	Structures de données	NSI T <sup>ale</sup>
	Les Arbres	Cours/TD

Source : [https://qkzk.xyz/docs/nsi/cours\\_terminale/structures\\_donnees/arbres/](https://qkzk.xyz/docs/nsi/cours_terminale/structures_donnees/arbres/)  
\*

## 1. Qu'est-ce qu'un arbre ?

Dessignons un arbre :



## 2. Arbre en informatique

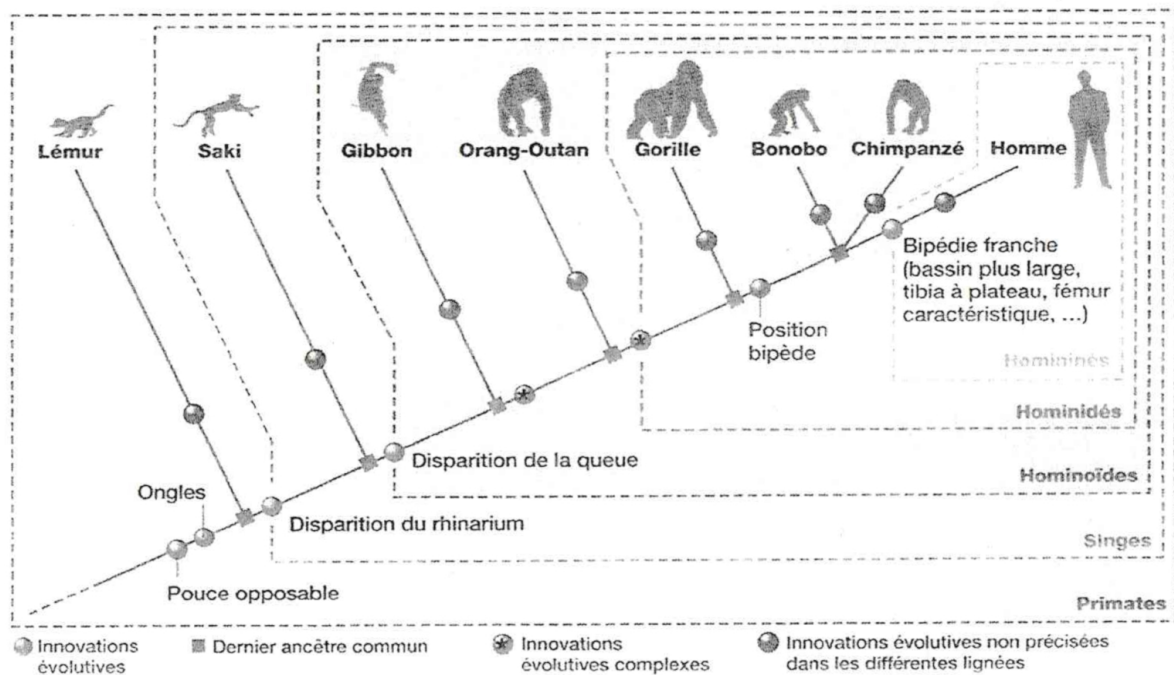
Les **arbres** sont des structures de données :

- hiérarchiques
- naturellement récursives,

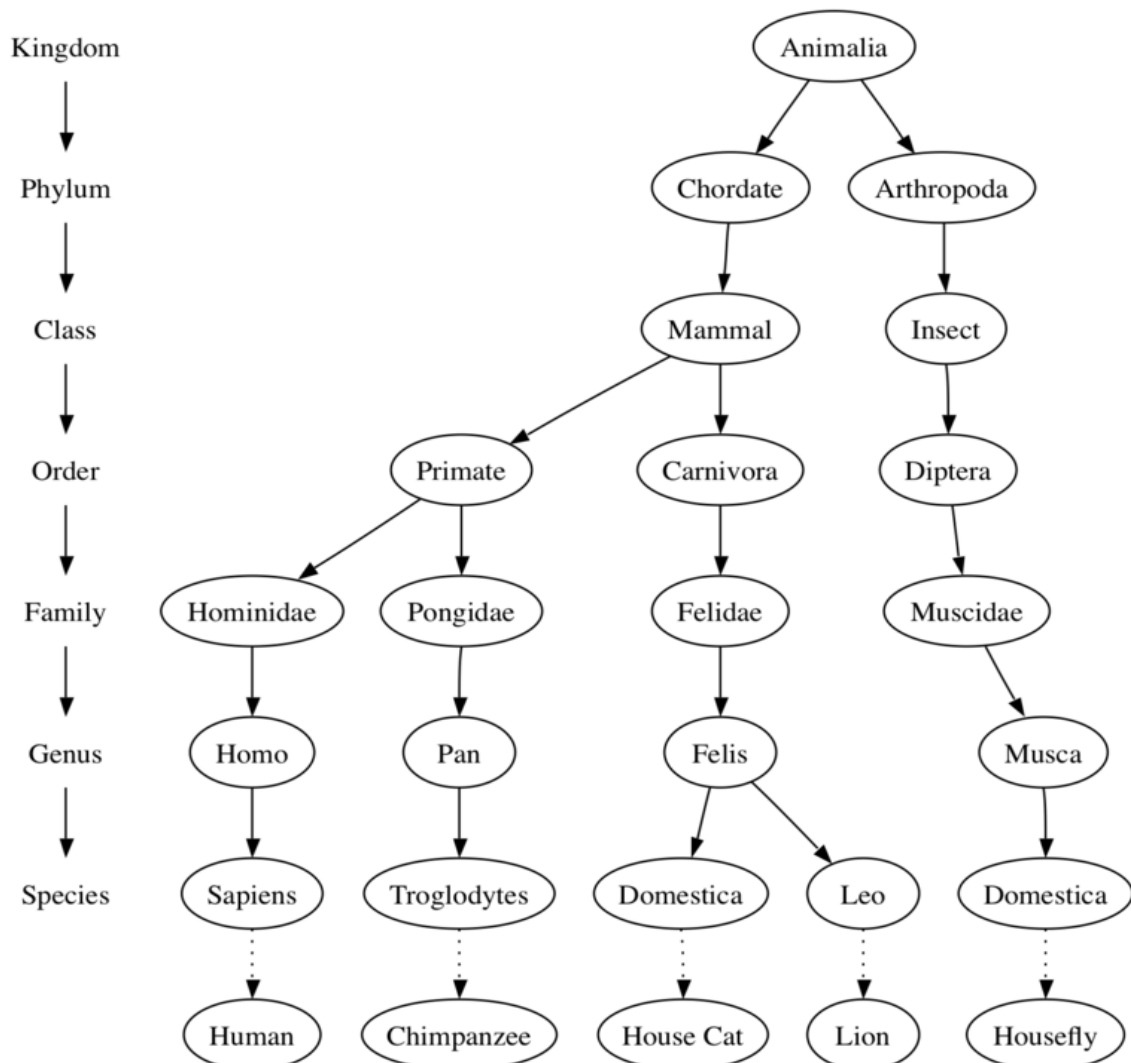
utilisées pour représenter des ensembles de données structurées hiérarchiquement.

### 3. Quelques exemples

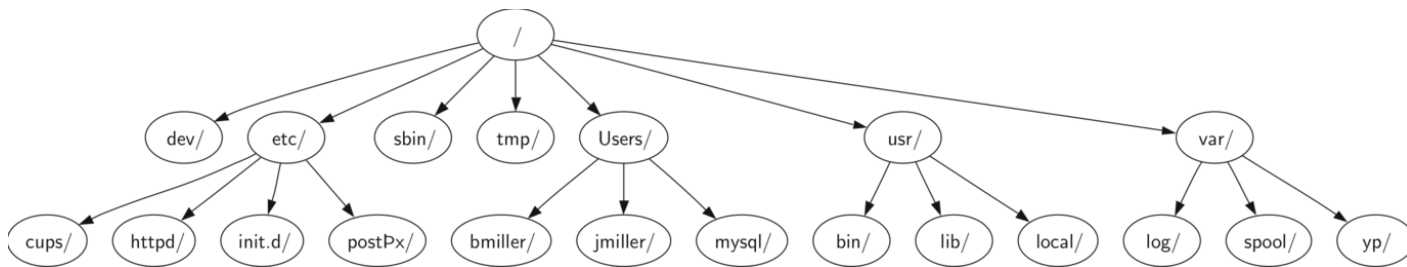
En biologie :



ou :

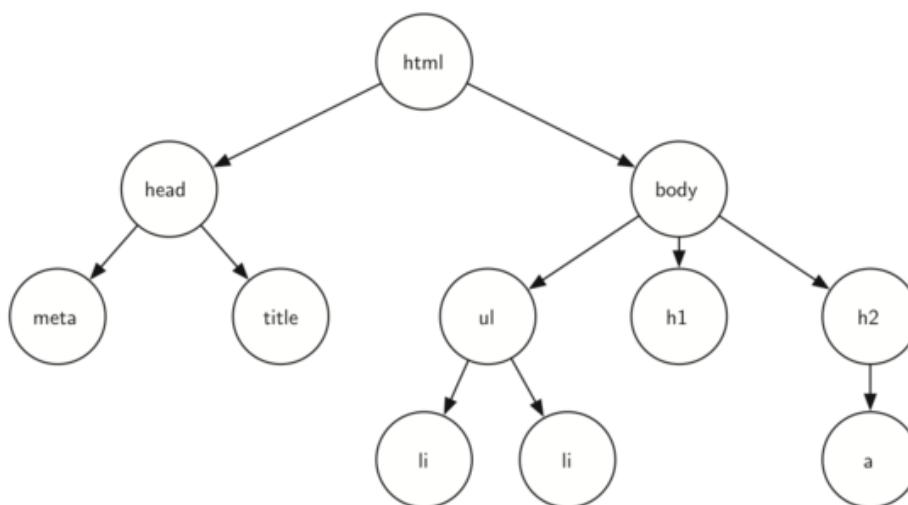


Les dossiers d'un ordinateur :



Les balises d'une page web forment un arbre (appelé DOM) :

```
<html xmlns="http://www.w3.org/1999/xhtml"
  xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type"
    content="text/html; charset=utf-8" />
  <title>simple</title>
</head>
<body>
<h1>A simple web page</h1>
<ul>
  <li>List item one</li>
  <li>List item two</li>
</ul>
<h2><a href="http://www.cs.luther.edu">Luther CS </a></h2>
</body>
</html>
```



Un arbre est une structure de données non-linéaire (comparée aux tableaux, listes, piles, et files qui sont des structures linéaires).

Une structure de données arborescente peut être définie récursivement comme un ensemble de nœuds accessibles depuis un nœud racine. Chaque nœud étant une structure composée d'une valeur et d'une liste de références vers d'autres nœuds, avec les contraintes suivantes : aucune référence n'est dupliquée (chaque nœud a un unique parent), et aucune référence ne désigne le nœud racine (qui n'a donc pas de parent).

Nous allons nous restreindre aux *arbres binaires* pour lesquels la liste des références vers d'autres nœuds comporte au plus deux éléments.

Ces arbres binaires sont largement utilisés, par exemple sous forme d'[ABR – arbres binaires de recherche](#) –, de [tas](#), d'arbres équilibrés comme les [AVL](#), ou encore d'[arbres bicolores rouge-noir](#).

## 4. Un peu (beaucoup...) de vocabulaire préliminaire

### → Terminologie

- un **nœud** est caractérisé par
  - une donnée (on parle aussi d'étiquette),
  - un nombre fini de fils.
- une **arête** relie deux nœuds. Chaque nœud, à l'exception de la racine, est relié à un autre nœud, son père, par exactement une arête entrante. Chaque nœud peut avoir une ou plusieurs arêtes sortantes le reliant à ses fils. On parle aussi de *lien*.
- la **racine** d'un arbre est le seul nœud sans père.
- un **chemin** est une liste de nœuds reliés par des arêtes.
- une **branche** est le chemin le plus court reliant un nœud à la racine.
- les **fils** sont l'ensemble des nœuds reliés à un même nœud par des arêtes entrantes.
- le **père** ou **parent** est le nœud relié à ses nœuds fils par une arête sortante.
- un **sous-arbre** est l'ensemble des nœuds et arêtes d'un nœud parent et de ses descendants.
- une **feuille** est un nœud sans fils.
- un **nœud interne** est un nœud qui n'est pas une feuille.

Pour assurer la cohérence de ces définitions, on considère que l'arbre vide n'est pas un nœud.

### → Quelques mesures sur les arbres

- la **taille** d'un arbre est le nombre de nœuds de l'arbre.
- la **profondeur d'un nœud** (ou hauteur d'un nœud) est le nombre d'arêtes sur la branche qui le relie à la racine. La profondeur de la racine est nulle.
- la **hauteur d'un arbre** est la profondeur maximale de l'ensemble des nœuds de l'arbre.
- l'**arité d'un nœud** est le nombre de fils du nœud.
- l'**arité d'un arbre** est le nombre maximal de fils des nœuds de l'arbre.

### → *Définition.* Arbre binaire

Un **arbre binaire** est donc un arbre d'arité deux.

Un arbre binaire est

- soit l'arbre vide, noté  $\Delta$  ;
- soit un triplet  $(e, g, d)$ , appelée **nœud**, dans lequel
  - $e$  est l'élément, ou encore **étiquette**, de la racine de l'arbre,
  - $g$  est le **sous-arbre gauche** de l'arbre, et
  - $d$  est le **sous-arbre droit** de l'arbre.

Les sous-arbres gauche et droit d'un arbre binaire non vide sont eux-mêmes des arbres binaires. La structure d'arbre binaire est donc une structure récursive.

- on appelle **fils gauche**, resp. **fils droit**, le sous-arbre gauche, resp. droit, d'un nœud.

## 5. Se familiariser avec les arbres binaires

### → Quelques arbres binaires

Dessinez chacun des arbres ci-dessous, donnez sa taille et sa hauteur, le nombre de feuilles, le nombre de nœuds à chaque profondeur.

1.  $(1, \Delta, \Delta)$
2.  $(3, (1, \Delta, (4, (1, \Delta, (5, \Delta, \Delta)), \Delta)), \Delta)$
3.  $(3, (1, (1, \Delta, \Delta), \Delta), (4, (5, \Delta, \Delta), (9, \Delta, \Delta)))$
4.  $(3, (1, (1, \Delta, \Delta), (5, \Delta, \Delta)), (4, (9, \Delta, \Delta), (2, \Delta, \Delta)))$

### → Des arbres binaires

- Combien de feuilles au minimum comporte un arbre binaire de hauteur  $h$  ? Au maximum ?
- Combien de nœuds au minimum comporte un arbre binaire de hauteur  $h$  ? Au maximum ?

### → Squelettes d'arbres binaires

On appelle squelette ou forme d'arbres binaires tout arbre binaire dans lequel on ne tient pas compte des étiquettes.

- Combien y a-t-il de squelettes d'arbres binaires de taille 0, de taille 1 ?
- Dessinez tous les squelettes d'arbres binaires de taille 2, 3, 4 ; donnez-en le nombre.

### → Taille et hauteur

Proposez des algorithmes pour calculer

- la **taille** d'un arbre binaire
- la **hauteur** d'un arbre binaire. On conviendra conventionnellement que la hauteur de l'arbre vide  $\Delta$  est -1.

## 6. Caractériser les arbres binaires

Les arbres binaires sont caractérisés par le fait que chaque nœud possède au plus deux fils.

D'autres caractéristiques sont définies, qui permettent par exemple d'identifier des arbres pour lesquels le coût de certaines opérations sera minimal, ou de définir des algorithmes spécifiques à ces arbres.

- Un **arbre binaire filiforme ou dégénéré** est un arbre dans lequel tous les *nœuds internes* n'ont qu'un seul fils. (Un arbre filiforme ne possède donc qu'une unique feuille.)
- Un **arbre binaire localement complet** ou **arbre binaire strict** est un arbre dont tous les *nœuds internes* possèdent exactement deux fils. (Autrement dit, un arbre binaire localement complet est un arbre dont chaque nœud possède zéro ou 2 fils. L'arbre vide n'est pas localement complet.)
- Un **arbre binaire complet** est un arbre binaire localement complet dans lequel toutes les feuilles sont à la même profondeur. (Il s'agit d'un arbre dont tous les niveaux sont remplis.)
- Un **arbre binaire parfait** est un arbre dans lequel tous les niveaux sont remplis à l'exception éventuelle du dernier, et dans ce cas les feuilles du dernier niveau sont alignées à gauche.
- Un **arbre binaire équilibré** est un arbre dont les deux fils sont des arbres équilibrés dont les hauteurs diffèrent d'au plus une unité. Ainsi, l'accès à n'importe lequel des *nœuds* est en moyenne minimisé.

Comme le mentionne la [page Wikipedia Arbre binaire#Types d'arbres binaires](#), il existe des usages contradictoires des termes complet et parfait qui peuvent parfois être intervertis. Des confusions peuvent aussi exister entre le français et l'anglais, dans lequel on trouve les termes perfect et complete. La terminologie utilisée ici est cohérente avec celle de Froidevaux et al.[2](#), alors que Beauquier et al.[3](#) en utilisent une autre.

## 7. Se repérer dans cette forêt d'arbres

### → Compter ces arbres

- Combien de nœuds au maximum comporte un arbre localement complet de hauteur  $h$  ?  
Au minimum ?
- Combien de nœuds comporte un arbre complet de hauteur  $h$  ?
- Combien de squelettes d'arbres parfaits de hauteur  $h$  existe-t-il ?
- Combien de squelettes d'arbres filiformes de hauteur  $h$  existe-t-il ?

### → Reconnaître ces arbres

Proposez des algorithmes pour les prédicats

- reconnaître un arbre binaire filiforme
- reconnaître un arbre binaire localement complet
- reconnaître un arbre binaire complet
- reconnaître un arbre binaire parfait

### → Superposer deux arbres

Proposez un prédicat pour tester l'égalité du squelette de deux arbres binaires.

### → Numéroté les nœuds

La numérotation de Sosa-Stradonitz des nœuds d'un arbre binaire, utilisée notamment en généalogie, est la suivante :

- le nœud racine est numéroté par 0
- si un nœud numéroté par  $i$
- son fils gauche est numéroté par  $2i+1$
- son fils droit est numéroté par  $2(i+1)$

Cette numérotation peut être utilisée pour représenter un arbre dans un tableau.

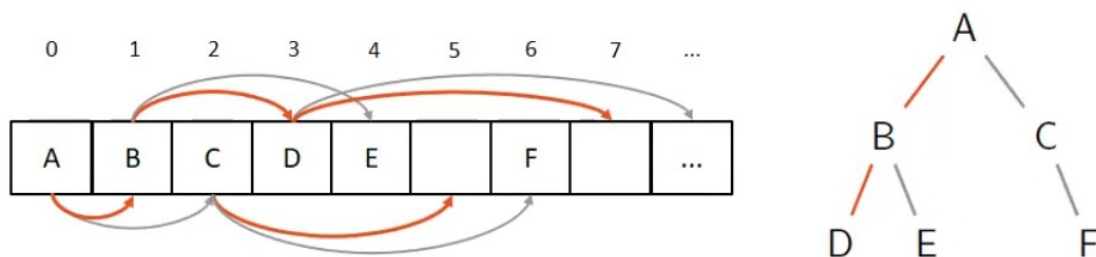
## 8. Représentations des arbres binaires

### → Représentation par un tableau

La représentation d'un arbre binaire par un tableau repose sur la relation suivante entre les indices d'un père et de ses fils :

$$\begin{cases} i_{\text{filsGauche}} = 2 \times i_{\text{père}} + 1 \\ i_{\text{filsDroit}} = 2 \times (i_{\text{père}} + 1) \end{cases}$$

Exemple :



Combien d'éléments doit contenir un tableau utilisé pour représenter

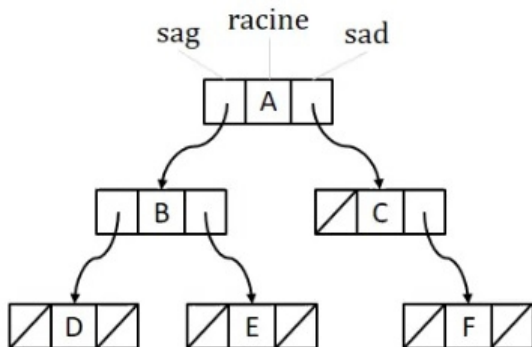
- un arbre binaire complet de  $n$  nœuds ?
- un arbre binaire parfait de  $n$  nœuds ?
- un arbre binaire quelconque de  $n$  nœuds (dans le pire des cas) ?

Nous voyons assez facilement l'inconvénient d'une telle représentation : si l'arbre a beaucoup de "trous", nous nous retrouvons avec un tableau comportant beaucoup de cases vides. Nous lui privilégierons la seconde représentation.

### → Représentation par une structure récursive : le nœud

Le nœud est une structure récursive définie par :

- Une valeur (la donnée du nœud)
- Les références vers le sous-arbre gauche et le sous-arbre droit (qui sont donc eux-mêmes représentés par un nœud, d'où la définition *récursive*)

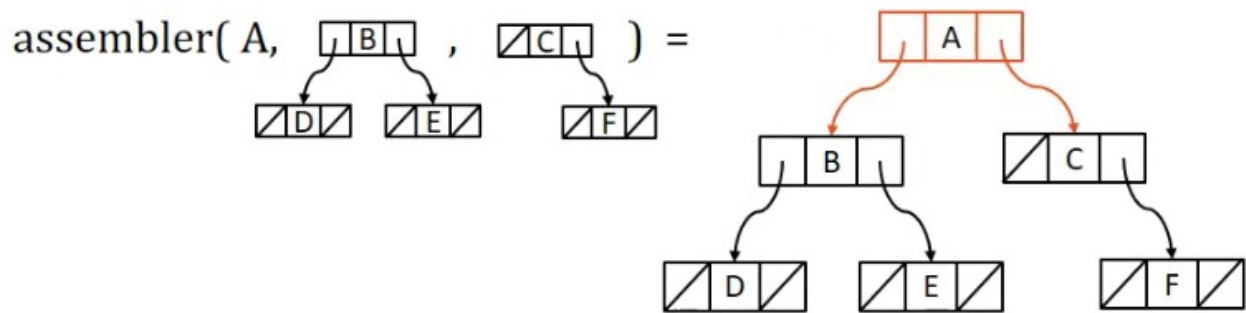


```
class nœud :  
    # Constructeur  
    def __init__(self, valeur) :  
        self.donnee = valeur  
        self.gauche = None  
        self.droit = None  
    # Affichage  
    def __str__(self) :  
        return str(self.donnee)
```

A partir de cette représentation, on peut définir par exemple l'opération *assembler* :

**Élément x ArbreBinaire x ArbreBinaire → ArbreBinaire**

Avec cette représentation, l'opération *assembler* est particulièrement facile à comprendre, et à écrire.



```
class nœud :
    [...]
    # Assembler
    def assembler(self, ag, ad) :
        self.gauche = ag
        self.droit = ad
```

C'est cette représentation que nous utiliserons de préférence.



## 9. Spécification du Type Abstrait de Données (TAD) Arbre Binaire

### → Spécifications d'un TAD :

Les **spécifications** d'un Type Abstrait de Données comportent 2 sections

- La **signature** (la syntaxe) : elle décrit le TAD et est composée :
  - Du **Type** : c'est le nom du TAD
  - Des **Opérations** : elles sont décrites en indiquant les types des paramètres en entrée et en sortie
- La **sémantique** : elle décrit l'utilisation du TAD et est composée :
  - Des **Préconditions** : les conditions de fonctionnement des opérations
  - Des **Axiomes** : qui indiquent les résultats attendus des opérations

### → Spécifications du TAD Arbre Binaire :

#### Type

Arbre Binaire

#### Opérations

Soit E le type de l'étiquette de l'arbre, sag (sad) est l'opération pour accéder au sous-arbre gauche (droit).

- creerArbreVide :  $\rightarrow \text{ArbreBinaire}[E]$
- estVide :  $\text{ArbreBinaire}[E] \rightarrow \text{Booléen}$
- racine :  $\text{ArbreBinaire}[E] \rightarrow E$
- sag, sad :  $\text{ArbreBinaire}[E] \rightarrow \text{ArbreBinaire}[E]$
- assembler :  $E \times \text{ArbreBinaire}[E] \times \text{ArbreBinaire}[E] \rightarrow \text{ArbreBinaire}[E]$

#### Préconditions

Soit a un ArbreBinaire.

- racine(a), sag(a) et sad(a) si et seulement si non estVide(a)

#### Axiomes

Soient a, g, d des ArbreBinaire[E], r un E.

- $\text{estVide}(\text{arbreVide}) = \text{Vrai}$
- $\text{estVide}(\text{assembler}(r, g, d)) = \text{Faux}$
- $\text{racine}(\text{assembler}(r, g, d)) = r$
- $\text{sag}(\text{assembler}(r, g, d)) = g$
- $\text{sad}(\text{assembler}(r, g, d)) = d$

## 10. Parcours des arbres binaires

### → Parcours d'arbre

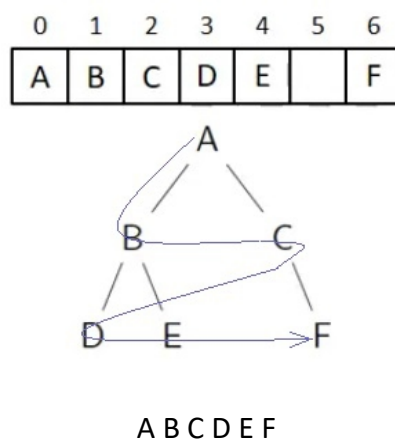
Notre objectif est de **visiter tous les nœuds** d'un arbre, sans pour autant visiter deux fois le même nœud.

Dans une liste, il suffit de lire les éléments l'un après l'autre ; mais comme l'arbre est une structure de données non séquentielle, c'est un peu plus complexe.

### → Parcours en largeur

Le **parcours en largeur** consiste, simplement, à **lire les nœuds dans le sens de la lecture**. Si l'arbre est représenté par un tableau, cela revient simplement à lire les cases de celui-ci, l'une après l'autre.

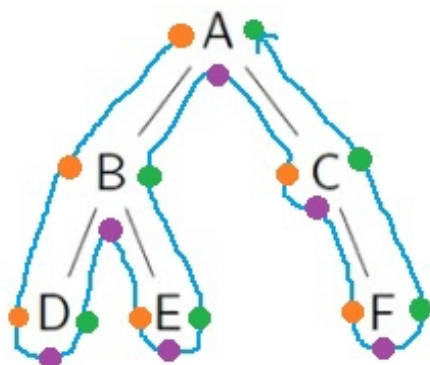
Exemple :



### → Parcours en profondeur d'abord

Le **parcours en profondeur** est plus aventureux, il consiste à explorer un **chemin jusqu'au bout**, avant de revenir en arrière pour emprunter un autre chemin. Cet algorithme s'écrit récursivement, pour s'adapter facilement à notre structure.

Remarquons que nous passons sur chaque nœud à trois reprises.



– Lorsqu’on traite un nœud dès sa **première visite** (couleur orange), nous obtenons l’**ordre préfixe** des nœuds. On parle aussi de **parcours RGD** (Racine Gauche Droit).

Algorithmiquement, cela revient à faire

```
[nœud] + appelRécursif sur sag + appelRécursif sur sad
```

– Lorsqu’on traite un nœud lors de sa **deuxième visite** (couleur violette), nous obtenons l’**ordre infixé** des nœuds. On parle aussi de **parcours GRD** (Gauche Racine Droit).

Algorithmiquement, cela revient à faire

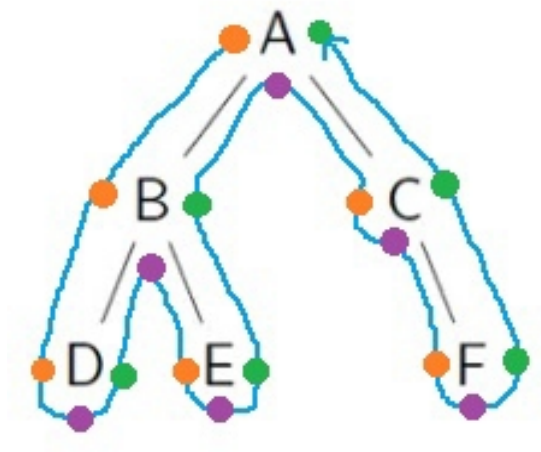
```
appelRécursif sur sag + [nœud] + appelRécursif sur sad
```

– Lorsqu’on traite un nœud lors de sa **dernière visite** (couleur verte), nous obtenons l’**ordre postfixé** des nœuds. On parle aussi de **parcours GDR** (Gauche Droit Racine).

Algorithmiquement, cela revient à faire

```
appelRécursif sur sag + appelRécursif sur sad + [nœud]
```

Exemple



– Parcours **préfixe** : A B D E C F

– Parcours **infixe** : D B E A C F

– Parcours **postfixe** : D E B F C A

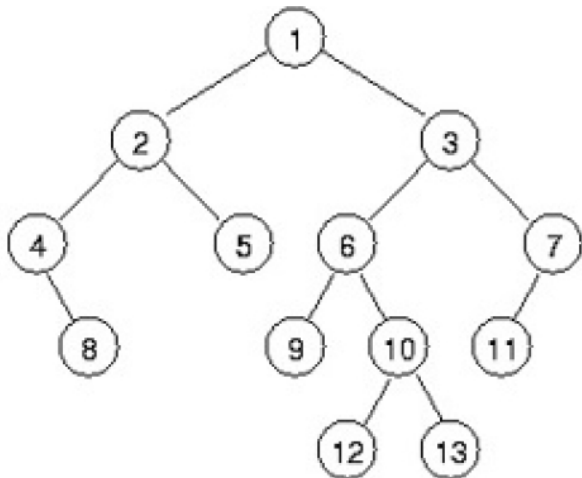
## 11.À suivre...

Il s’agira de découvrir les ABR, *arbres binaires de recherche*.

## 12.Exercices d'application :

### Exercice 1 : Arbre binaire et représentation par un tableau

Compléter le tableau suivant pour associer à chaque arbre son schéma et sa représentation par un tableau

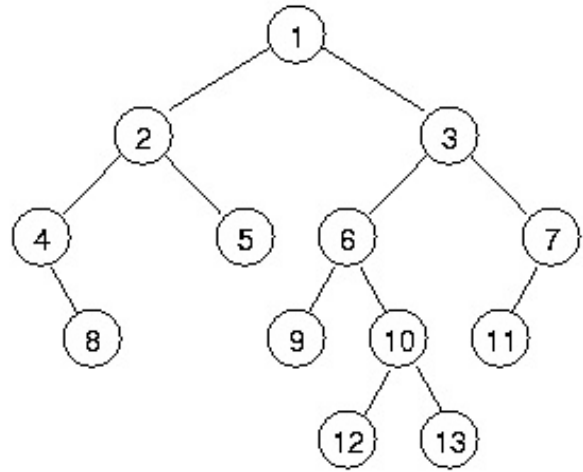
Schéma	Tableau															
																
	<table><tr><td>N</td><td>S</td><td>I</td><td>L</td><td>E</td><td></td><td>O</td><td>N</td><td></td><td>B</td><td>L</td><td></td><td></td><td>U</td><td>M</td></tr></table>	N	S	I	L	E		O	N		B	L			U	M
N	S	I	L	E		O	N		B	L			U	M		

## Exercice 2 : Parcours d'un arbre

Donner l'ordre du

1. parcours en largeur
2. parcours en profondeur préfixe
3. parcours en profondeur infixé
4. parcours en profondeur postfixé

pour l'arbre ci-contre.



Est-il possible de retrouver l'arbre, seulement à partir du résultat de son parcours ?