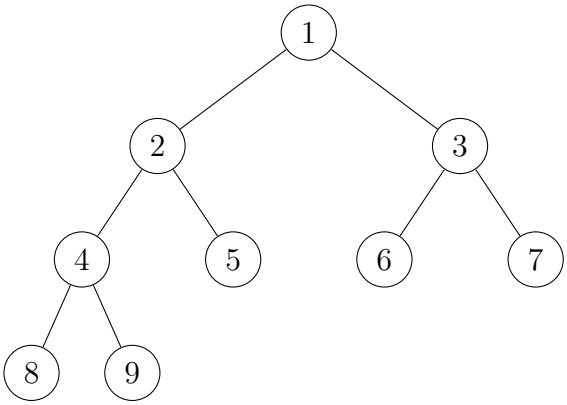
Nous avons le parcours en profondeur en mode postfixe, traitement du sommet à la dernière visite.

Par exemple, étant donné l'arbre



le parcours en profondeur en mode préfixe est $[1; 3; 7; 9; 6; 4; 2; 5; 8; 0; 10]$,
le parcours en profondeur en mode postfixe est .....
Et dans le cas des arbre binaire entier, nous avons le parcours en profondeur en mode infixe, traitement du sommet à la deuxième visite.

ii. Parcours en largeur.
Parcours de l'arbre « étage par étage » .
Sur l'exemple précédent,le parcours en largeur est $[1; 3; 5; 10; 7; 9; 6; 8; 0; 4; 2]$.
Dans le cas d'un arbre binaire entier :



la numérotation correspond au parcours en largeur.
Nous pouvons remarquer que :
— numéro du fils gauche = numéro du sommet multiplié par 2 ,
— numéro du fils droit = ...
— numéro du père = ...
Un tel arbre est avantageusement codé en tableau :

		père		sommet		fg	fd	
		$n/2$		n		$2n$	$2n+1$	

II - Notion informatique d'arbre.

Tous les arbres sont étiquetés et ordonnés (par leur position).

1) Arbres binaires

Nous distinguons les arbres binaires entiers homogènes et les arbres binaires entiers hétérogènes.

(a) Arbres binaires entiers homogènes.

Nous avons deux variantes, soit tous les sommets ont des étiquettes de même type, soit tous les noeuds ont des étiquettes de même type et les feuilles sont des arbres vides.

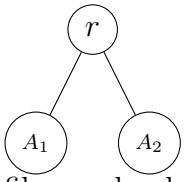
i. Structure de données abstraite.

Dans ce cadre, l'arbre vide est considéré comme un arbre.

— Description.

Définition récursive d'un arbre binaire entier homogène Ab_homo :

→ l'arbre vide est un un arbre binaire entier homogène ,

→ si A_1 et A_2 sont deux arbres binaires entiers homogènes et r un élément d'arité 2 , A = 

est un arbre binaire entier homogène , A_1 est le « fils gauche de A » , A_2 le « fils droit de A » .

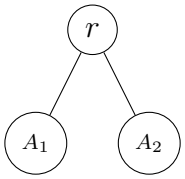
$$\text{Ab_homo} = \text{Arbre_vide} + \text{Elt} \times \text{Ab_homo} \times \text{Ab_homo}$$

— Opérations de base.

→ Constructeurs.

arbre_vide : $\to$ Ab_homo , $\mapsto \varepsilon$

construire : Elt $\times$ Ab_homo $\times$ Ab_homo $\to$ Ab_homo ,

$(r, A_1, A_2) \mapsto A =$ 

.

→ Opérateurs.

racine : Ab_homo $\to$ Elt

fils_gauche : Ab_homo $\to$ Ab_homo

fils_droit : Ab_homo $\to$ Ab_homo

→ Opérateurs.

est_vide : Ab_homo $\to \mathbb{B}$

nb_sommets : Ab_homo $\to \mathbb{N}$

hauteur : Ab_homo $\to \mathbb{N}$

ii. Structure de données concrète : implémentation en Caml.

— Déclaration d'un type « arbre binaire entier homogène » .

Un type union avec argument polymorphe et récursif :

```
type 'a ab_homo =
  | Vide
  | Noeud of 'a * ('a ab_homo) * ('a ab_homo)
;;
```

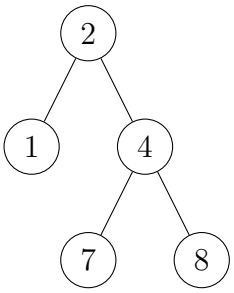
ou plus simplement :

```
type  abh =
  | V
  | N of int * abh *  abh
;;
```

— Programmation des opérations de base.

```
let arbre_vide () =

let construire r fg fd =

let racine a =

let fils_gauche a =

let fils_droit a =

let est_vide a =
```

— Saisie d'un arbre.



Saisir l'arbre A =

```
let a =
```

— Autres opérations.

```
let rec nb_noeuds a =

let rec nb_feuilles a =

let rec hauteur a =
```

iii. Parcours d'arbre.
Plus simple, le parcours en profondeur ... les fonctions renvoient la liste des étiquettes.

```
let rec parcours_profondeur_prefixe a =

let rec parcours_profondeur_postfixe a =

let rec parcours_profondeur_infixe a =
```

Plus délicat, le parcours en largeur ... la fonction renvoie la liste des étiquettes, une fonction auxiliaire récursive est attendue.

```
let  parcours_largeur a =
```

(b) Arbres binaires entiers hétérogènes.

F désignant le type des étiquettes d'une feuille, N désignant le type des étiquettes d'un noeud,

le type E des étiquettes d'un sommet est la réunion disjointe de F et N :

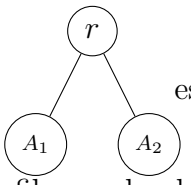
i. Structure de données abstraite.

Nous n'avons pas d'arbre vide.

— Description.

Définition récursive d'un arbre binaire entier hétérogène :

→ une feuille est un arbre binaire entier hétérogène,

→ si A_1 et A_2 sont deux arbres binaires entiers hétérogènes et r un élément de N , 

est un arbre binaire entier hétérogène, A_1 est le « fils gauche de A » , A_2 le « fils droit de A » .

$$\text{Ab_het} = \text{Feuille} + \text{Noeud} \times \text{Ab_het} \times \text{Ab_het}$$

— Opérations de base.

Nous avons seulement à adapter le cas des arbres homogènes.

ii. Structure de données concrète : implémentation en Caml.

— Déclaration d'un type arbre binaire entier hétérogène.

Un type union avec argument polymorphe et récursif :

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

ou pour un arbre binaire hétérogène :

```
type  abhet =
  |
  |
  |
;;
```

— Saisie d'un arbre.

Déterminer la représentation arborescente de la formule

$$(x + 2) + ((3 * y) - 4)$$

et définir alors l'arbre binaire entier hétérogène associé :

```
let a =
```

— Autres opérations.
Reprendre les questions traitées dans le cadre des arbres entiers homogènes.
Une remarque ... Pour le parcours, nous devons créer un type qui réunit les types `'f` et `'n` :

```
type elt = F of 'f | N of 'n ;;
```

Nous pouvons remarquer que la connaissance du parcours en profondeur préfixe (ou postfixe) permet de reconstituer l'arbre.
Nous pouvons chercher à écrire une fonction qui à partir de la liste des sommets parcourus lors d'un parcours en profondeur préfixe, reconstitue l'arbre.

```
let rec construire_parcours_prefixe a =
```

Proposer une solution pour un parcours en profondeur en mode postfixe :

```
let rec construire_parcours_postfixe a =
```

Est-ce possible avec la liste des sommets parcourus lors d'un parcours en mode infixe ?

2) **Arbres généraux.**

(a) Arbres généraux homogènes.

— avec un tableau des fils d'un noeud

```
type 'a arbre =
  | Vide
  | Noeud of 'a * ('a arbre array)
;;
```

— avec une liste des fils d'un noeud

```
type 'a arbre =
  | Vide
  | Noeud of 'a * ('a arbre list)
;;
```

(b) Arbres généraux hétérogènes.

— avec un tableau des fils d'un noeud

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

— avec une liste des fils d'un noeud

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