



# Chap 08 · Les arbres

---

Prise de note par Léo BERNARD en MP2I au lycée du Parc.

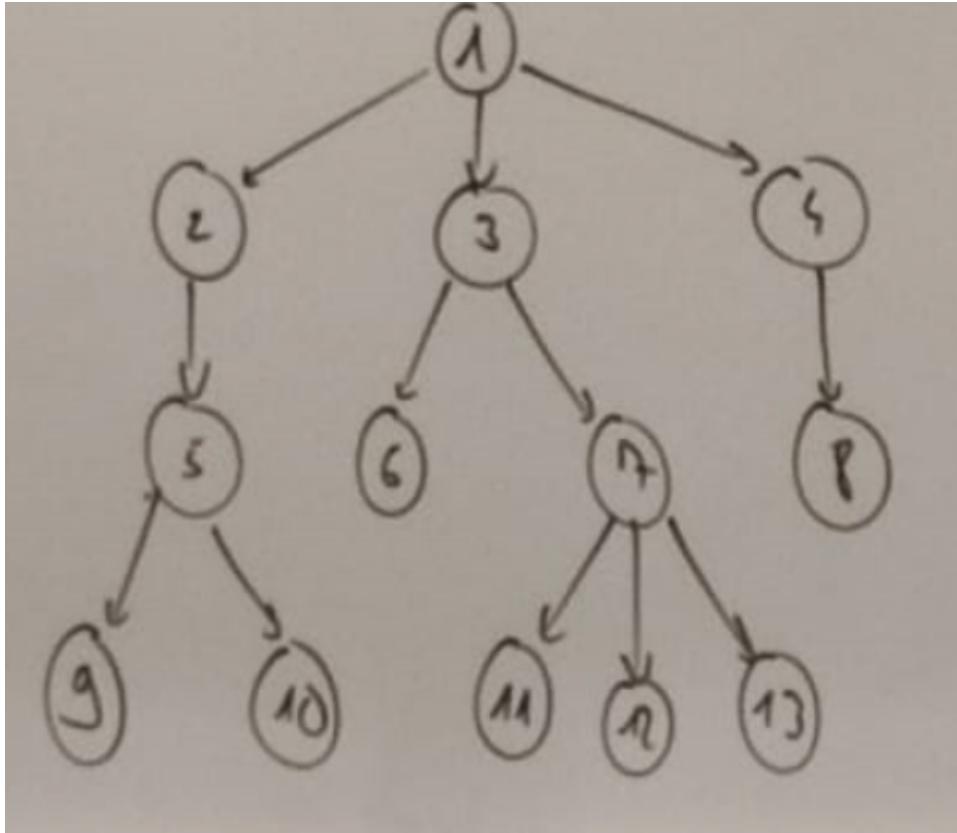


Pour nos amis les arbres à l'envers.

- Chap 08 · Les arbres
  - I. Definitions
    - I.A. Arbres binaires stricts
      - Induction structurelle
    - I.B. Arbres binaires non stricts
    - I.C. Parcours d'arbres binaires
      - I.C.1. Parcours en profondeur
      - I.C.2. Parcours en largeur
      - I.C.3. En OCaml
      - I.C.4. Unicité
    - I.D. Un peu de dénombrement
      - I.D.1 Relation entre hauteur, nombre de noeuds et nombre de feuilles
      - I.D.2. Nombres d'arbres binaires de taille fixée
  - II. Arbres non binaires
    - II.A. Lecture unique
    - II.B. Transformation en arbre binaire
- 

Un arbre est une structure de données qui peut se représenter sous forme hiérarchique.

 **Exemple:** un arbre est constitué de **nœuds** qui peuvent porter des étiquettes



Cet arbre contient 13 nœuds étiquetés de 1 à 13.

- Chaque nœud a un certain nombre d'enfants ou fils.
  - les enfants du nœud 1 sont les nœuds 2, 3, et 4
  - le seul enfant du nœud 2 est le nœud 3.
  - le seul nœud 9 n'a pas d'enfant.
- Le nombre d'enfants d'un nœud s'appelle son arité :
  - arité de (1) = 3
  - arité de (2) = 1
  - arité de (9) = 0
- Un nœud d'arité 0 s'appelle une feuille de l'arbre (ici, les feuilles sont 6, 8, 9, 10, 11, 12, 13)
- Un nœud d'arité non nulle s'appelle un nœud interne.
- Si  $x$  est un enfant de  $y$ , on dit que  $y$  est le parent ou le père de  $x$ .
- Dans un arbre, il existe un unique nœud qui n'a pas de père, on l'appelle la racine de l'arbre.
- Les ancêtres d'un nœud est l'ensemble de tous les nœuds "au dessus". Ancêtres de 7 = {3, 1}.
- Les enfants d'un même père sont dits frères.
- La profondeur d'un nœud est la longueur du chemin qui le relie à la racine.
  - La racine est à profondeur 0.
  - Ses enfants sont à profondeur 1
  - Ses petits-enfants à profondeur 2
- La hauteur de l'arbre est la profondeur maximale d'un nœud. Ici 3.

👉 **Remarque:** Il existe de nombreuses variations autour de cette définition sur la structure même de l'arbre (nb de fils, place des étiquettes...)  
 Sur les définitions (hauteur = 3 ou 4 sur cet exemple)  
 La convention utilisée sera toujours mentionnée dans les sujets.

On va principalement travailler sur des **arbres binaires**: l'arité de chaque nœud est majorée par 2. Même ici, il y a plusieurs définitions possibles.

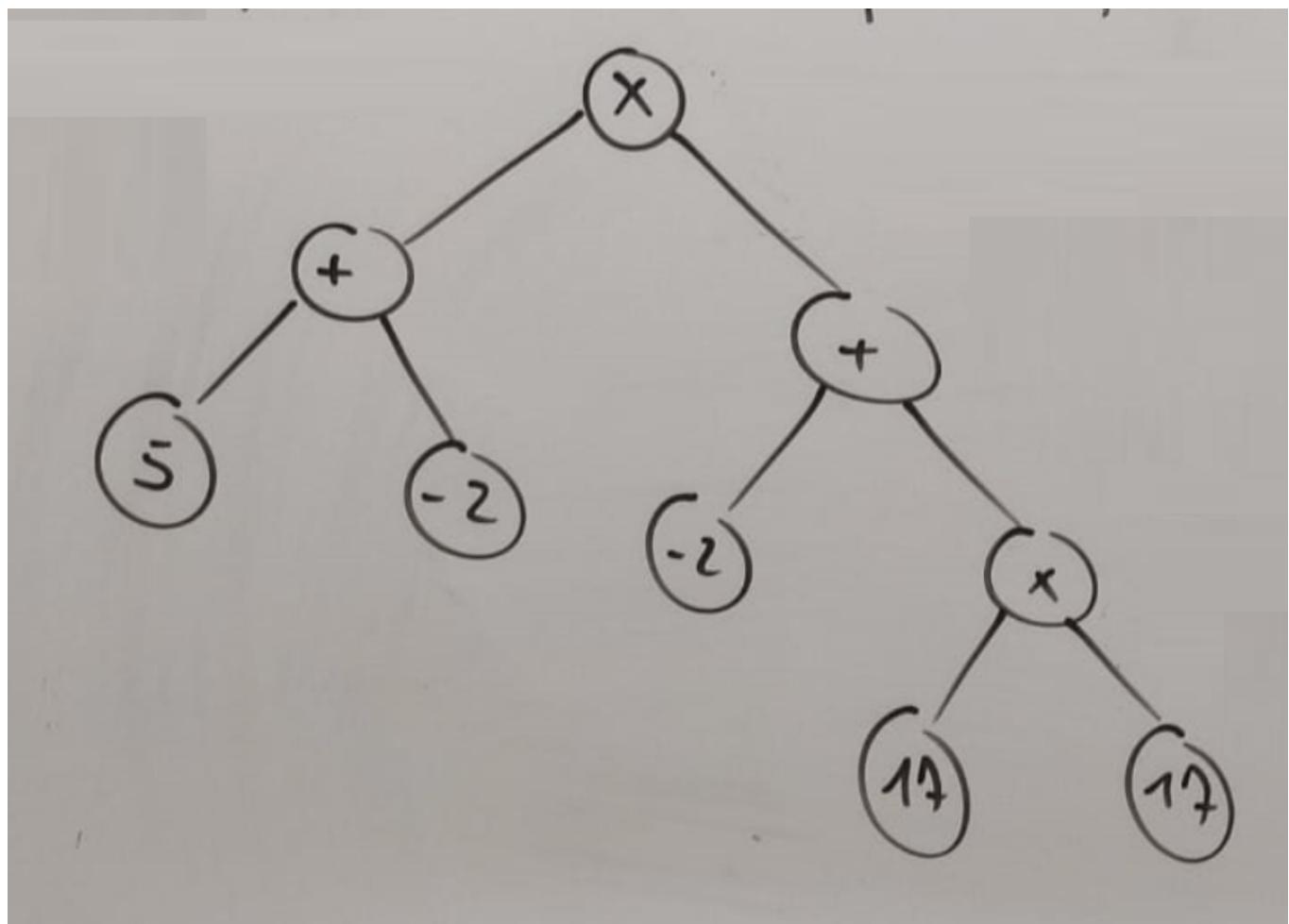
💡 **Exemple :** Expressions arithmétique avec les opérateurs + et ×.

$$(5 + (-2)) \times ((-2) + 17 \times 17)$$

On peut représenter cette expression par un arbre binaire.

Nœuds internes: opérateur

Feuilles:  $\in \mathbb{Z}$



L'arité de chaque nœud vaut 0 ou 2.

On parle d'**arbre binaire strict**.

💡 **Exemple :** le nombre de partitions d'un entier  $p$  en au plus  $q$  parties.

Autrement dit, le nombre de façons d'arranger 5 en somme de 3 termes :

$(2 + 2 + 1), (3 + 1 + 1), (0 + 0 + 5), \dots$

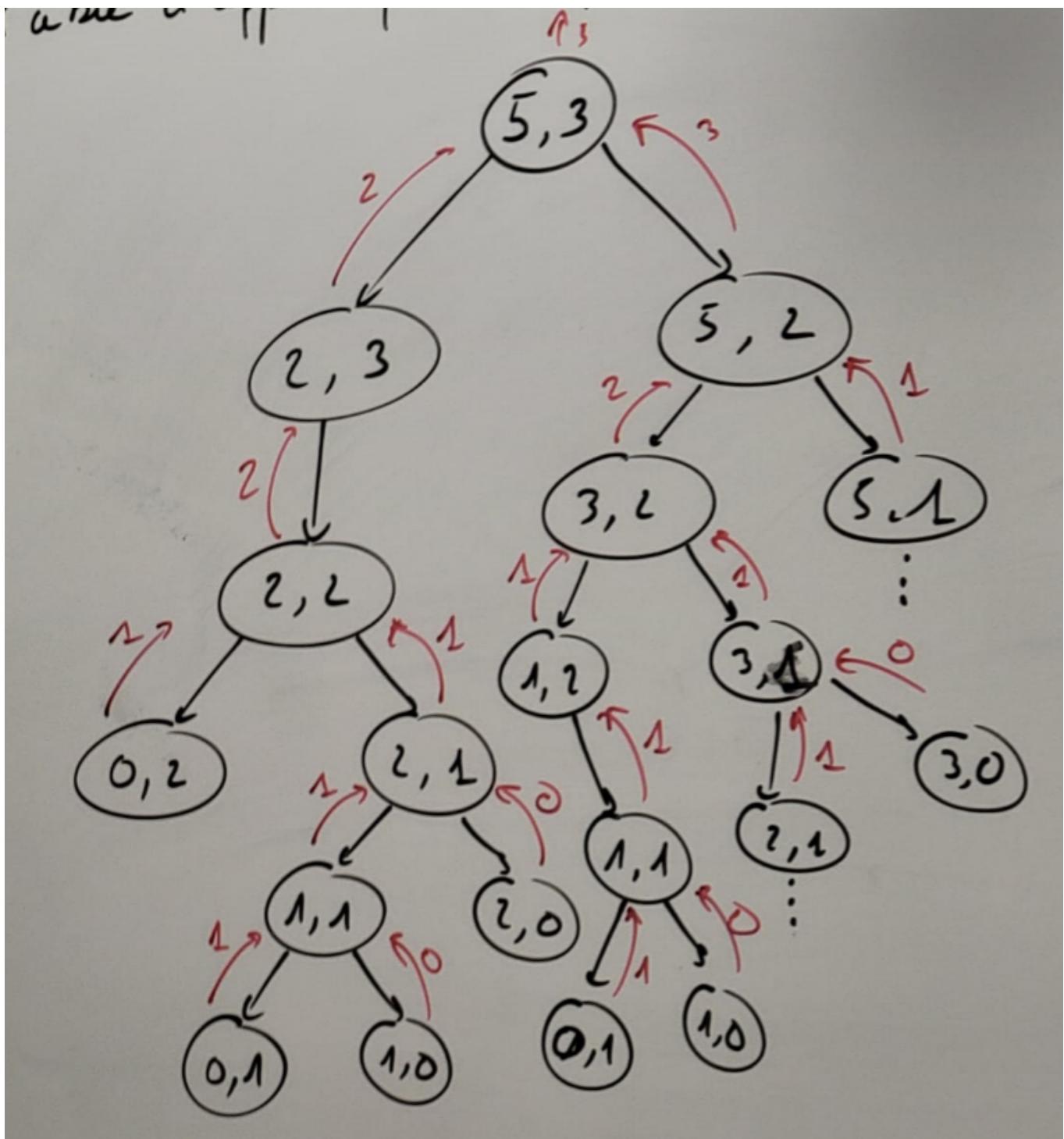
```

let rec nb_partitions p q =
  if p = 0 then 1
  else
    if q = 1 then 1
    else
      nb_partitions p (q-1)
      + nb_partitions (p-q) q
  
```

```

else if q = 0 then 0
else if q > p then nb_partitions p p
else nb_partitions (p-q) q + nb_partitions p (q-1)

```



## I. Definitions

### I.A. Arbres binaires stricts

 Définition : On considère :

- un ensemble  $\mathcal{F}$ , l'ensemble des étiquettes des feuilles, et

- un ensemble  $\mathcal{N}$ , l'ensemble des étiquettes des nœuds internes.  
( $\mathcal{F}$  et  $\mathcal{N}$  représentent l'ensemble des valeurs que peuvent prendre respectivement les feuilles et les nœuds internes.)

On définit l'ensemble  $\mathcal{A}(\mathcal{N}, \mathcal{F})$  des arbres binaires stricts comme le plus petit ensemble qui vérifie :

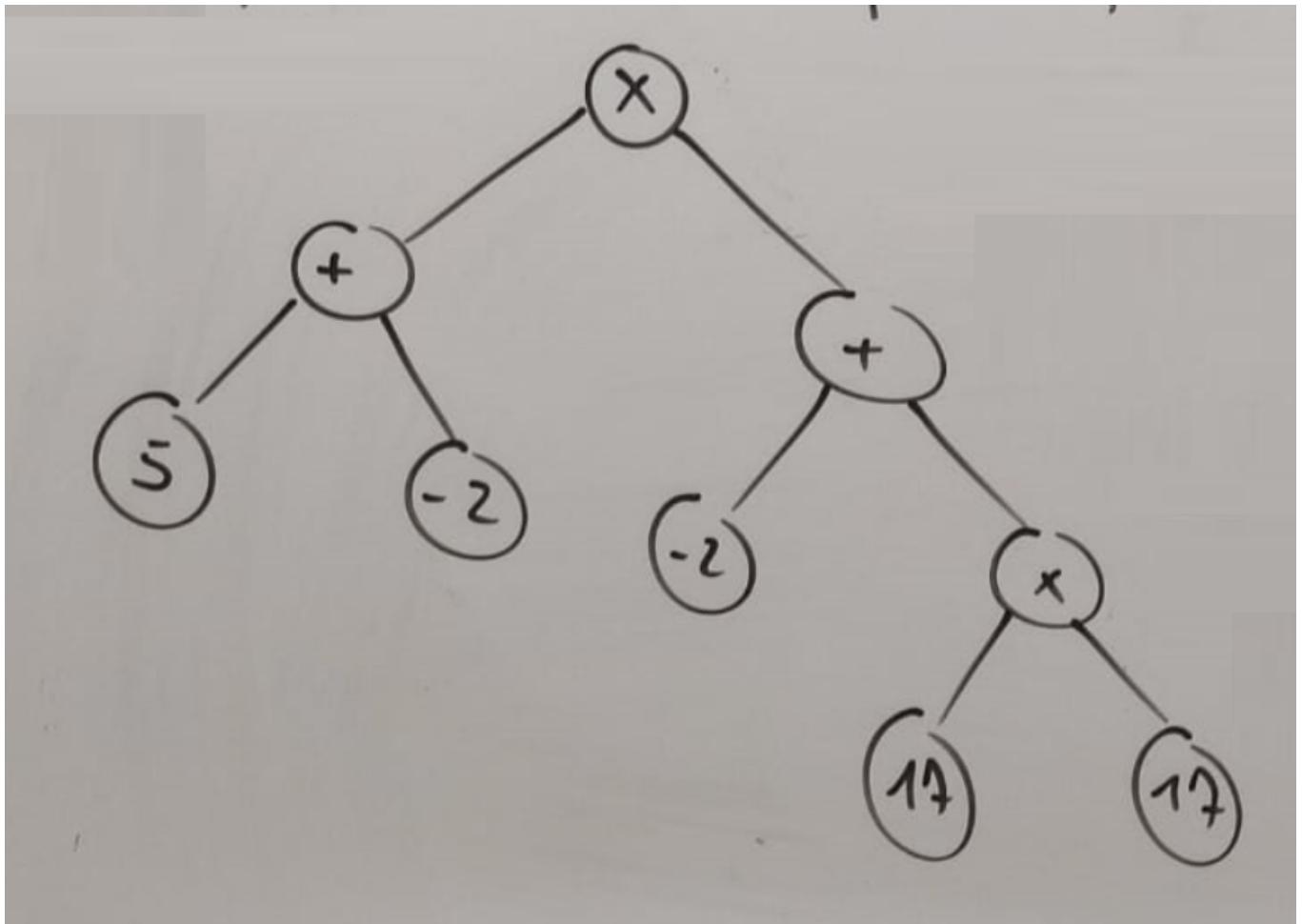
- les feuilles sont des arbres :  $\mathcal{F} \subset \mathcal{A}(\mathcal{N}, \mathcal{F})$
- si  $g$  et  $d$  sont deux arbres et si  $x \in \mathcal{N}$ , alors Nœud  $(x, g, d) \in \mathcal{A}(\mathcal{N}, \mathcal{F})$

Autrement dit, c'est l'ensemble des façons d'arranger les feuilles et les nœuds internes d'un arbre tel que les feuilles restent des feuilles et les nœuds internes restent des nœuds internes.

On a une correspondance immédiate avec un type OCaml.

```
type ('n, 'f) arbre_bin =
| Feuille of 'f
| Nœud of 'n * ('n, 'f) arbre_bin * ('n, 'f) arbre_bin
```

 Exemple avec l'arbre précédent :



```
let expr =
  Nœud (
    Fois,
```

```

Noeud (
  Plus (* + *),
  Feuille 5,
  Feuille (-2)
),
Noeud (
  Plus (* + *),
  Feuille (-2),
  Noeud (
    Fois (* × *),
    Feuille 17,
    Feuille 17
  )
)
)

```

## Induction structurelle

La définition donnée est un cas particulier d'ensemble inductif. On reviendra sur ces notions, mais pour l'instant, on peut faire :

### Preuve par induction structurelle :

Si  $\mathcal{P}$  est un prédicat sur les arbres tel que :

- pour tout  $f \in \mathcal{F}$ ,  $\mathcal{P}(f)$
- pour tout  $x \in \mathcal{N}$  et  $g, d \in \mathcal{A}(\mathcal{N}, \mathcal{F})$ ,  
 $(\mathcal{P}(g) \text{ et } \mathcal{P}(d)) \implies \mathcal{P}(\text{Noeud}(x, g, d))$

alors  $\mathcal{P}$  est vérifié pour tout arbre de  $\mathcal{A}(\mathcal{N}, \mathcal{F})$

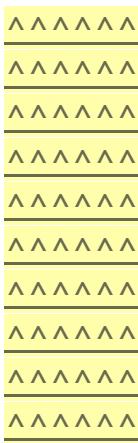
Dit autrement :

Pour un prédicat  $\mathcal{P}$  sur les arbres, si :

- $\mathcal{P}$  est vrai pour toutes les feuilles, et
- $\mathcal{P}$  est vrai pour tous les nœuds internes  $g$  et  $d$  possibles **COMPLETER**

AAAAAAAAAAAAA  
AAAAAAAAAAA

^ ^ ^ ^ ^  
^ ^ ^ ^ ^



### Définition par induction :

On définit une fonction  $\varphi$  sur  $\mathcal{A}(\mathcal{N}, \mathcal{F})$  en :

- donnant ses valeurs sur les feuilles
- définissant  $\varphi(\text{Noeud}(x, g, d))$  en fonction de  $x, \varphi(g), \varphi(d)$

### Définition :

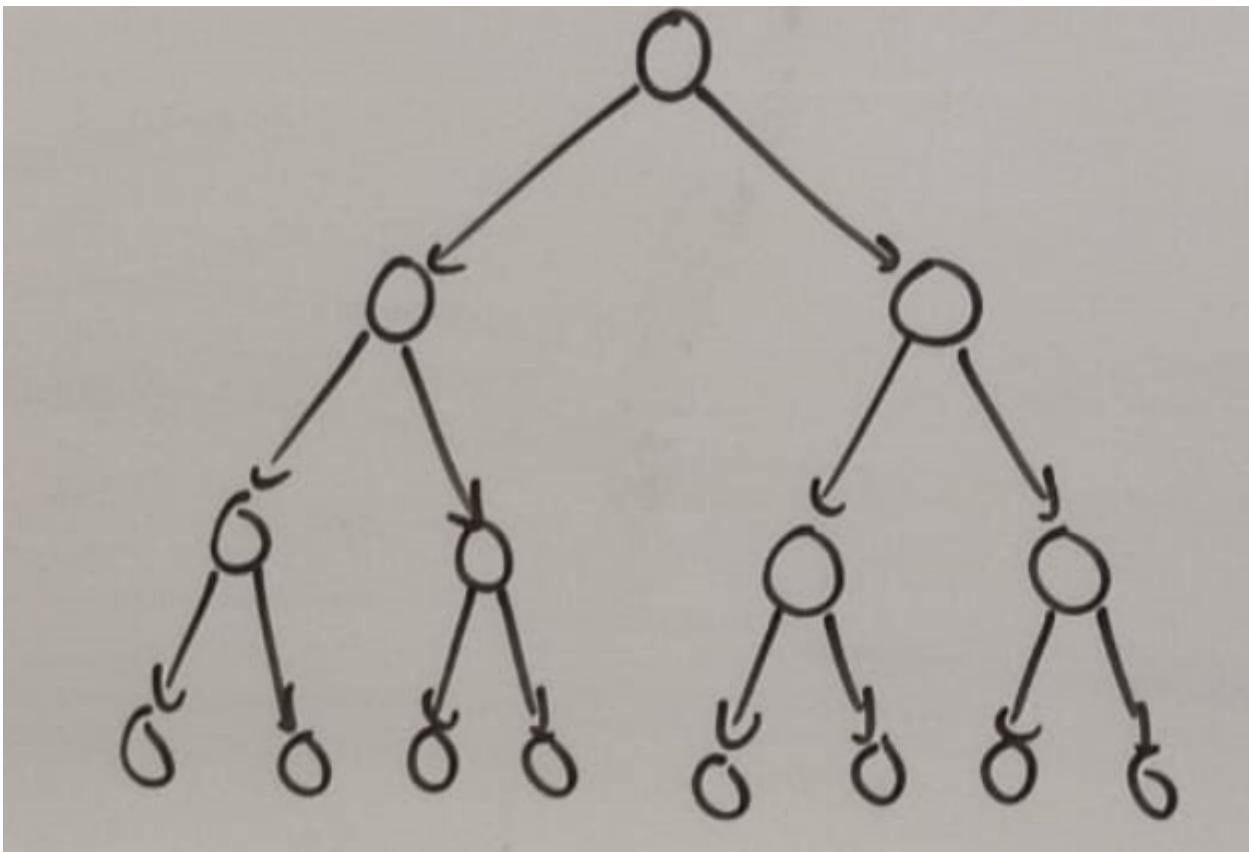
La hauteur d'un arbre binaire strict est définie par :

- $h(f) = 0$  si  $f$  est une feuille.
- $h(\text{Noeud}(x, g, d)) = 1 + \max(h(g), h(d))$

### Définition :

Un arbre binaire parfait est :

- soit une feuille,
- soit de la forme  $\text{Noeud}(x, g, d)$  où  $g$  et  $d$  sont 2 arbres binaires parfaits de *même hauteur*.



💡 **Proposition :** Un arbre binaire strict est parfait si et seulement si toutes ses feuilles sont à la même profondeur.

🛠️ **Démonstration :** par induction structurelle:

- Si notre arbre  $T$  est une feuille, c'est évident.
- si  $T = \text{Nœud}(x, g, d)$ , on suppose que  $g$  et  $d$  vérifient la propriété voulue.  
 $(\Rightarrow)$  Si  $T$  est parfait, alors  $g$  et  $d$  sont parfaits de même hauteur par définition, donc par hypothèse, toutes leurs feuilles sont à la même profondeur. (identique entre  $g$  et  $d$  car hauteur égales). Donc toutes les feuilles de  $T$  sont à la même profondeur.  
 $(\Leftarrow)$  Si toutes les feuilles de  $T$  sont à la même profondeur, alors en restreignant à  $g$  ou à  $d$ 
  - toutes les feuilles de  $g$  sont à la même profondeur  $\Rightarrow g$  est parfait
  - toutes les feuilles de  $d$  sont à la même profondeur  $\Rightarrow d$  est parfait

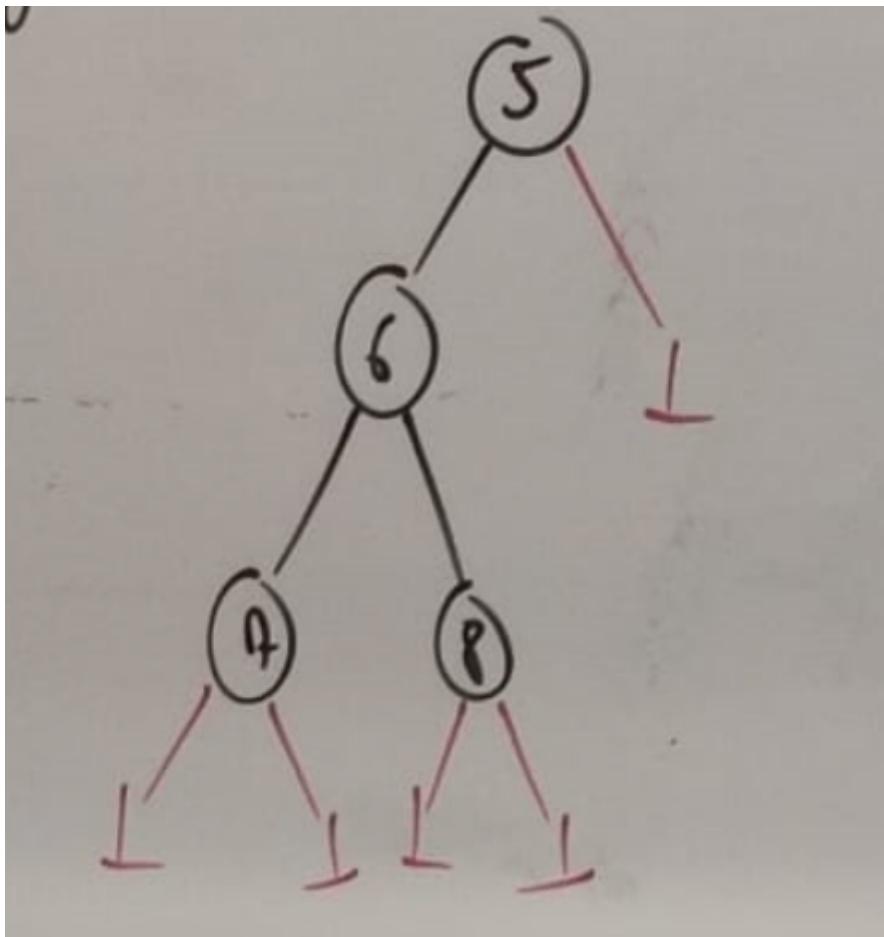
et ils ont la même hauteur. Donc  $T$  est parfait.

On peut conclure l'induction structurelle.

## I.B. Arbres binaires non stricts

📋 **Définition :** On considère un ensemble  $\mathcal{E}$ , l'ensemble des étiquettes des noeuds. L'ensemble  $\mathcal{A}(\mathcal{E})$  des arbres binaires non stricts étiquetés par  $\mathcal{E}$  est défini inductivement par :

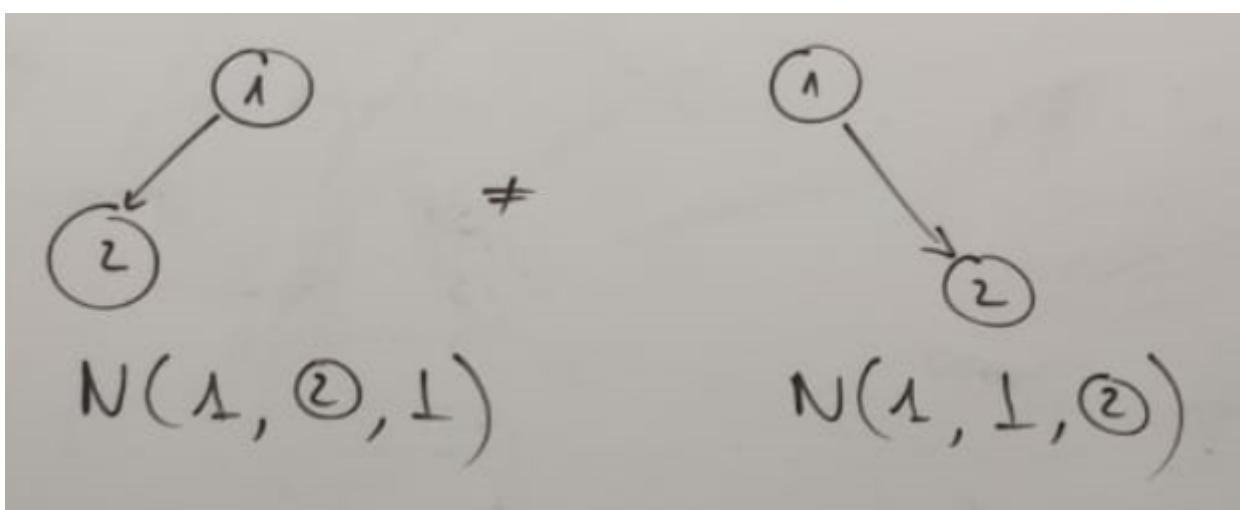
- l'arbre vide  $\perp$  appartient à  $\mathcal{A}(\mathcal{E})$
- si  $g$  et  $d$  appartiennent à  $\mathcal{A}(\mathcal{E})$  et  $x \in \mathcal{E}$ , alors l'arbre  $N(x, g, d) \in \mathcal{A}(\mathcal{E})$



- un feuille (donc d'arité nulle) possède dans cette définition 2 fils vides.  $N(x, \perp, \perp)$
- un nœud d'arité 1 va posséder en plus 1 fils vide.

**Définition :** l'arité d'un nœud est son nombre d'enfants non vides.

Si un nœud n'a qu'un seul fils, il sera placé à gauche ou à droite suivant la manière dont on le construit:

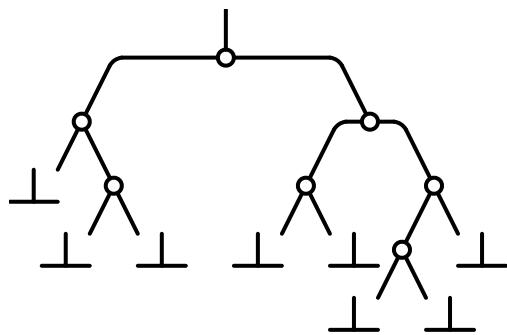


Le type OCaml pour modéliser de tels arbres :

```
type 'a arbre_binaire_non_strict =
| V
| N of 'a & 'a abns * 'a abns
```

👉 Remarque : on peut voir ces arbres non stricts comme cas particuliers des arbres stricts : en dessinant les sous-arbres vides.

Mais certaines définitions comme la hauteur ne collent plus tellement : on donne alors comme hauteur pour l'arbre vide ( $-1$ ).



## I.C. Parcours d'arbres binaires

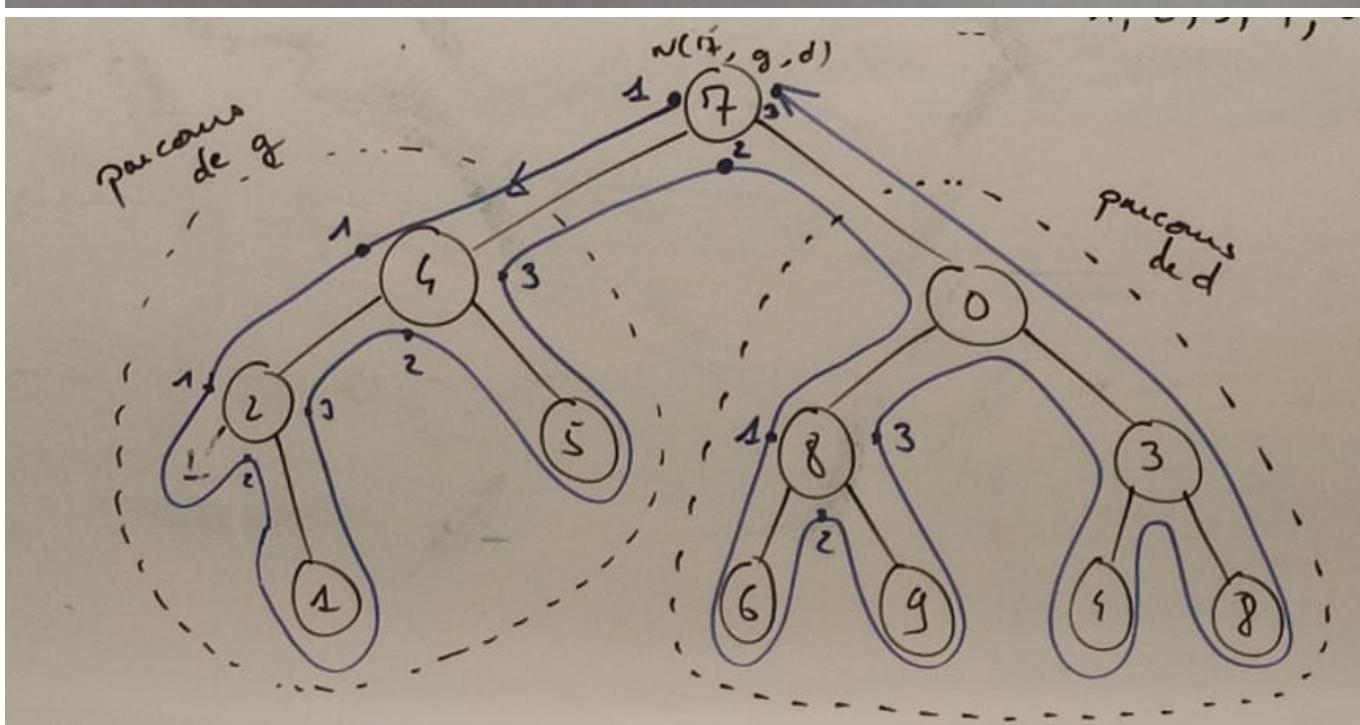
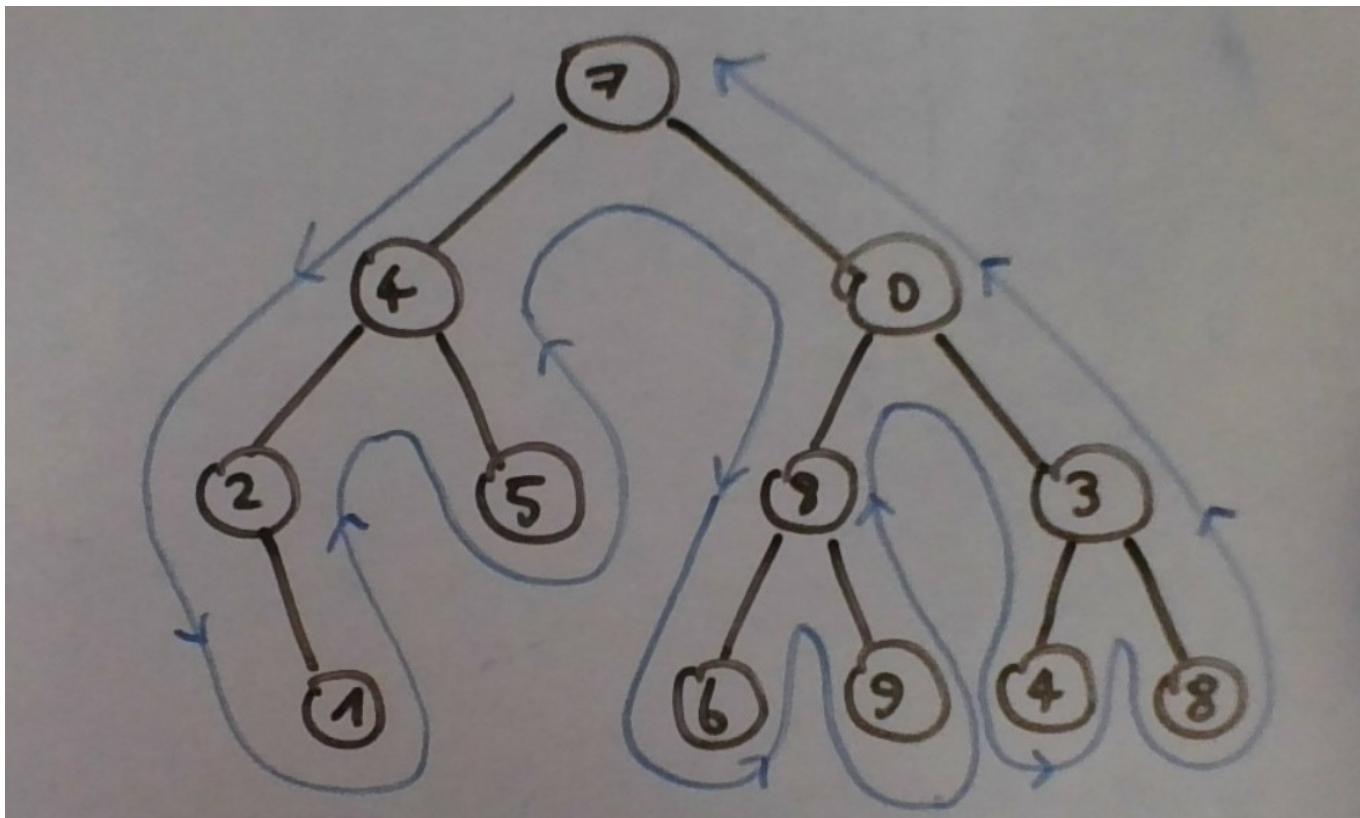
Parcourir un arbre, c'est décrire l'ensemble de nœuds dans un "certain" ordre. Un parcours peut avoir comme but de synthétiser une information à partir de toutes les étiquettes ou de la structure de l'arbre, de rechercher un nœud particulier ou encore de modifier l'arbre...

D'une certaine manière, un parcours commence toujours à la racine. (*D'un point de vue informatique, il n'y a pas de différence entre un arbre et sa racine.*) Mais cela ne signifie pas pour autant que l'on souhaite effectuer la première opération sur l'étiquette de la racine.

Tous les parcours que nous allons décrire se font de gauche à droite: on commence par le sous-arbre gauche avant le sous-arbre droit.

### I.C.1. Parcours en profondeur

On commence par descendre le plus profondément possible (jusqu'à une feuille) avant de remonter pour parcourir le reste de l'arbre.



**💡 Deux remarques importantes :**

- C'est un parcours naturellement récursif : on traite la racine et à partir de là, on parcourt récursivement le sous-arbre gauche, puis le sous-arbre droit.
- Chaque noeud interne est visité 3 fois:
  - avant d'explorer sont fils gauche,
  - entre l'exploration du fils gauche et celle du fils droit,
  - après l'exploration du fils droit.

Cette 2<sup>ème</sup> remarque induit 3 ordres différents sur les noeuds.

- **Ordre préfixe** : Le traitement d'un nœud se fait à la *première* visite.

1. Racine,
2. Sous-arbre gauche,
3. Sous-arbre droit.

Sur l'exemple : 7 4 2 1 5 0 8 6 9 3 4 8

C'est à chaque fois qu'on touche à gauche.

- **Ordre infixé** : Le traitement d'un nœud se fait à la *deuxième* visite.

1. Sous-arbre gauche,
2. Racine,
3. Sous-arbre droit.

Sur l'exemple : 2 1 4 5 7 6 8 9 0 4 3 8

C'est à chaque fois qu'on touche en bas.

- **Ordre postfixé** : Le traitement d'un nœud se fait à la *troisième* visite.

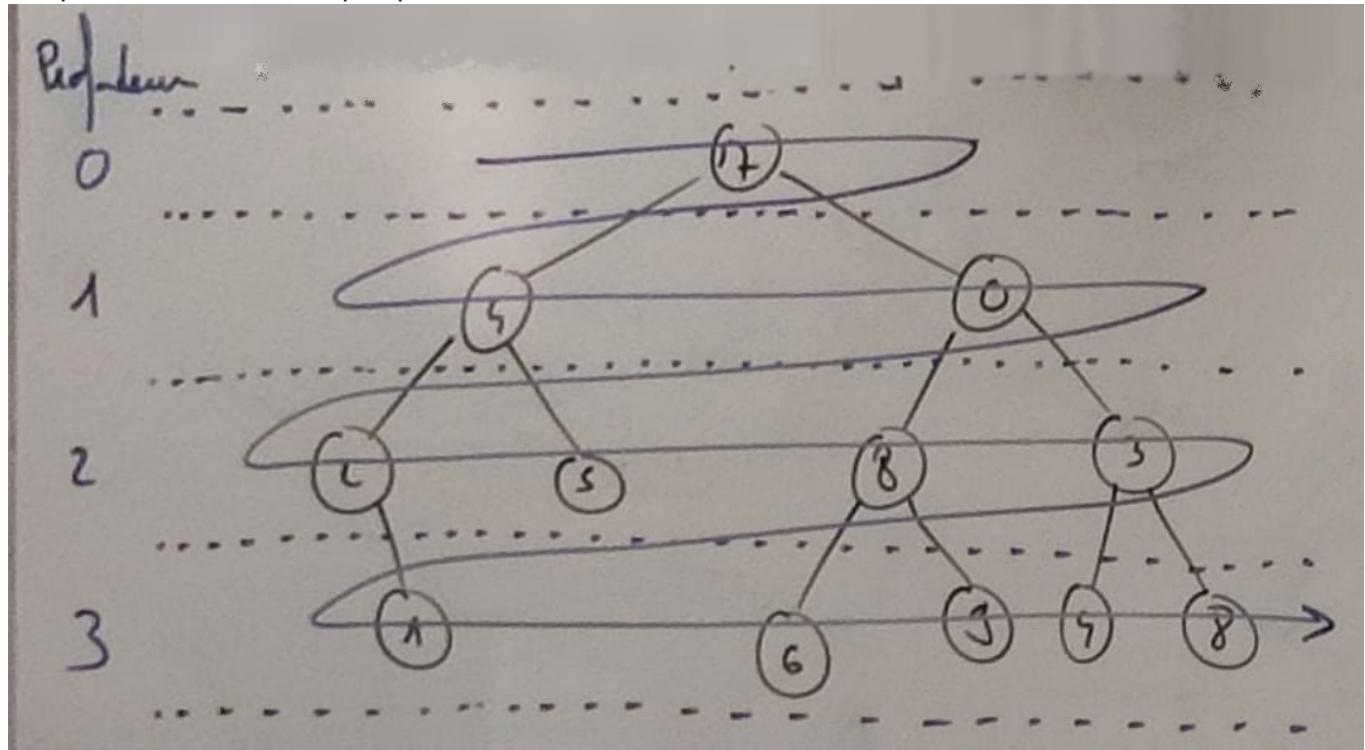
1. Sous-arbre gauche,
2. Sous-arbre droit,
3. Racine.

Sur l'exemple : 1 2 5 4 6 9 8 4 8 3 0 7

C'est à chaque fois qu'on touche à droite.

## I.C.2. Parcours en largeur

On parcourt les nœuds par profondeur croissante.



L'arbre induit sur les étiquettes est 7 4 0 2 5 8 3 1 6 9 4 8.

## I.C.3. En OCaml



Sur les arbres binaires stricts, on veut une fonction qui affiche les étiquettes dans l'ordre préfixe :

```
type arbre =
| Feuille of int
| Noud of int * arbre * arbre

let rec preorder t =
  match t with
  | Feuille x -> Printf.printf "%d\n" x
  | Noeud (x, g, d) -> Printf.printf "%d\n" x; preorder g; preorder d

let rec inorder t =
  match t with
  | Feuille x -> Printf.inorder "%d\n" x
  | Noeud (x, g, d) -> inorder g; Printf.printf "%d\n" x; inorder d

let rec postorder t =
  match t with
  | Feuille x -> Printf.postorder "%d\n" x
  | Noeud (x, g, d) -> postorder g; postorder d; Printf.printf "%d\n" x;
```

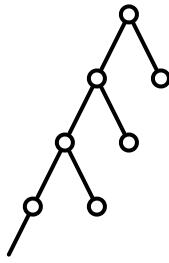
Si on veut récupérer la liste des étiquettes :

```
let prefix t =
  match t with
  | Feuille x -> [x]
  | Noeud (x, g, d) -> [x] :: (prefix g @ prefix d)
```

Pour la complexité de cette fonction, en fonction de la taille  $n$  de l'arbre:

$$\left\{ \begin{array}{l} C(1) = 1 \\ C(n) = \underbrace{1}_{::} + \underbrace{C(l)}_{\text{appel sur } g, \text{ avec } l = |g|} + \underbrace{l}_{@} + \underbrace{C(n-1-l)}_{\text{l'appel sur } b} \end{array} \right.$$

Le pire des cas se produit avec  $l$  le plus grand possible, i.e.  $l = n - 2$  (cas strict) chaque étape : on obtient un arbre en forme de peigne.



Dans ce cas (la taille de l'arbre est impaire),

$$\begin{aligned}
 C(2n+1) &= 1 + C(2n-1) + (2n-1) + C(1) \\
 &= (2n+1) + C(2n-1)
 \end{aligned}$$

$$\begin{aligned}
 \sum_{k=1}^n (C(2k+1) - C(2k-1)) &= \sum_{k=1}^n (2k+1) \\
 C(2n+1) - C(1) &= 2 \times \frac{n(n+1)}{2} + n
 \end{aligned}$$

$$C(2n+1) = (n+1)^2$$

Ainsi,  $C(n) = O(n^2)$

Pour améliorer la complexité, on passe par une fonction auxiliaire avec un accumulateur.

```

let prefix_opt t =
  let rec aux t acc =
    match t with
    | Feuille x -> x :: acc
    | Noeud (x, g, d) -> x :: (aux g (aux d acc))
  in aux [] t
  
```

Complexité :

$$\begin{cases} C(1) = 1 \\ C(n) = 1 + C(l) + C(n-1-l) \end{cases}$$

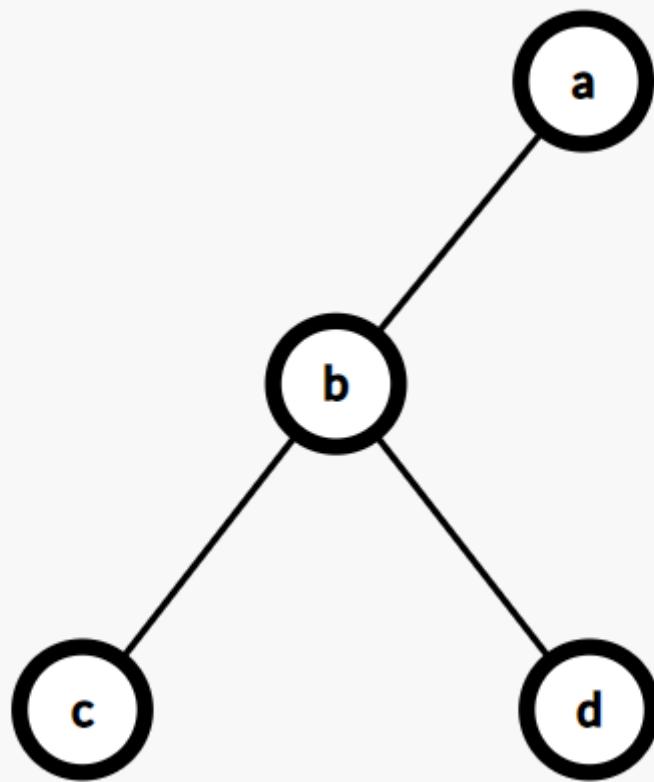
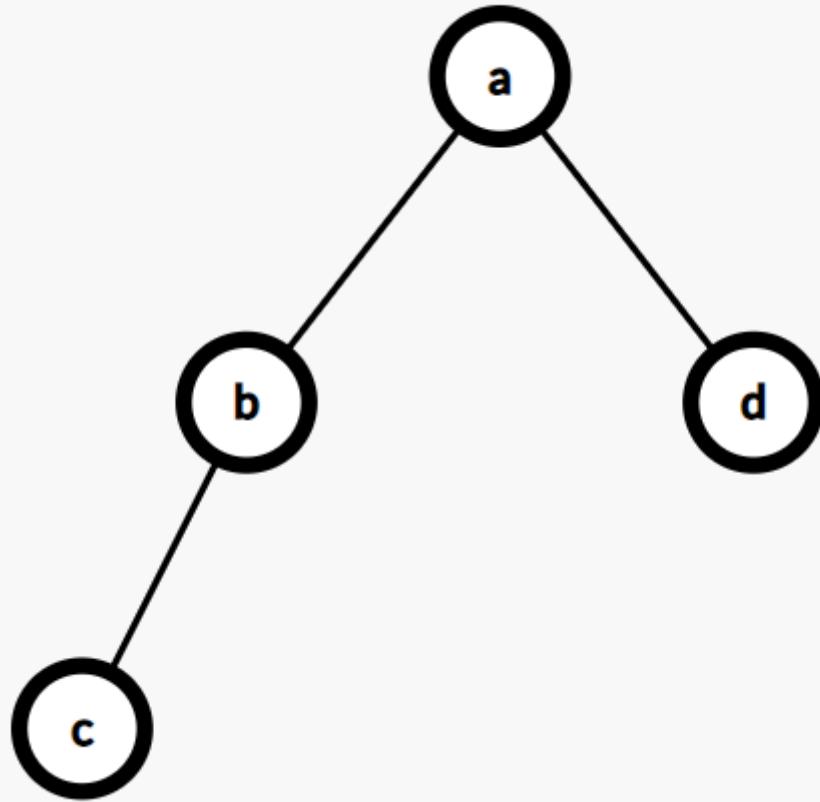
On obtient une complexité linéaire (on trouve la formule générale avec quelques exemples et on la démontre par récurrence).

#### I.C.4. Unicité

Si on dispose de l'énumération des étiquettes dans un ordre défini, est-il possible de reconstruire l'arbre correspondant ?  
(i.e. est-ce qu'il existe un arbre unique donnant cette énumération ?)

Dans le cas général, la réponse est non.

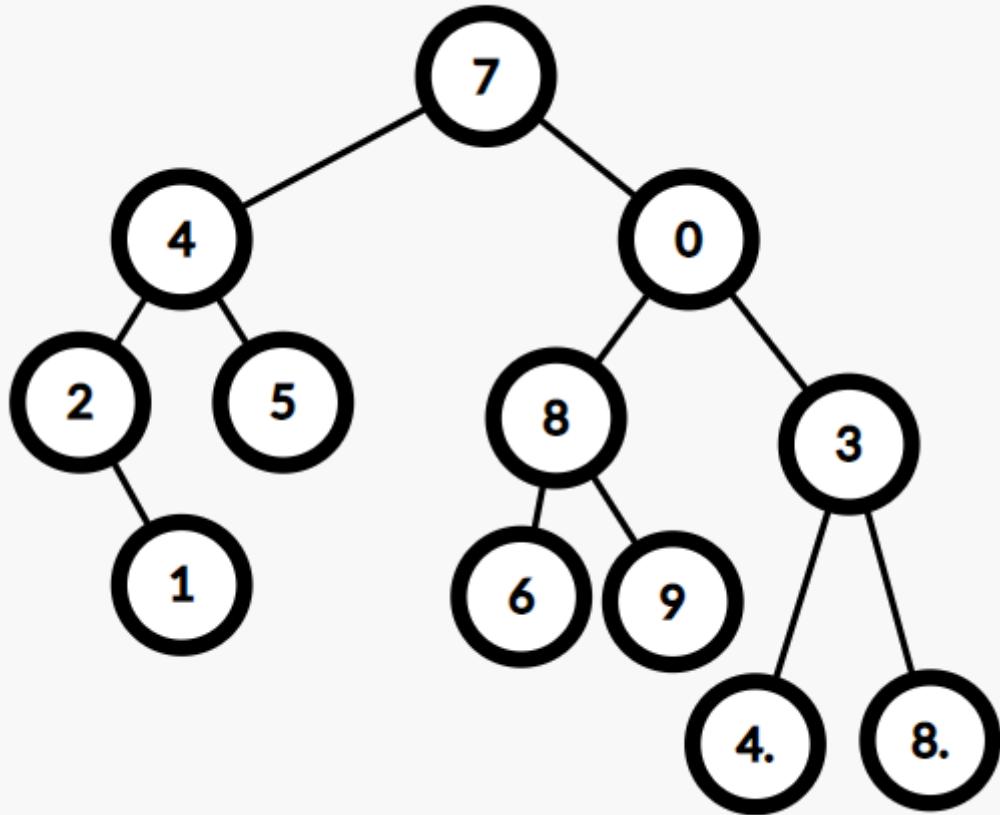
 **Exemple :** Parcours préfixe : a, b, c, d



Cependant, si l'énumération est un peu plus complète, c'est possible pour tous les ordres sauf l'ordre infixe :

- Si l'arbre est non strict, on indique les  $\perp$ .

Par exemple, pour l'arbre utilisé pour présenter les parcours :

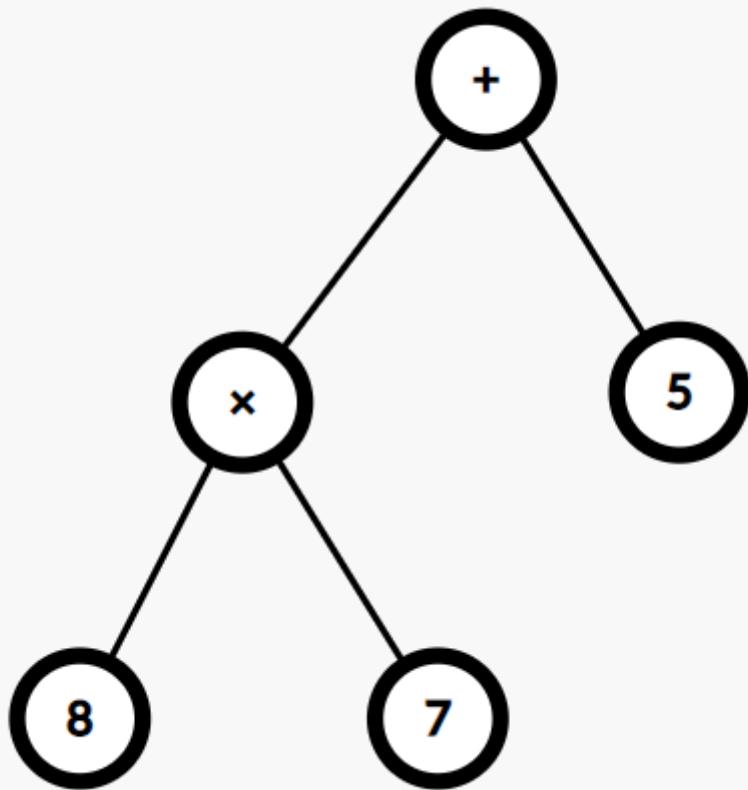


Préfixe : 7 4 2 ⊥ 1 ⊥ ⊥ 5 ⊥ ⊥ 0 8 6 ⊥ ⊥ 9 3 4 ⊥ ⊥ 8 ⊥ ⊥

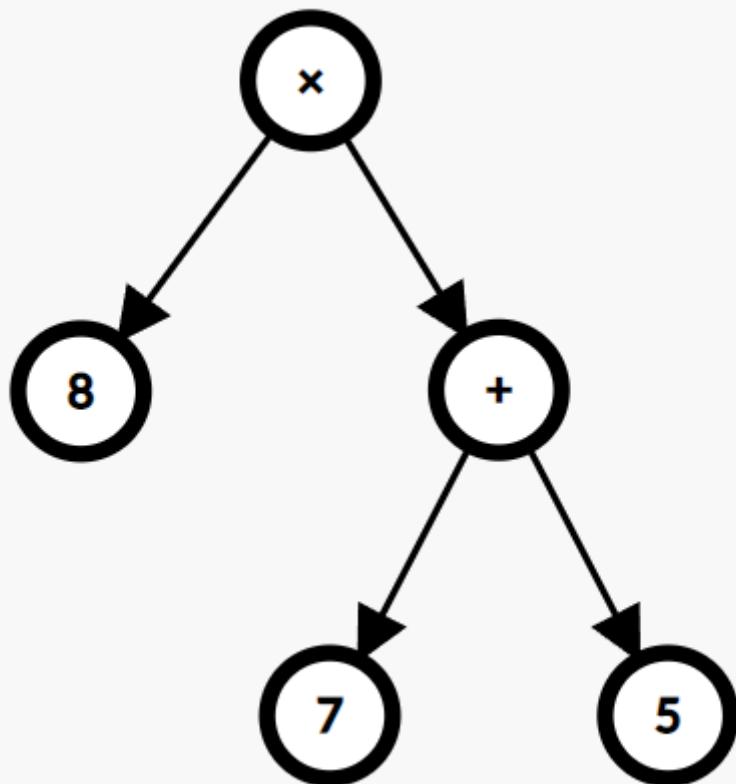
- Si l'arbre est strict, on différencie feuille et nœud interne.

 Exemple : Ordre infixe :  $8 \times 7 + 5$

Deux arbres ont ce parcours infixe :



Postfixe `8 7 × 5 +`

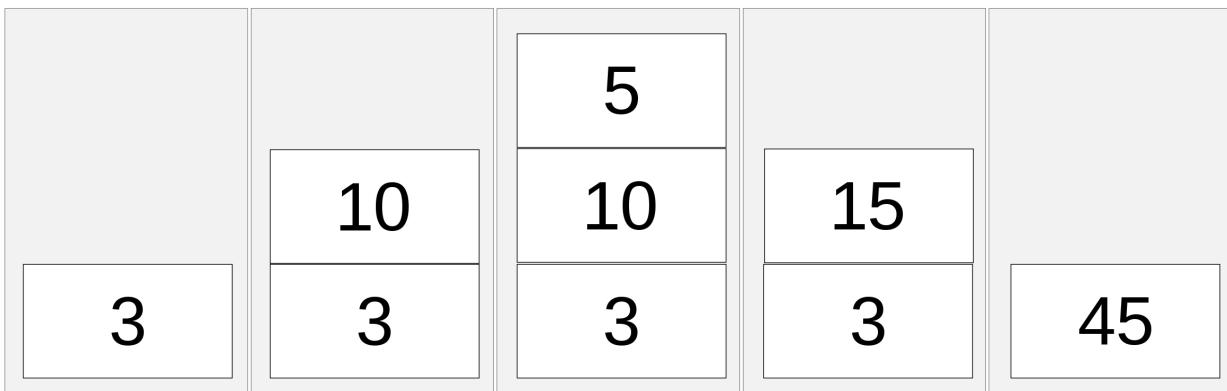


Postfixe `8 7 5 + ×`

Voir notation polonaise inversée : pas besoin de règles ou de parenthèses !  
Postfixe = polonaise inversée

Equation:    3    10    5    +    \*

Stack



Input

3                10                5                +                \*

 Exemple: Montrer qu'un arbre binaire (dont toutes les clés/étiquettes sont uniques) est uniquement déterminé par les listes de ces étiquettes données par un parcours préfixe et infixé.

Soit A un arbre.

On va prouver par induction < récurrence ? >

Cas de base:  $A = (a)$  ok.

Hérité: On a  $A = \dots$ , B et C Arbres, on sait caractériser C et C via préfixe et infixé.

Ainsi, infixe ( $A$ ) =  $[a_0, a_1, \dots, a_n]$

préfixe ( $A$ ) =  $[\underbrace{a_i, \dots, a_n}_{\text{racine de } A}, \dots, a_n]$  avec  $i \in \llbracket 0, n \rrbracket$

Ainsi, infixe ( $A$ ) =  $[a_0, a_1, \dots, ]$

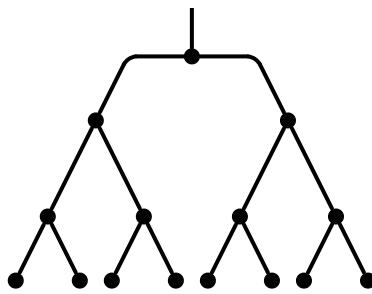
Euh j'ai pas compris ce qu'a fait Raphael H, je crois que la vraie correction est sur le site du prof.

## I.D. Un peu de dénombrement

### I.D.1 Relation entre hauteur, nombre de nœuds et nombre de feuilles

 Proposition: En notant  $n$  le nombre de nœuds d'un arbre binaire strict et  $f$  son nombre de feuilles, on a :

$$f = n + 1$$



### Démonstration :

Par induction structurelle :

→ Si l'arbre est une feuille, alors  $n = 0$  et  $f = 1 = 0 + 1 \rightarrow \text{ok.}$

→ Sinon, l'arbre est de la forme  $A = (x, A_g, A_d)$ . On a alors :

$$\begin{cases} f_g = n_{g+1} \\ f_d = n_d + 1 \end{cases} \text{ par hypothèse d'induction}$$

et alors,  $\begin{cases} f = f_g + f_d \\ n = n_g + n_d + 1 \end{cases}$

i.e.,

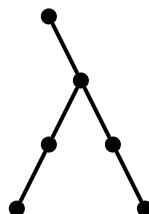
$$f = (n_g + 1) + (n_d + 1)$$

$$f = (n_g + n_d + 1) + 1$$

$$f = n + 1$$

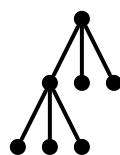
ok.

Remarque : C'est faux en général pour un arbre binaire non strict.



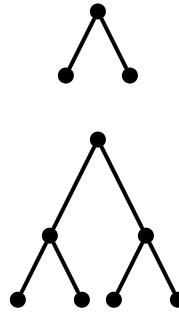
Exercice : On appelle arbre  $k$ -aire strict tout arbre dont tous les nœuds internes ont exactement  $k$  fils.

Trouver une relation entre  $n$  et  $f$  pour un arbre  $k$ -aire.



$$f = (k - 1)n + 1$$

 **Proposition :** Un arbre binaire (strict) de hauteur  $h$  a au plus  $2^{k+1} - 1$  nœuds (et  $2^k - 1$  nœuds internes)



 **Remarque :** De nombreux algorithmes sur des arbres (de taille  $n$ ) ont une complexité proportionnelle à la hauteur de l'arbre.

Si l'arbre est bien équilibré, cette complexité vaut essentiellement  $\log_2(n)$ .

 **Exercice :** On appelle arbre binaire complet de hauteur  $h$  un arbre binaire strict dont toutes les feuilles sont à profondeur  $h$  ou  $h - 1$ .

Donner pour un tel arbre :

1. Un encadrement du nombre total de nœuds en fonction de la hauteur.

- o L'arbre parfait de hauteur  $h$  est complet, donc  $n \leq 2^{k+1} - 1$  (et on ne peut pas faire mieux comme majoration)
- o Un arbre complet de hauteur  $h$  doit contenir au moins une feuille de profondeur  $h$  (donc au moins 2).

Au minimum, c'est un arbre parfait de hauteur  $(h - 1)$  sur lequel on rajoute 2 feuilles

$$n \geq 2^k + 1$$

$$2^k \leq \boxed{2^k + 1 \leq n \leq 2^{k+1} - 1}$$

$$2^k \leq n < 2^{k+1}$$

$$2^k \leq n \leq k + 1$$

$$\boxed{h = \lfloor \log_2 n \rfloor}$$

2. Un encadrement de la hauteur en fonction du nombre total de nœuds.

- o Exo #?!④ : On considère un arbre binaire vérifiant la propriété suivante :

Pour tout nœud interne, la hauteur de ses 2 sous-arbres différents au plus de 1.

Montrer que :

$$h \leq \frac{3}{2} \log_2(n + 1)$$

## I.D.2. Nombres d'arbres binaires de taille fixée

 **Définition :** Les nombres de Catalan sont définis par :

- $c_0 = 1$
- $\forall n \in \mathbb{N}, c_{n+1} = \sum_{k=0}^n c_k c_{n-k}$

$c_n$  arbres.

 **Démonstration :** On procède par récurrence sur  $n$ .

- $n = 0$  : Il existe un unique arbre sans nœud interne : une feuille  $c_0 = 1$
- Un arbre binaire à  $n + 1$  nœuds internes est constitué de :
  - une arbre,
  - un sous-arbre ayant  $k$  nœuds internes (avec  $k \in \llbracket 0, n \rrbracket$ ), choisi librement : par HR, il y a  $c_k$  possibilités.
  - un sous-arbre droit ayant  $k' = n - k$  nœuds internes :  $c_{n-k}$  possibilités.

Donc  $c_{n+1} = \sum_{k=0}^n c_k c_{n-k}$ . On peut donc conclure la récurrence.

 **Théorème :** Pour tout  $n \in \mathbb{N}$ , on a :

- $(n + 2)c_{n+1} = 2(2n + 1)c_n$
  - $c_n = \frac{1}{n + 1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n + 1}$
- $$\binom{2n}{n + 1} = \frac{(2n)!}{(n + 1)!(n - 1)!}$$

*bah il a effacé la suite avant que je copie*

compléter

Par récurrence sur  $n$  :

$$n = 0 : \frac{1}{0+1} \binom{0}{0} = 1 = c_0$$

Si c'est au rang  $n$  :

$$c_{n+1} \stackrel{1^{\text{e}} \text{ formule}}{=} \frac{2(2n+1)}{n+2} \times \frac{1}{n+1} \binom{2n}{n}$$

$$= \frac{2(2n+1)(2n)!}{(n+2)(n+1)(n!)^2}$$

$$= \frac{(2n+2)(2n+1)(2n)!}{(n+2)(n+1)^2(n!)^2}$$

$$= \frac{1}{n+2} \times \frac{(2n+2)!}{(n+1)!^2}$$

Donc la proposition est vraie au rang  $(n+1)$  et on peut conclure la récurrence.

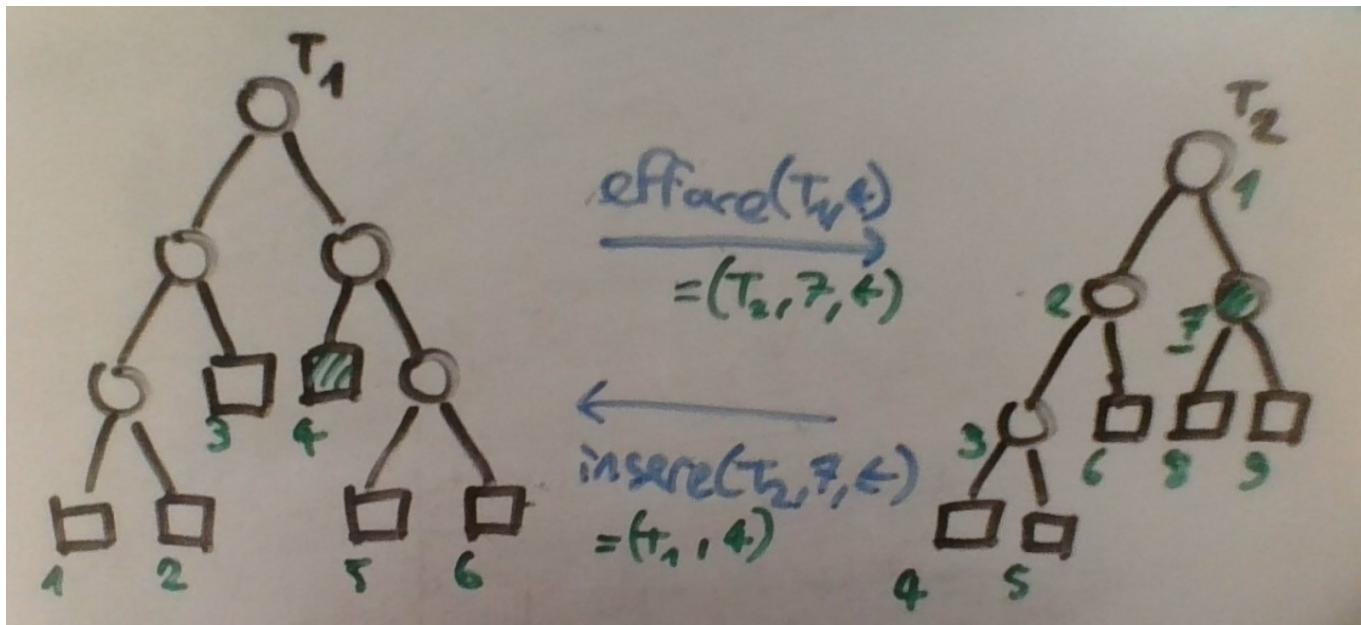
$$\underbrace{(n+2)}_{\substack{\text{Nb de feuilles} \\ \text{d'un tel arbre} \rightarrow}} \underbrace{c_{n+1}}_{\substack{\text{nombre d'arbres} \\ \text{avec } (n+1) \\ \text{nœuds internes}}} = \underbrace{2}_{\substack{\text{Un choix supplé-} \\ \text{mentaire } \varepsilon \in \{\leftarrow, \rightarrow\}}} \underbrace{(2n+1)}_{\substack{\text{Nb total de noeuds} \\ \text{d'un tel arbre} \rightarrow}} \underbrace{c_n}_{\substack{\text{Nb d'arbres à} \\ \text{n nœuds internes}}}$$

On cherche une bijection entre :

- l'ensemble des couples  $(A, f)$  où  $A$  est un arbre à  $(n+1)$  nœuds internes,  $x$  est un nœud interne et  $f$  est une feuille de  $A$
- l'ensemble des triplets  $(B, x, \varepsilon)$  où  $B$  est un arbre à  $n$  nœuds internes,  $x$  est un nœud quelconque de  $B$  et  $\varepsilon$  est choisi dans l'ensemble  $\{\leftarrow, \rightarrow\}$
- application **efface** qui à un couple  $(A, f)$  associe le triplet  $(B, x, \varepsilon)$ 
  - $(B)$  est obtenu en supprimant la feuille  $f$  ainsi que son père. La frère de  $f$  est remonté d'un cran.
  - $x$  est le nœud remonté
  - $\varepsilon$  vaut  $(\leftarrow)$  si  $f$  est un fils gauche et  $(\rightarrow)$  pour un fils droit.
- Application **insère** qui à un triplet  $(B, x, \varepsilon)$  associe le couple  $(A, f)$ 
  - $A$  est obtenue à partir de  $B$  en créant un nouveau nœud à la place de  $x$ .
    - Si  $\varepsilon = (\leftarrow)$ , le fils gauche de ce nouveau nœud est une feuille, et le fils droit est  $x$ .
    - Si  $\varepsilon = (\rightarrow)$ , on inverse.
  - $f$  est la feuille nouvellement créée.

**efface** et **insère** sont bijections réciproques l'une de l'autre, d'où l'égalité recherchée.

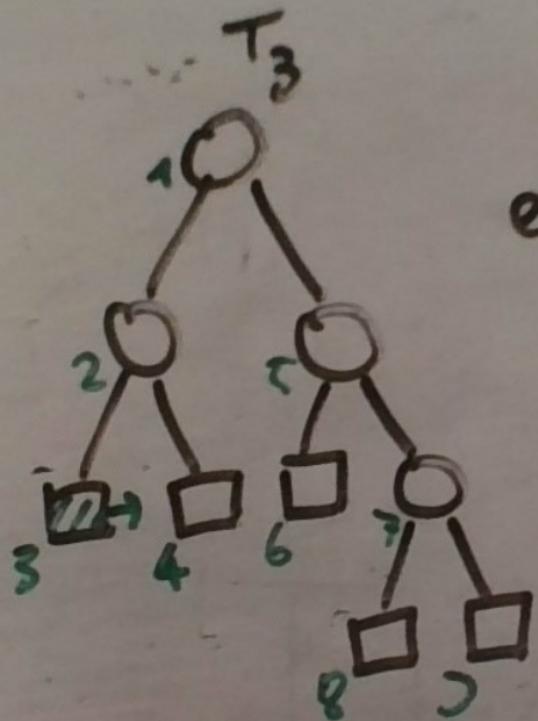
 **Exemple** : pour numérotter feuille et nœud, on choisir l'ordre préfixe.



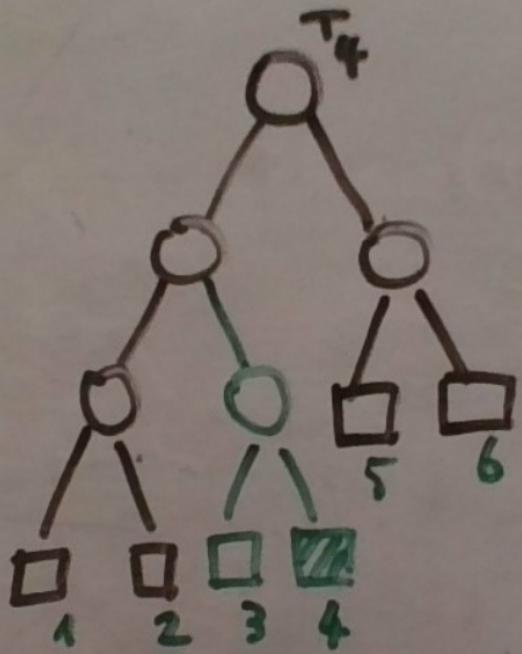
### Exercice :

Calculer :

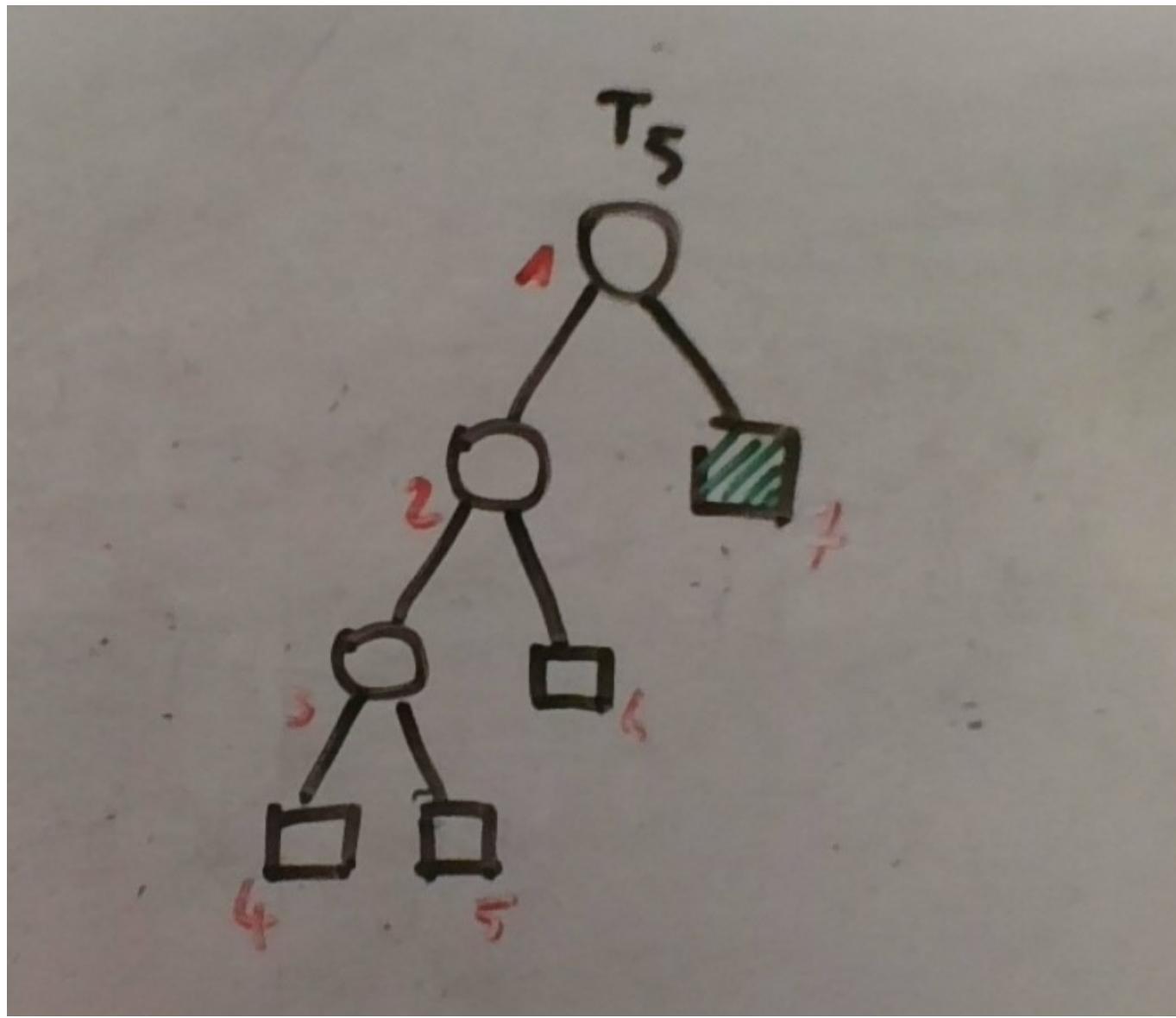
- $\text{efface}(T_1, 2)$
- $\text{insère}(T_2, 6, \rightarrow)$
- $\text{efface}(T_2, 5)$
- $\text{insère}(T_1, 1, \leftarrow)$



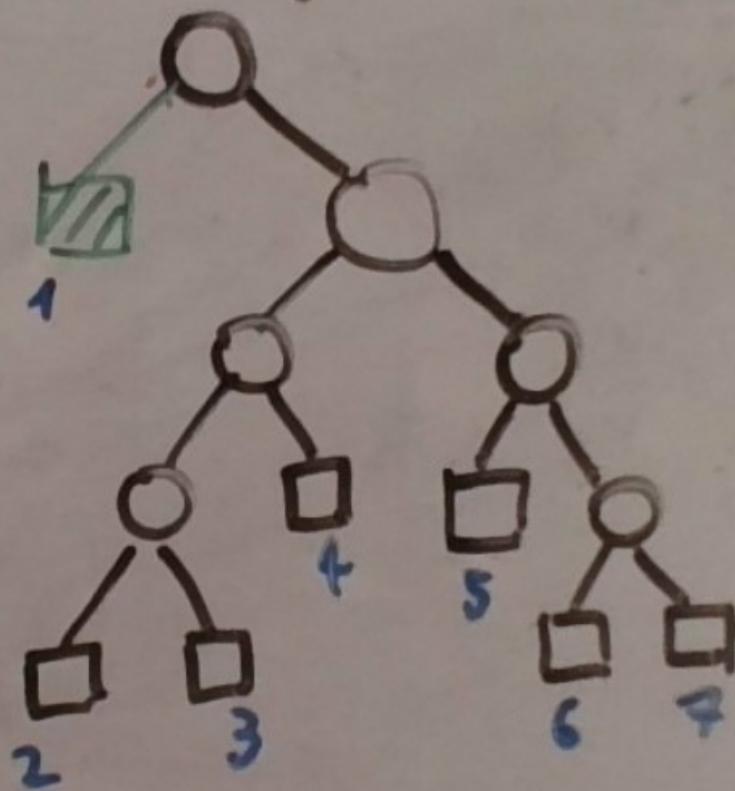
$\text{efface}(T_1, 2) = (T_3, 3, \rightarrow)$



$\text{insère}(T_2, 6, \rightarrow) = (T_4, 4)$



$$\text{efface}(\tau_2, 5) = (\tau_3, \tau, \rightarrow)$$



$$\text{insere}(\tau_1, 1, \leftarrow) = (\tau_6, \leftarrow)$$

Exercice: formule de Sterling :  $n! \sim \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$

Donner un équivalent simple de  $c_n$ .

$$\begin{aligned} c_n &= \frac{1}{n+1} \binom{2n}{n} \\ &= \frac{1}{n+1} \times \frac{(2n)!}{n!^2} \end{aligned}$$

$$\begin{aligned} &\sim \frac{\sqrt{4\pi n} \left(\frac{2n}{e}\right)^{2n}}{(n+2)2\pi n \left(\frac{n}{e}\right)^{2n}} \\ &\sim \frac{4^n}{(n+1)\sqrt{\pi n}} \end{aligned}$$

$$\sim \frac{4^n}{\sqrt{\pi} n^{\frac{3}{2}}}$$

## II. Arbres non binaires

Les nœuds peuvent avoir une arité quelconque : chaque nœud porte une étiquette et possède une liste d'enfants. Une feuille est simplement un nœud dont la liste d'enfants est vide.

```
type 'a arbre = N of 'a * ('a arbre list)
```

On peut parcourir un tel arbre :

- en profondeur : préfixe ou postfixe, mais pas infixe.
- en largeur.

### II.A. Lecture unique

On définit un parcours préfixe d'un tel arbre par :

$$\text{prefix}(N(x, [])) = (x, 0)$$

$$\text{prefix}(N(x, [a_1, \dots, a_k])) = (x, k), \text{prefix}(a_1), \dots, \text{prefix}(a_k)$$

« , » = concaténation

 Définition : suite équilibrée :

- on définit le poids d'une suite  $u = (x_1, k_1), \dots, (x_n, k_n)$  par :

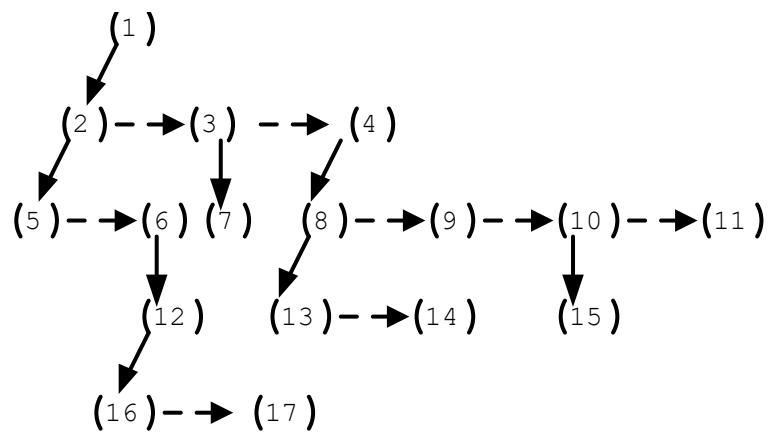
$$p(u) = \sum_{i=1}^n (k_i - 1)$$

(la suite vide a un poids nul.)

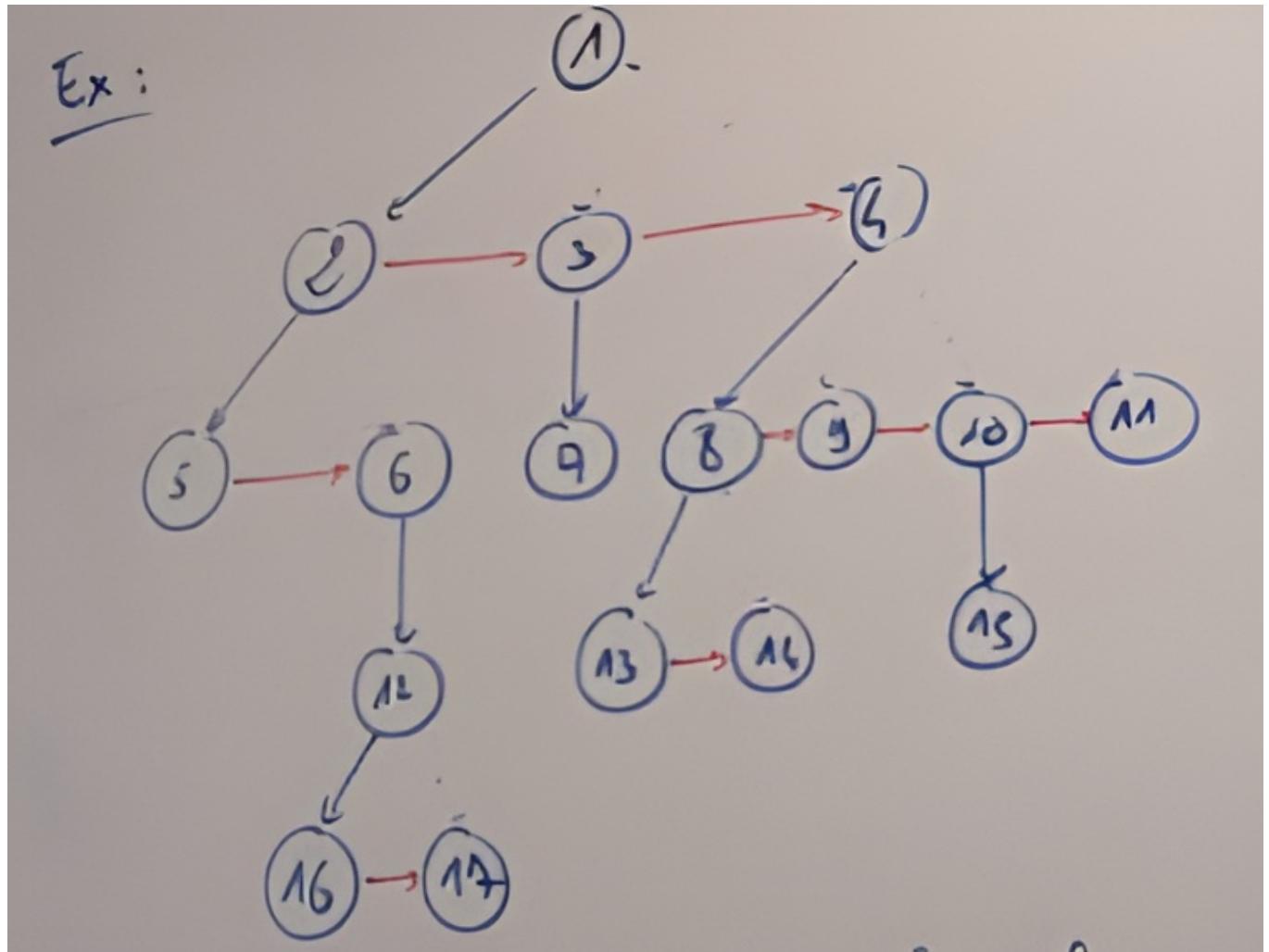
- un préfixe de  $u$  est une suite de la forme  $(x_1, k_1), \dots, (x_j, k_j)$  avec  $j \in \llbracket 1, n \rrbracket$ .
- le préfixe est dit strict si  $j < n$ .

### II.B. Transformation en arbre binaire

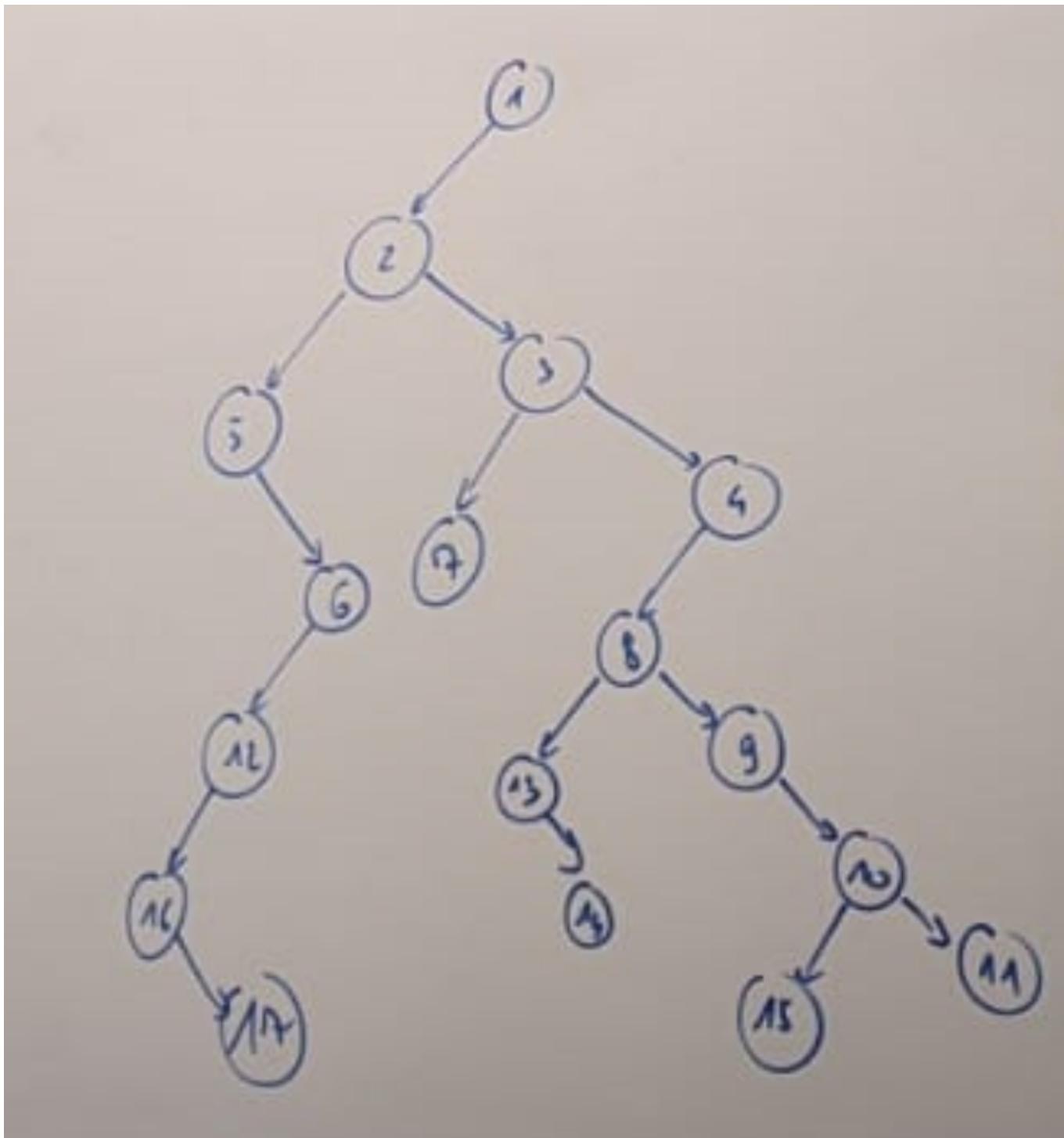
 Ex:



$\dashrightarrow$  : Pointeur

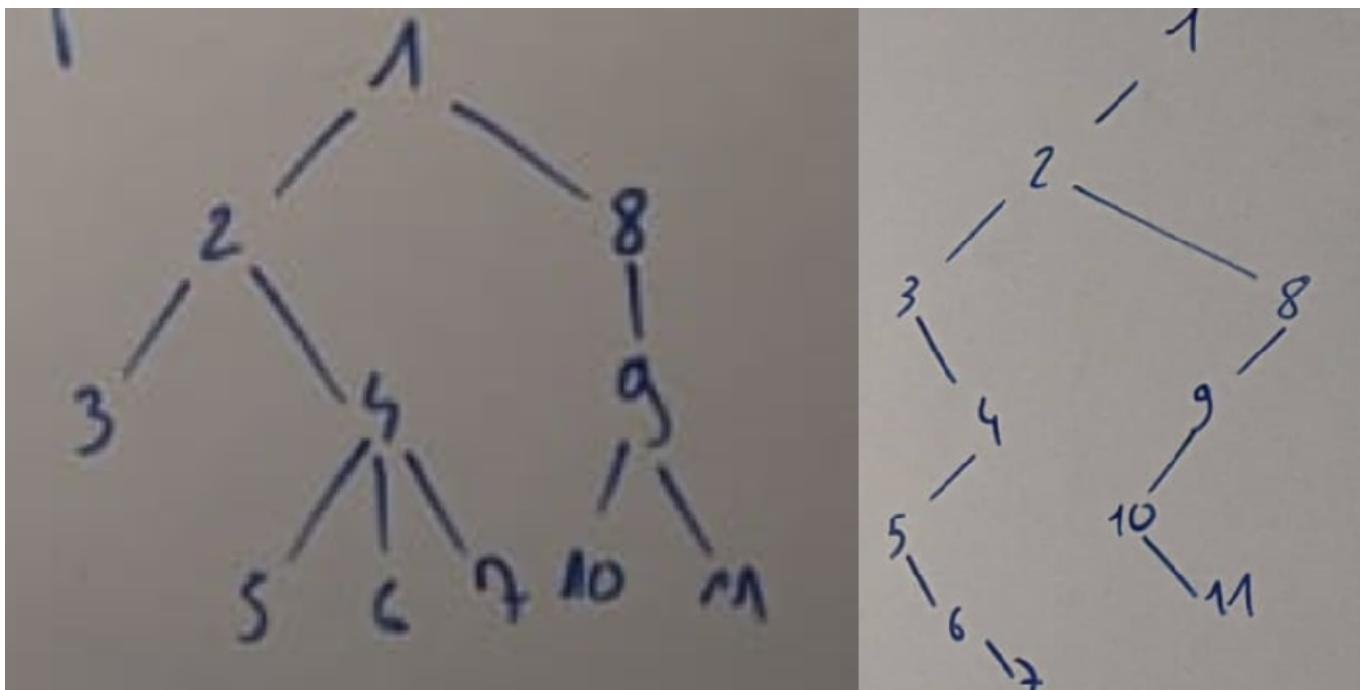


La descendance est codée par une liste chaînée.

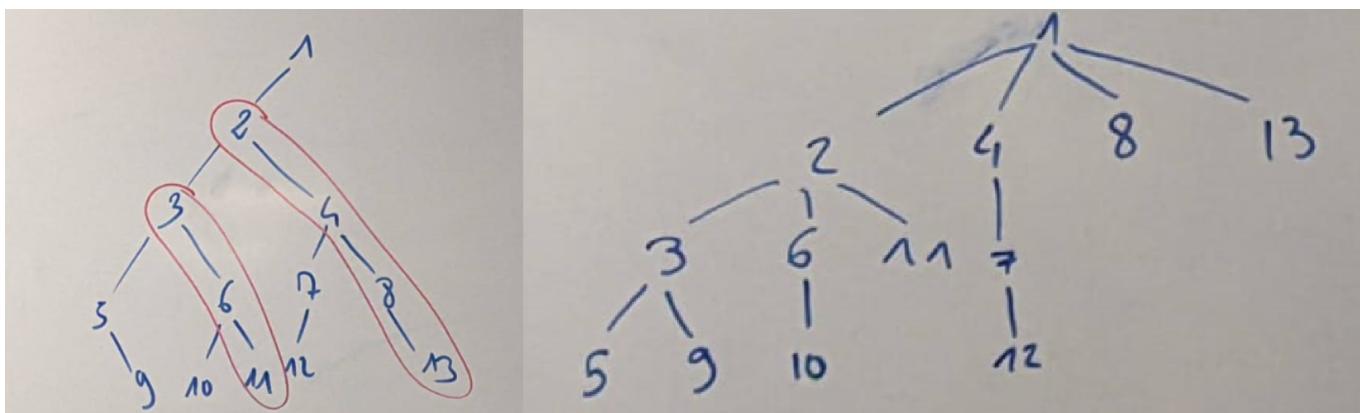


La manière standard de transformer un arbre quelconque en un arbre binaire (non strict) est représentation LCRS. (*left child, right sibling = fils de gauche, frère droit*)

Ex: transformer l'arbre suivant en arbre binaire.



Dans l'autre sens :



Exercice :

1. Écrire une fonction

```
vers_binaire : arbre_n_aire -> arbre_binaire_non_strict
```

2. Écrire une fonction :

```
vers_n_aire : arbre_binaire_non_strict -> arbre_n_aire
```

```
(* Représentation LCRS *)
type 'a arbre = Nn of 'a * ('a arbre list);;
type 'a arbre_bin_ns =
| V2
| N2 of 'a * arbre_bin_ns * arbre_bin_ns

let vers_binaire arbre_n_aire =
match arbre_n_aire with
| Nn (r, []) -> N2 (r, V2, V2)
| Nn (r, head::tail) -> N2 jspjspJSP
```

## Correction :

```
type 'a arb_bin    = V | N of 'a * 'a arb_bin * 'a arb_bin;;
type 'a arb_naire = Noeud of 'a * 'a arb_naire list;;
```

```
let to_bin (arb_naire : 'a arb_naire) : 'a arb_bin =
  let rec aux a f =
    (*binarise a dont la liste des frères est f*)
    match a, f with
    | Noeud(r, []), [] -> N(r, V, V)
    | Noeud(r, g::q), [] -> N(r, aux g q, V)
    | Noeud(r, []), f1::fq -> N(r, V, aux f1 fq)
    | Noeud(r, g::q), f1::fq -> N(r, aux g q, aux f1 fq) in
      aux arb_naire []
  ;;

```

```
let to_naire arb_bin =
  let rec aux a =
    match a with
    | V -> []
    | N(r, g, d) -> Noeud(r, aux g) :: aux d in
  match aux arb_bin with
  | [arb] -> arb
  | _ -> failwith "erreur"
;;

```

