# Numérique et Sciences Informatiques
# Chapitre VII - Structures de données hiérarchiques

Classe de terminale

# I. Généralités

Un arbre est une structure de données hiérarchique naturellement récursive. Cette structure est utilisée pour représenter des données structurées hiérarchiquement.

**Exemples**

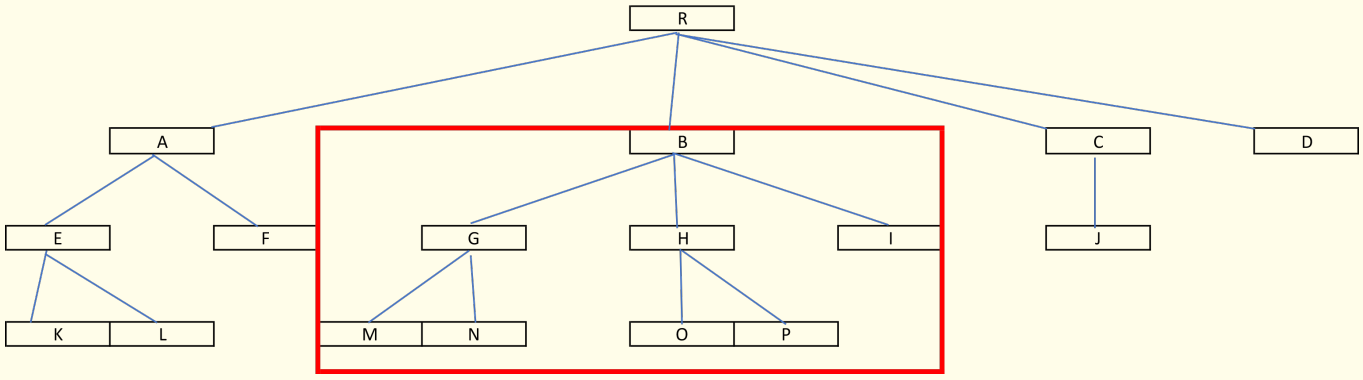
- un système de fichiers (chaque fichier est contenu dans un dossier qui est lui-même contenu dans un autre dossier, etc. qui est contenu dans la racine)
- un site Web (chaque page d'un site peut être atteinte à partir d'une page parente, qui peut elle même être atteinte à partir d'une autre page parente, etc. qui peut être atteinte à partir de la page principale)
- Les grades à l'armée (général de division > général de brigade > colonel > ... > major > ... > soldat)

**Définition**

Un arbre enraciné est une structure de données non linéaire, mais elle peut elle aussi être définie récursivement :

- chaque nœud est accessible depuis un nœud racine.
- chaque nœud est composé d'une valeur et d'une liste de références vers d'autres nœuds avec les contraintes suivantes :
  - chaque nœud a un unique parent.
  - le nœud racine n'a pas de parent

**Exemples**



**Vocabulaire**

Avant d'aller plus loin dans les détails, il s'agit de voir le vocabulaire relatif aux arbres.

Un **nœud** est caractérisé par :

- Une **donnée** (ou étiquette)
- Un nombre fini de **fils**

Une **arête** relie deux nœuds. Une **arête sortante** est une arête qui nous éloigne de la racine. Une **arête entrante** est une arête qui nous approche de 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.
La **racine** d'un arbre est le seul nœud sans père.

**Remarque**

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.

Un **chemin** est une liste de nœuds reliés par des arêtes sortantes.
Les **ancêtres** d'un nœud A sont les nœuds situés sur le chemin reliant la racine à A.
Les **descendants** d'un nœud A sont les noeuds dont A est un ancêtre.
Une **branche** est le chemin le plus court reliant un nœud à la racine.
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.

Par souci de simplicité, on identifiera un nœud à sa donnée (au lieu de l'identifier par sa donnée et l'ensemble de ses fils).

**Exemple**

Dans notre exemple, on a :

- 17 nœuds : A, B, ...
- Le lien entre A et F est une arête.
- Une arête sortante de G mène à N.
- L'arête entrante de I est B.
- Le nœud R est la racine de l'arbre.
- Le nœud B possède trois fils respectivement G, H et I.
- Le père du nœud J est le nœud C.
- Le chemin qui mène du nœud A au nœud L est (A, E, L).
- Les nœuds G et P sont des descendants du nœud B.
- La branche du nœud M est (R, B, G, M).
- La partie entourée en rouge est un sous-arbre de notre arbre.
- Les nœuds K, L, F, M, N, O, P, I, J et D sont des feuilles de l'arbre.
- Le nœud H est une nœud interne.

**Mesures sur les arbres**

La **taille** d'un arbre est le nombre de nœud de l'arbre.
La **profondeur** d'un nœud est le nombre d'arêtes entre lui et la racine. La profondeur de la racine est donc égale à 0.
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 de ce nœud.
L'**arité d'un arbre** est l'arité maximale de ces nœuds.

**Exemple**

Dans notre exemple, on a :

- La taille de l'arbre est 17.
- La profondeur du nœud G est 2.
- La hauteur de l'arbre est 3.
- L'arité du nœud B est 3 alors que l'arité du nœud C est 1 et que l'arité du nœud D est 0.
- L'arité de l'arbre est 3.

## II. Arbres binaires

Un **arbre binaire** est un arbre d'arité 2.

**Définition**

Un arbre binaire est :

- Soit l'arbre vide, noté $\Delta$.
- Soit un triplet (e, g, d) appelé nœud, dans lequel :
  - e est la donnée de la racine de l'arbre.
  - g est le sous-arbre gauche de l'arbre.
  - d est le sous-arbre droit de l'arbre.

**Remarque**

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

**Remarque**

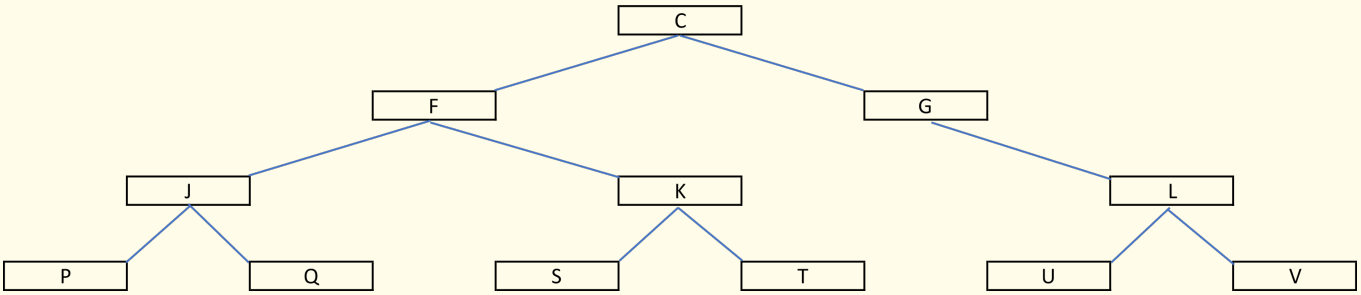
La hauteur d'un arbre binaire vide vaut -1.

**Vocabulaire**

On appelle **fils gauche**, respectivement **fils droit** le sous-arbre gauche, respectivement le sous-arbre droit.
Un arbre binaire **filiforme** (ou **dégénéré**) est un arbre binaire dont tous les nœuds internes n'ont qu'un seul fils.
Un arbre binaire **localement complet** est un arbre binaire dont tous les nœuds internes possèdent exactement deux fils.
Un arbre **complet** est un arbre binaire localement complet dont toutes les feuilles ont la même profondeur.
Un arbre binaire **parfait** est un arbre binaire dans lequel tous les niveaux sont remplis à l'exception éventuelle du dernier, et dans ce cas les feuilles sont alignées à gauche.
Un arbre binaire **équilibré** est un arbre binaire dont les deux fils sont des arbres binaires équilibrés dont les hauteurs diffèrent d'au plus une unité. Autrement dit, pour chaque nœud, la différence des hauteurs du sous-arbre gauche et du sous-arbre droit doit valoir $-1$, 0 ou 1.

**Exemples**

Voici un exemple de représentation d'un arbre binaire :



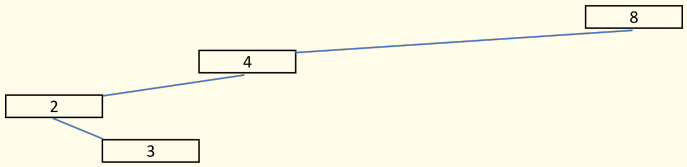

Sa taille est 12.
La prfondeur du nœud K est 2.
La hauteur de l'arbre est 3.
L'arité du nœud G est 1. Celle du nœud J est 2.
L'arité de l'arbre est 2.
On peut aussi écrire :

$$(C, (F, (J, (P, \Delta, \Delta), (Q, \Delta, \Delta)), (K, (S, \Delta, \Delta), (T, \Delta, \Delta))), (G, \Delta, (L, (U, \Delta, \Delta), (V, \Delta, \Delta))))$$

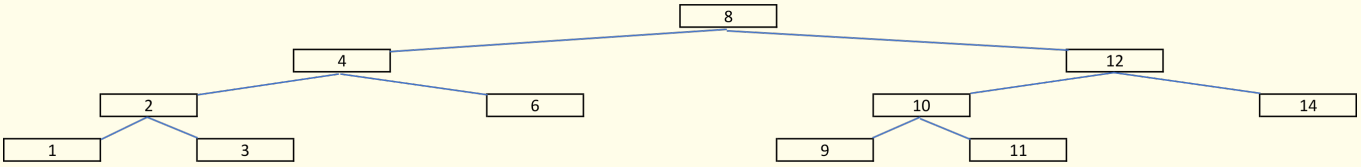
Cet arbre binaire :

- n'est pas filiforme (car par exemple le nœud L possède deux fils : U et V)
- n'est pas localement complet (car le nœud G n'a pas deux fils)
- n'est pas complet car pas localement complet
- n'est pas parfait car le niveau 2 n'est pas rempli
- n'est pas équilibré car le sous-arbre gauche de G a une hauteur égale à $-1$ et le sous-arbre droit de G une hauteur égale à 1. Et $1 - (-1) = 2$.

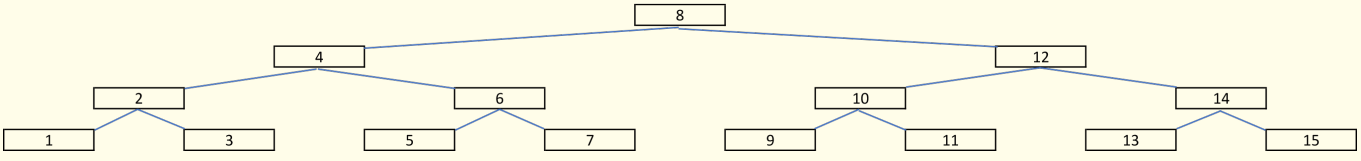
Voici un exemple d'arbre binaire filiforme.




Chacun de ses nœuds internes n'a qu'un fils.
Voici un exemple d'arbre localement complet.



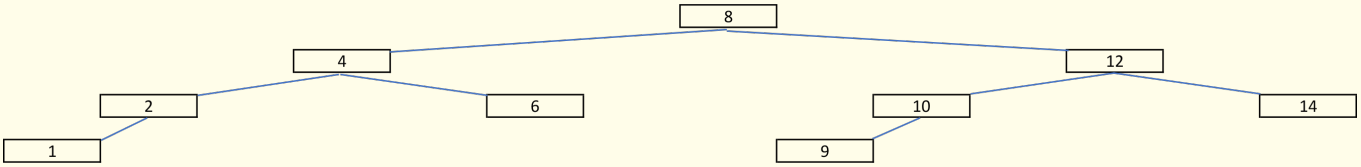

Chacun de ses nœuds internes a deux fils.
Voici un exemple d'arbre complet.



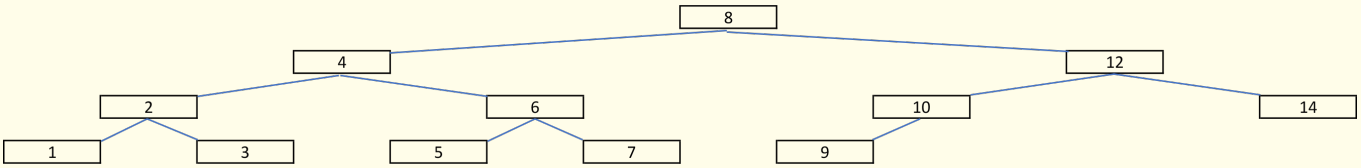

Chacun de ses nœuds internes a deux fils. Ses feuilles sont toutes à même profondeur.

**Exemple - suite**

Voici un exemple d'arbre parfait.




Tous les niveaux de profondeur sont remplis à l'exception du dernier.
Voici un exemple d'arbre équilibré.




Le fils gauche et le fils droit de chaque neud sont des arbres équilibrés de profondeur qui diffèrent d'au plus un niveau.

**Remarque**

Un arbre complet est forcément localement complet, parfait et équilibré.

# III. Quelques algorithmes sur les arbres binaires

## III.1. Calcul de la taille

**Algorithme 1 :** Taille d'un arbre

**si** *l'arbre est vide* **alors**
| taille ←0
**sinon**
| taille ←1 + taille du sous-arbre de gauche + taille du sous-arbre de droite
**fin**

## III.2. Calcul de la hauteur

**Algorithme 2 :** Hauteur d'un arbre

**si** *l'arbre est vide* **alors**
| hauteur ←renvoyer −1
**sinon**
| hauteur ←1 + max(taille du sous-arbre de gauche , taille du sous-arbre de droite)
**fin**

### III.3. Parcourir un arbre de différentes façons

**Algorithme 3 :** Parcours préfixe d'un arbre

Lors d'une visite d'un nœud :

- on traite l'étiquette d'un nœud
- on visite son fils gauche
- on visite son fils droit

**Algorithme 4 :** Parcours infixe d'un arbre

Lors d'une visite d'un nœud :

- on visite son fils gauche
- on traite l'étiquette d'un nœud
- on visite son fils droit

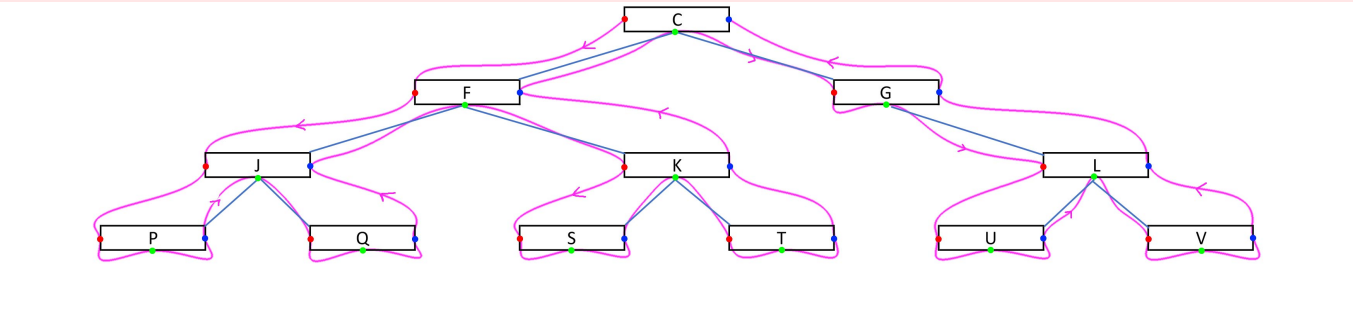
**Algorithme 5 :** Parcours suffixe (ou postfixe) d'un arbre

Lors d'une visite d'un nœud :

- on visite son fils gauche
- on visite son fils droit
- on traite l'étiquette d'un nœud

**Remarque**

On peut résumer ces trois parcours par le schéma suivant :



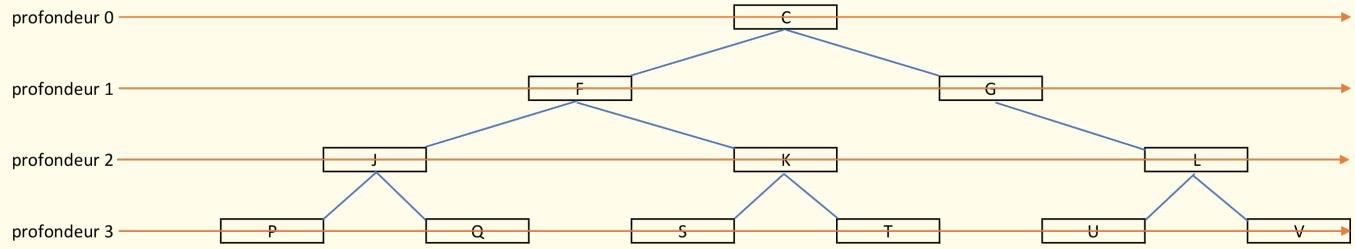

En suivant le parcours rose, on peut voir :

- Le parcours préfixe (points rouge) : C - F - J - P - Q - K - S - T - G - L - U - V
- Le parcours infixe (points verts) : P - J - Q - F - S - K - T - C - G - U - L - V
- Le parcours suffixe (points bleus) : P - Q - J - S - T - K - F - U - V - L - G - C

**Algorithme 6 :** Parcours en largeurs d'abord

- On visite les nœuds de profondeur 0 (soit la racine)
- On visite les nœuds de profondeur 1 de gauche à droite
- On visite les nœuds de profondeur 2 de gauche à droite
- ...

**Exemple**




Le parcours en largeur donne donc :
C - F - G - J - K - L - P - Q - S - T - U - V

## IV. Arbres binaires de recherche

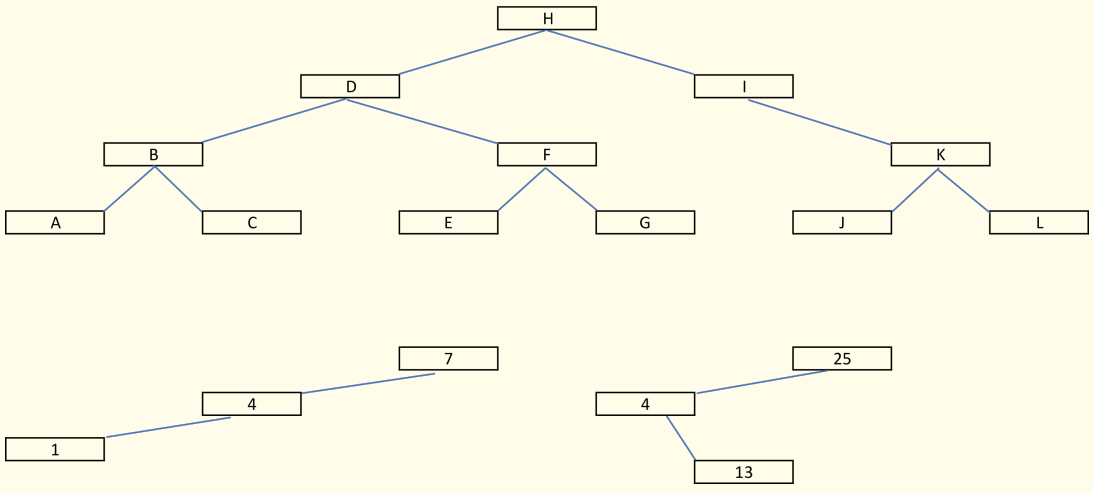
Un arbre binaire de recherche (ABR) est un arbre binaire utilisé pour réaliser des opérations de recherche, d'insertion et de suppression de valeurs le plus efficacement possible. C'est pourquoi les valeurs des étiquettes des nœuds appartiennent à un ensemble ordonné.

**Définition**

Un **arbre binaire de recherche** est un arbre binaire tel que pour tout nœud d'étiquette $e$ :

- les étiquettes de tous les nœuds de son sous-arbre de gauche sont inférieures ou égales à $e$.
- Les étiquettes de tous les nœuds de son sous-arbre de droite sont supérieures à $e$.

**Exemples d'arbres binaire de recherche**




Chaque sous-arbre d'un arbre binaire de recherche est donc un arbre binaire de recherche.
Un arbre binaire de recherche est donc une structure de données récursive.

# V. Algorithmes sur les arbres binaires de recherche

## V.1. Rechercher une clé dans un arbre binaire de recherche

**Algorithme 7 :** Recherche d'une valeur dans un ABR

```
Entrées : valeur, ABR
si l'arbre est vide alors
    renvoyer 'valeur non présente'
sinon
    si le nœud visité a la bonne étiquette alors
        renvoyer 'Trouvé'
    sinon
        si la valeur cherchée est inférieure à l'étiquette de l'ABR alors
            chercher dans le sous-arbre de gauche
        sinon
            chercher dans le sous-arbre de droite
        fin
    fin
fin
```

## V.2. Insérer une clé dans un arbre binaire de recherche

On veut insérer une valeur $v$ dans un arbre binaire de recherche.

**Algorithme 8 :** Insertion d'une valeur dans un ABR

```
Entrées : v, ABR
si l'arbre est vide alors
    renvoyer l'arbre constitué d'une unique feuille v
sinon
    si la valeur v est inférieure ou égale à la racine alors
        renvoyer l'arbre formé de :
            • la racine
            • l'arbre formé du fils gauche dans lequel aura été inséré la valeur v
            • le fils droit
    sinon
        renvoyer l'arbre formé de :
            • la racine
            • le fils gauche
            • l'arbre formé du fils droit dans lequel aura été inséré la valeur v
    fin
fin
```

### V.3. Complexité

**Remarque**

L'intérêt des arbres binaires de recherche est le fait que l'efficacité est la même pour les opérations de recherche, d'ajout d'une étiquette ou de suppression d'une étiquette (pas au programme).

**Remarque**

La recherche dans un arbre équilibré est de coût logarithmique en moyenne et dans le meilleur des cas. On dit que la complexité est en $\mathcal{O}(log_2(n))$.
Dans le pire des cas, le coût est linéaire. On dit que la complexité est en $\mathcal{O}(n)$.

**Démonstration**

En effet, chacun de ces algorithmes nécessite de parcourir l'arbre de sa racine à une feuille en comparant une valeur à l'étiquette d'un nœud. Ainsi il faut au plus autant de comparaisons que la hauteur de l'arbre.

Soit un arbre de taille $n$.
Dans le pire des cas, l'arbre est filiforme et donc sa hauteur est $n-1$. Le coût est donc linéaire.

Dans le meilleur des cas, l'arbre est parfait et donc sa hauteur est $\lfloor log_2(n) \rfloor$. Donc le coût est logarithmique.

En moyenne, on admet que le coût est logarithmique.