# Les arbres

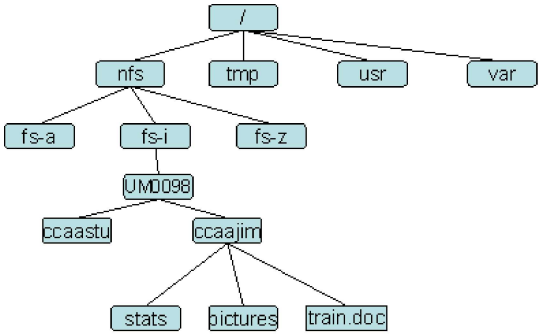
*Dernière mise à jour le : 05/01/2024*

## Introduction

Un **arbre** est une structure *hiérarchique* permettant de représenter de manière symbolique des informations structurées.

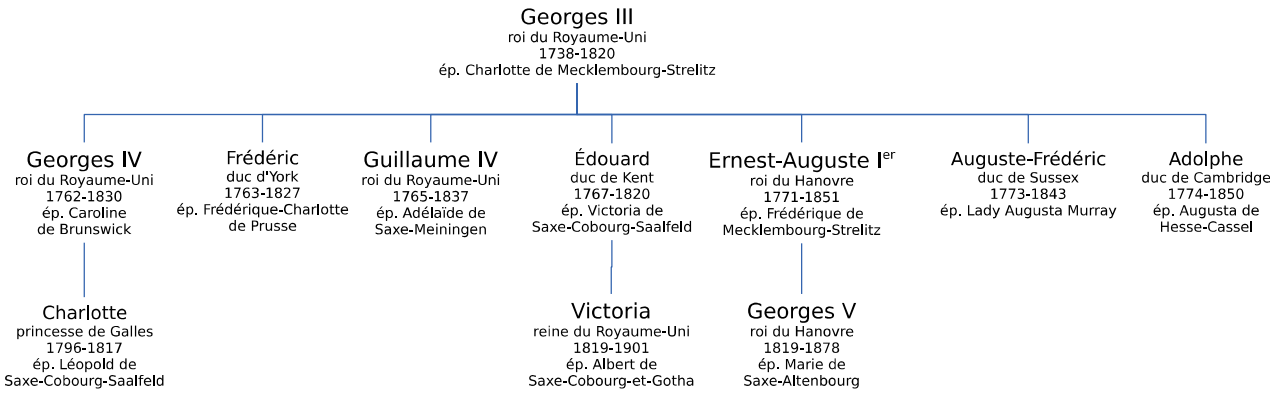
Par exemple :

- Un **dossier, contenant des dossiers et des fichiers**, chaque dossier pouvant contenir des dossiers et des fichiers :



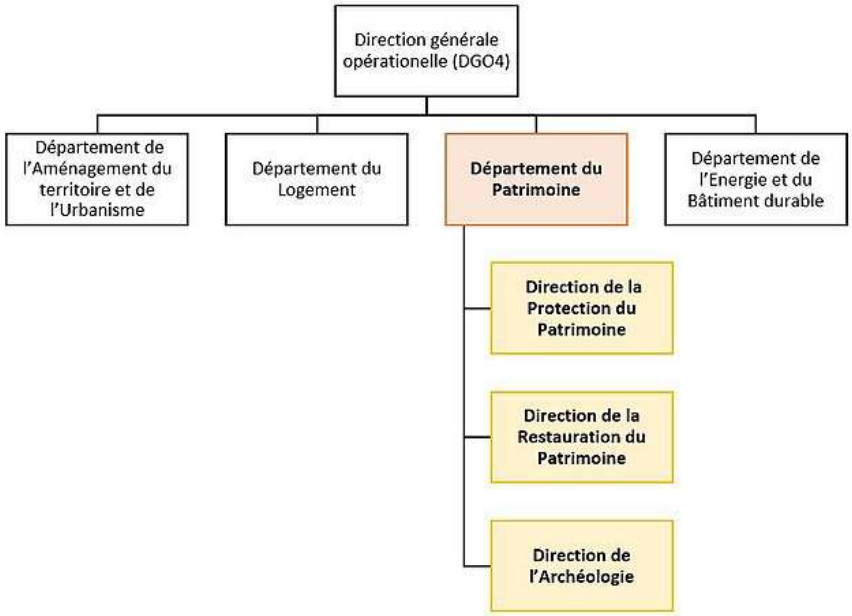

Crédits : Jimbotyson, CC BY-SA 3.0, via Wikimedia Commons.

- Un **arbre généalogique** des descendants ou des ascendants :



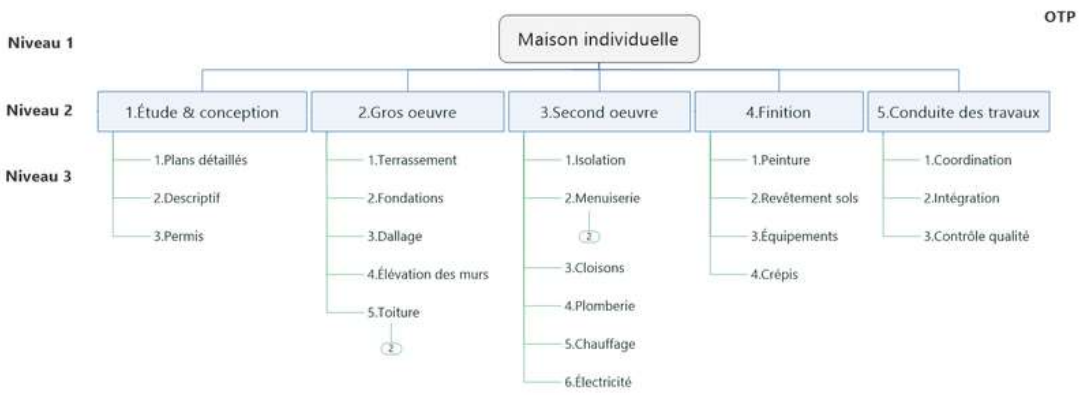

Crédits : Ælfgar, CC BY-SA 3.0, via Wikimedia Commons

- Un **organigramme** :



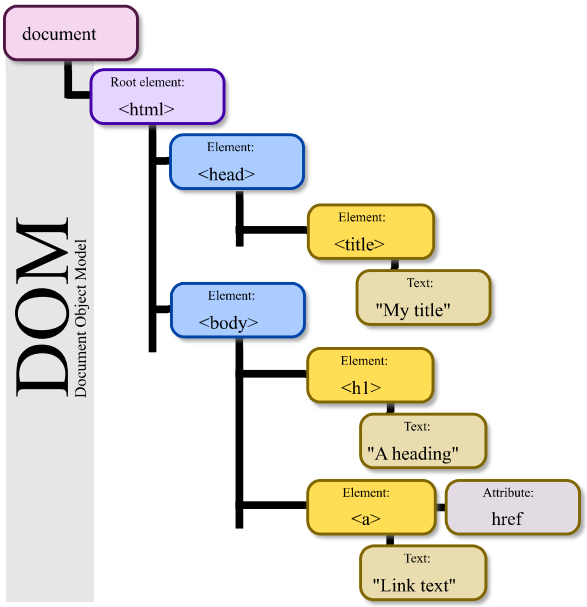

Crédits : Adriencld, CC BY-SA 4.0, via Wikimedia Commons

- Une **tâche complexe décomposée en tâches élémentaires et en tâches complexes** :




Crédits : Cth027, CC BY-SA 4.0, via Wikimedia Commons

- La **structure d'une page Web** (le DOM (Document Object Model) permet alors de modifier une page Web en modifiant, en ajoutant ou en supprimant des noeuds de l'arbre) :




Crédits : Birger Eriksson, CC BY-SA 3.0, via Wikimedia Commons

Dans le même ordre d'idée, le format JSON a également une structure arborescente.

## Arbres quelconques

Dans tous ces exemples, on a défini un cas où l'information est élémentaire (fichier, tâche élémentaire), et un cas général où l'information structurée contient deux ou plusieurs informations de même structure.

Dans la terminologie informatique, on utilise les termes de

- **feuille** pour les informations élémentaires,
- **noeud** pour chaque embranchement de l'arbre,
- **racine** pour le(s) noeud(s) principal(aux).

> **Attention** : l'analogie avec les arbres réels peut s'avérer trompeuse. Les arbres - en informatique - sont le plus souvent représentés avec la racine en haut, puis les noeuds, et les feuilles en bas.

Il s'agit d'une structure de données absraite permettant de représenter une collection de données par des noeuds organisés de manière hiérarchique : il y a un (parfois plusieurs) noeud racine et chaque noeud dépend d'un *antécédent* (sauf la racine) et a des *descendants* (sauf les feuilles).

Dans le vocabulaire des arbres on utilise les termes *père* et *fils* pour désigner respectivement un antécédent et les descendants.

- **Père** : chaque noeud possède exactement un seul noeud *père*, celui dont il est issu, à l'exception de la racine qui n'en a pas.
- **Fils** : chaque noeud peut avoir un nombre arbitraire de noeuds *fils*, dont il est le père.

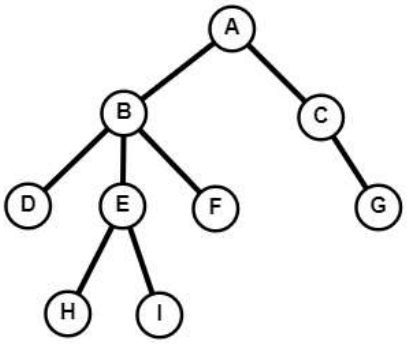
Ainsi, avec ces définitions :

- les **feuilles** sont les noeuds qui n'ont pas de fils,
- un noeud qui n'a pas de père s'appelle une **racine**.

Tous les noeuds qui ne sont pas des feuilles sont appelés des **noeuds internes** (et le feuilles parfois appelées des *noeuds externes*).

L'intérêt des arbres est d'y stocker de l'information. Pour cela, chaque noeud peut contenir une ou plusieurs valeurs. L'information portée par un noeud s'appelle l'**étiquette** du noeud (ou la *valeur*, ou la *clé*).

## Exemple




Dans cet arbre :

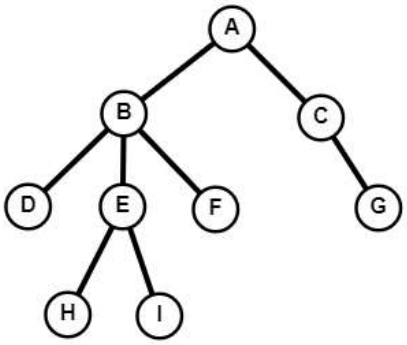
- La racine est le noeud A.
- Le noeud B possède 3 fils (les noeuds D, E et F), le noeud C possède un fils (le noeud G), le noeud F ne possède aucun fils.
- Le noeud B a pour père le noeud A.
- Les feuilles sont les noeuds D, H, I, F et G (ceux qui n'ont pas de fils).

## Caractéristiques d'un arbre

- La **taille** d'un arbre est le nombre de noeuds qu'il possède.
- La **profondeur** d'un noeud est la longueur du chemin le plus court entre ce noeud et la racine (la racine a donc une profondeur égale à 0).
- La **hauteur** d'un arbre est la profondeur maximale de ses noeuds (elle vaut 0 pour l'arbre réduit à sa racine et $-1$ par convention pour un arbre vide).

⚠ **Attention** : On trouve aussi dans la littérature, que la profondeur de la racine est égale à 1, ce qui modifie la hauteur de l'arbre également puisqu'alors l'arbre réduit à la racine a pour hauteur 1 et l'arbre vide a pour hauteur 0. Les deux définitions se valent, il faut donc bien lire celle qui est donnée.

## Exemple




- La taille de l'arbre est égale à 9 (il possède 9 noeuds : 4 noeuds internes et 5 feuilles).
- Le noeud E a une profondeur égale à 2 (le chemin A-B-E a une longueur égale à 2).
- La hauteur de l'arbre est égale à 3 (la prodondeur maximale est égale à 3, c'est celle des noeuds les plus profonds : H et I).

*Dans la suite, on ne s'intéressera qu'aux arbres dont les noeuds ont au plus deux fils.*

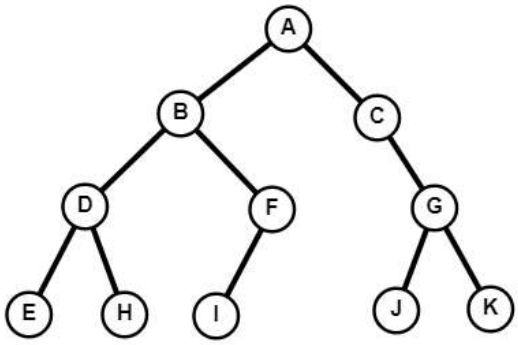
# Arbres binaires

Seuls les arbres binaires sont au programme de Terminale NSI.

## Définition et vocabulaire spécifique

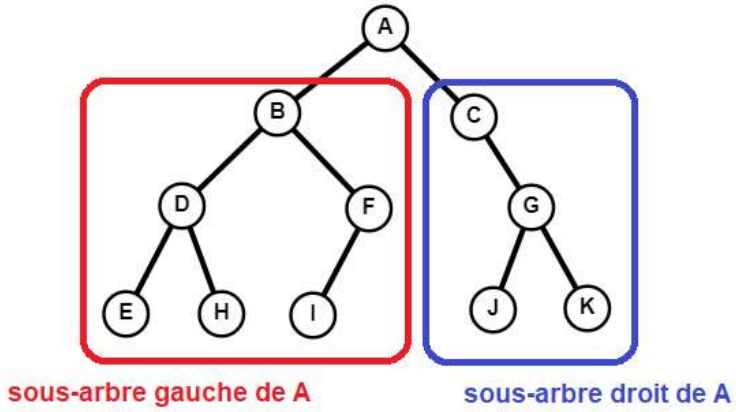
> Un **arbre binaire** est un arbre dont tous les noeuds ont au plus deux fils.

L'arbre vu dans le paragraphe précédent n'est pas binaire car le noeud B possède 3 fils.
En revanche, l'arbre suivant est binaire.




Les définitions vues précédemment pour des arbres quelconques restent bien évidemment valables pour les arbres binaires. Dans le cas d'un arbre binaire, chaque noeud possède deux sous-arbres, éventuellement vides, que l'on appelle **sous-arbre gauche** et **sous-arbre droit**.

Par exemple, dans le cas de l'arbre binaire précédent, le noeud A possède un sous-arbre gauche et un sous-arbre droit comme la figure suivante le montre.




Les sous-arbres gauche et droit de A sont eux-mêmes des arbres dont les racines sont respectivement B et C. Ces noeuds possèdent eux-même des sous-arbres gauche et droit. Par exemple, le noeud C possède un sous-arbre gauche, qui est vide, et un sous-arbre droit qui est l'arbre dont la racine est G. Ainsi de suite...

> Faites tous les exercices proposés dans le notebook d'exercices.

*Ce qui suit est un résumé de ce qui a été vu dans les exercices.*

## Type abstrait `Arbre binaire`

De manière générale, on peut construire un arbre binaire comme un noeud composé de deux sous-arbres. L'arbre vide est représentée par la valeur `None`. Ainsi, une feuille est un noeud avec les sous-arbres gauche et droit à `None`. Pour annoter la structure de l'arbre avec des informations, on utilise des étiquettes pouvant être enregistrées à chaque noeud.

On peut ensuite parcourir un arbre par l'accès à son étiquette et à ses sous-arbres droit et gauche. Un prédicat permet de distinguer les feuilles des noeuds.

On peut ainsi spécifier un arbre binaire par le type abstrait suivant :

- Constructeur : `noeud : Etiquette x Arbre binaire x Arbre binaire -> Arbre binaire`
- Sélecteurs :

    - `droit : Arbre binaire -> Arbre binaire`
    - `gauche : Arbre binaire -> Arbre binaire`
    - `etiquette : Arbre binaire -> Etiquette`
- Prédicat : `est_feuille : Arbre binaire -> Booléen`

## Implémentation

Il existe, comme toujours, plusieurs implémentations possibles d'un arbre binaire. Une implémentation classique consiste à représenter chaque noeud comme un objet d'une classe `Noeud`.

```
class Noeud:
    def __init__(self, e, g=None, d=None):
        self.etiquette = e
        self.gauche = g
        self.droit = d

    def est_feuille(self):
        return not self.gauche and not self.droit

    # Une représentation possible de l'arbre
    def __repr__(self):
        ch = str(self.etiquette)
        if self.gauche or self.droit:
            ch = ch + '-(' + str(self.gauche) + ',' + str(self.droit) + ')'
        return ch
```

Il y a une petite subtilité à bien comprendre pour la méthode `__init__` : on a choisi de définir la valeur `None` par défaut aux arguments `gauche` et `droit`. Cela permet de construire une feuille (d'étiquette `'a'`) en écrivant

```
Noeud('a')
```

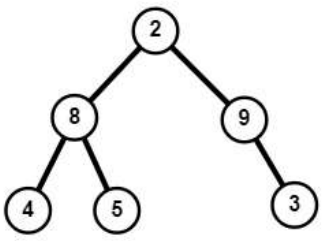
au lieu de

```
Noeud('a', None, None).
```

La construction d'un arbre s'effectue alors avec des noeuds ayant soit un seul argument (cas des feuilles), soit trois (cas général).

```
>>> A1 = Noeud(2, Noeud(8, Noeud(4), Noeud(5)), Noeud(9, None, Noeud(3)))
>>> A1
2-(8-(4,5),9-(None,3))
```

L'arbre `A1` ainsi construit représente l'arbre binaire ci-dessous.




## Calcul de la taille et de la hauteur d'un arbre binaire

La définition d'un arbre binaire étant récursive, il est naturel d'écrire des algorithmes récursifs pour effectuer des opérations sur les arbres binaires.

En particulier, on peut écrire assez facilement deux fonctions récursives `taille` et `hauteur` qui calculent respectivement la taille et la hauteur d'un arbre binaire. Il suffit de parcourir récursivement l'arbre avec les méthodes `gauche` et `droit`.

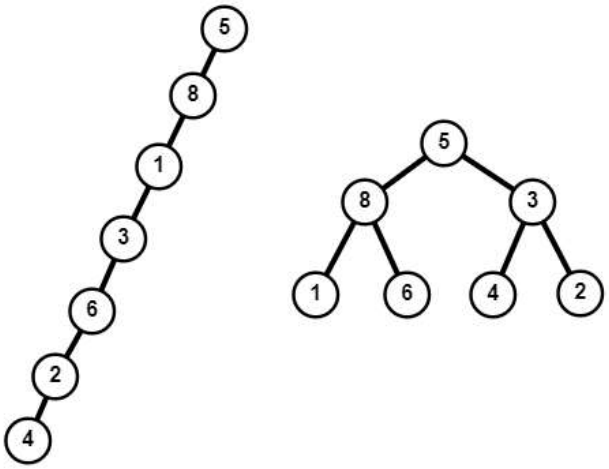
```python
def taille(A):
    """Renvoie la taille d'un arbre binaire A."""
    if A is None:
        return 0
    else:
        return 1 + taille(A.gauche) + taille(A.droit)

def hauteur(A):
    """Renvoie la hauteur d'un arbre binaire A"""
    if A is None:
        return -1
    else:
        return 1 + max(hauteur(A.gauche), hauteur(A.droit))
```

On obtient alors :

```
>>> taille(A1)
6
>>> hauteur(A1)
2
```

## Encadrement de la hauteur d'un arbre binaire

La hauteur d'un arbre binaire est la profondeur maximale de ses noeuds. Cependant un arbre binaire d'une taille donnée peut avoir un aspect totalement différent. En effet, les deux arbres binaires suivants sont de même taille (égale à 7) mais ont des "formes" très différentes.




Arbre 1 Arbre 2

Le premier est dit *filiforme* ou *dégénéré* tandis que le second est dit *parfait* (un arbre est dit *parfait* si tous les niveaux sont remplis c'est-à-dire si chaque noeud interne a exactement deux fils).

Si on cherche à encadrer la *hauteur* d'un arbre binaire, ces deux exemples fournissent les deux cas de figure extrêmes :

- Le premier arbre est l'arbre binaire de taille 7 dont la hauteur est maximale, elle vaut 6 (le noeud le plus profond 2 a une profondeur égale à 6).
- Le second arbre un arbre binaire de taille 7 dont la hauteur est minimale, elle vaut 2 (les 4 feuilles sont toutes à une profondeur égale à 2).

De manière générale, on a l'encadrement suivant.

Si on note $H$ la hauteur d'un arbre binaire à $N$ noeuds, alors :

$$\lfloor \log_2(N) \rfloor \leq H \leq N-1,$$

où $\lfloor \log_2(N) \rfloor$ est la partie entière du logarithme en base 2 de $N$, c'est-à-dire le nombre de bits nécessaire à son écriture en base 2 diminué d'une unité (c'est la définition des informaticiens).

> Cette propriété est expliquée dans un exercice.

On peut vérifier cela avec les arbres précédents de taille $N = 7$.

- D'une part, $N - 1 = 6$ qui correspond bien à la taille de l'arbre filiforme (arbre 1).
- D'autre part, $7 = (111)_2$ donc il faut 3 bits pour écrire 7 en base deux. On en déduit que $\lfloor \log_2(7) \rfloor = 3 - 1 = 2$ qui est bien la hauteur de l'arbre parfait (arbre 2).

Ces deux arbres étant les cas extrêmes, un arbre binaire à 7 noeuds a une hauteur comprise entre 2 et 6.

## Utilisation des arbres

Les applications des arbres sont nombreuses en informatique. Citons par exemple :

- les arbres de jeu qui permettent de représenter toutes les positions et tous les coups possibles d'un jeu (voir exercice 2)
- les arbres syntaxiques permettant de représenter la structure syntaxique d'une phrase ou d'un code source (lorsque l'interpréteur Python lit du code source, il construit d'abord l'arbre syntaxique du code)
- les *arbres binaires de recherche* qui permettent de rechercher ou d'insérer efficacement un élément dans un arbre (voir Thème 5, Chapitre 2)
- l'arbre de Huffmann sur lequel repose la méthode de compression de Huffmann, très utilisée pour compresser des données (textes, images, vidéos; sons, etc.) et que l'on retrouve dans les compressions JPEG, MPEG, MP3, ZIP, etc.
- le DOM (Document Object Model) permet de représenter la structure d'une page Web affichée sous forme d'un arbre, mais aussi de modifier les éléments de la page en modifiant l'arbre (souvent grâce à JavaScript).

## Bilan

- Un arbre est une structure de données abstraites permettant de représenter de manière hierarchique des données.
- Un arbre est formé de *noeuds* reliés entre eux par des branches : on distingue le noeud *racine*, les noeuds internes et les noeuds externes que l'on appelle les *feuilles*. On utilise les termes *fils* et *pères* pour symboliser la hiérarchie entre les noeuds.
- Un *arbre binaire* est un arbre dont tous les noeuds ont au plus deux fils. Cela permet de définir un arbre binaire non vide comme un noeud possédant un *sous-arbre gauche* et un *sous-arbre droit*, éventuellement vides. Ces sous-arbres étant également des arbres binaires (ou des arbres vides), on a définit ainsi les abres binaires de manière récursive, ce qui explique qu'il est naturel d'écrire des algorithmes récursifs pour effectuer des opérations sur les arbres binaires (calcul de la taille, de la hauteur) et pour les parcourir (voir Thème 5, Chapitre 2).
- On a vu dans les activités plusieurs implémentations d'un arbre binaire, mais il en existe d'autres.

---

**Références :**


- Equipe éducative DIU EIL, Université de Nantes.
- Ressource d'accompagnement Eduscol sur les structures de données.
- Livre *Prepabac NSI, Tle*, G. Connan, V. Petrov, G. Rozsavolgyi, L. Signac, éditions HATIER.


---


Germain BECKER, Lycée Mounier, ANGERS




Voir en ligne : info-mounier.fr/terminale_nsi/structures_donnees/arbres