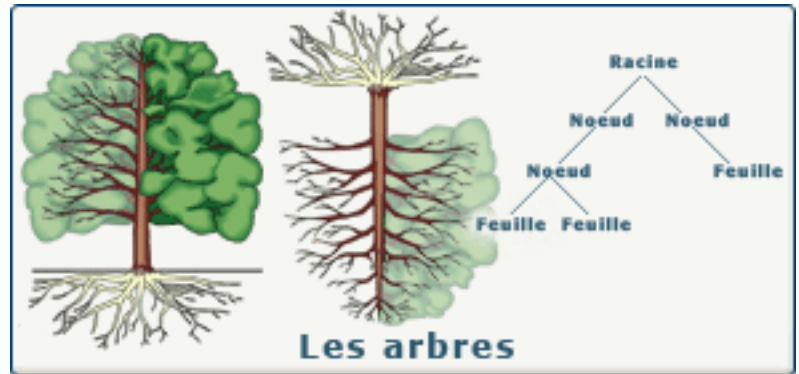


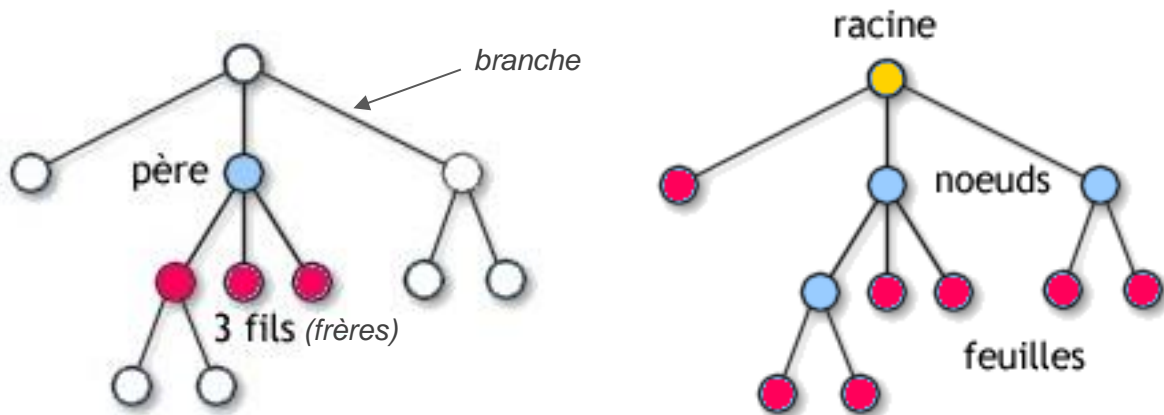


Comme nous venons de le voir avec les listes, piles, files, il n'existe pas seulement cette façon linéaire de stocker des données. Il est également possible de structurer les données hiérarchiquement, avec des arbres. **Leur utilité principale de ranger les données de telle sorte que les recherches soient plus efficaces.**



1. Vocabulaire sur les arbres

Un arbre est une structure de données constituée de **nœuds** et de **branches** qui les relient. Les nœuds peuvent être **père** et avoir des **fils** (d'autres nœuds). Le sommet de l'arbre est appelé **racine**, et un nœud qui ne possède pas d'enfants s'appelle une **feuille**.



Un arbre peut être caractérisé par :

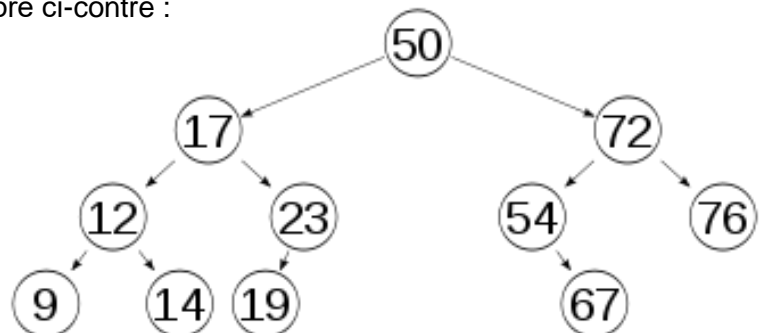
- **Sa taille** : Nombre de nœuds qui le compose
- **Son arité** : Nombre maximal de fils qu'un nœud peut avoir
- **Sa hauteur (profondeur)** : Profondeur à laquelle il faut descendre pour trouver la feuille la plus éloignée de la racine

Vocabulaire sur les nœuds :

- **Nœud racine** : Sommet de l'arbre.
- **Nœud père** : Nœud possédant des fils.
- **Nœud fils** : Nœud possédant un père.
- **Nœuds frères** : Nœuds ayant le même père.
- **Nœud feuille** : Nœuds au plus profond de l'arbre ne possédant pas de fils.
- **Profondeur d'un nœud** : profondeur depuis la racine (racine = 0 par convention en général)
- **Distance** : La distance d'un nœud à un autre est le nombre de branches qui les séparent.
- **Chemin d'un nœud** : Suite de nœuds à parcourir de la racine jusqu'à ce nœud.
- **Clé ou étiquette** : Valeur d'un nœud.

Travail 1 - Quelles sont les caractéristiques de l'arbre ci-contre :

Nœud racine : 50.....
 Nœuds feuilles :9,14,19,67 et 76.....
 Arité : ≥ 2
 Taille : 11.....
 Hauteur de l'arbre : 3.....
 Profondeur du nœud 54 : 2.....
 Distance entre les nœuds 14 et 23 : 3.....
 Citer deux nœuds frères :12 et 23.....



Nous allons nous intéresser par la suite uniquement aux arbres d'arité 2 que l'on appelle arbre binaire

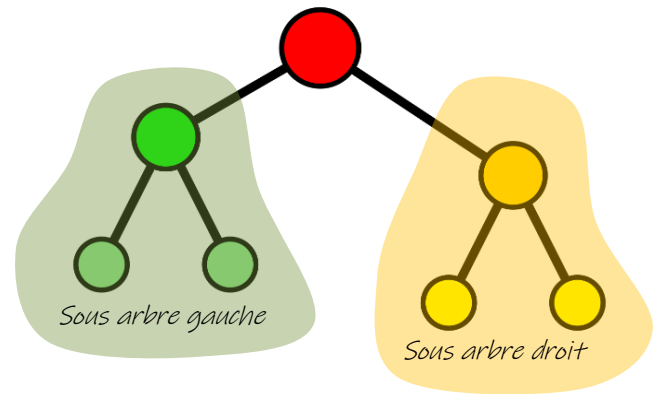
2. Les arbres binaires

2.1 Définition

Dans un arbre binaire, chaque nœud possède au plus deux fils, habituellement appelés « fils gauche » et « fils droit ».

On définit généralement à partir du nœud racine 2 sous arbres:

- Le sous arbre gauche
- Le sous arbre droit



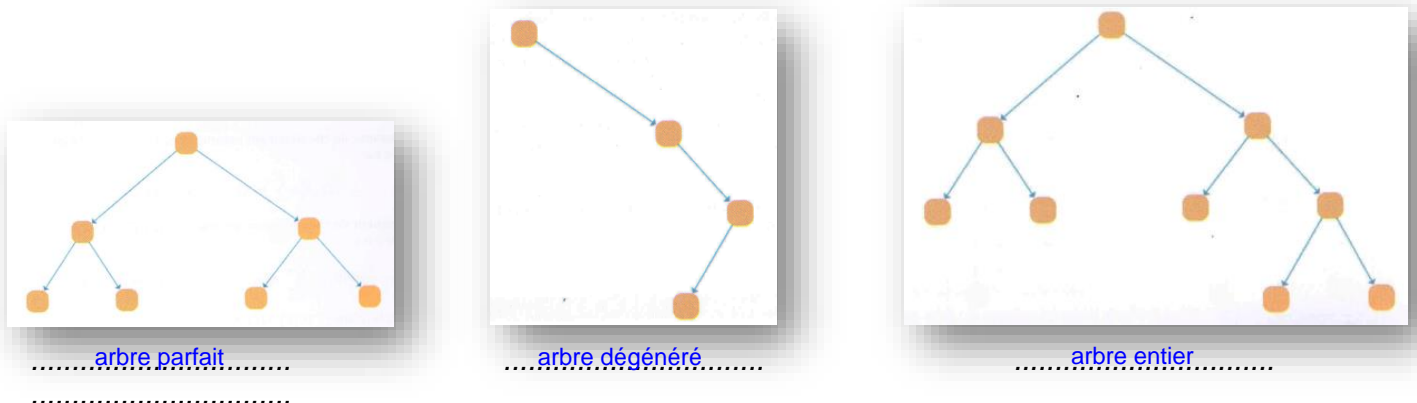
2.2 Cas particuliers

Arbre dégénéré : Les nœuds ne possèdent qu'un fils ou pas du tout.

Arbre entier : Chaque nœud possède soit 2 fils, soit aucun.

Arbre parfait : Arbre entier (Chaque nœud possède soit 2 fils, soit aucun), mais toutes les feuilles sont à la même profondeur (niveau hiérarchique le plus bas)

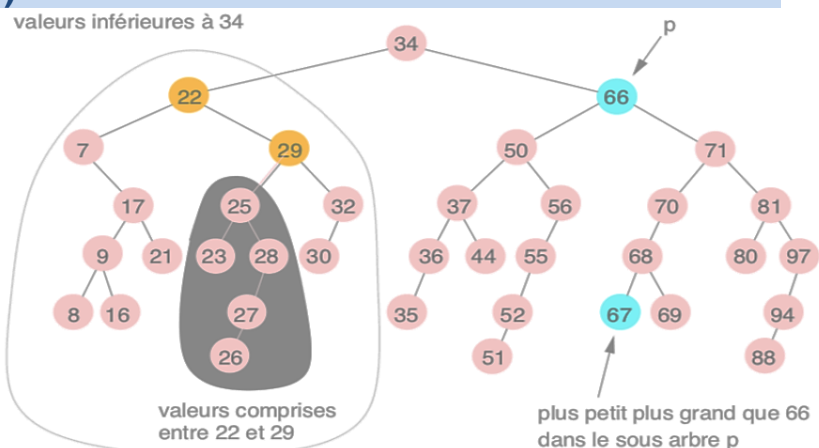
Nombre de nœuds de l'arbre = $2^h - 1$ avec h la hauteur de l'arbre



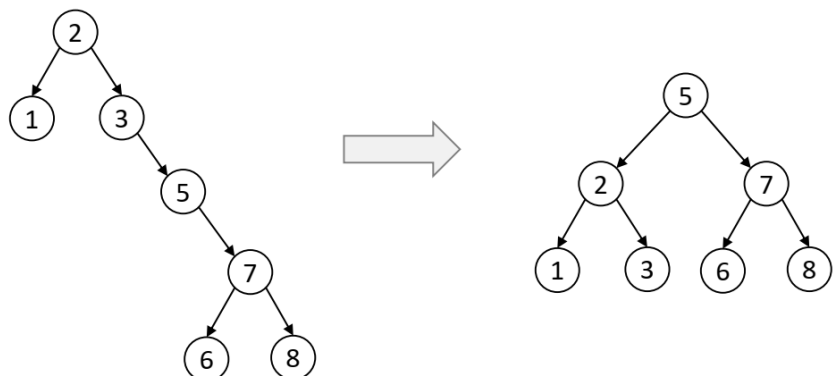
3. Arbre Binaire de Recherche (ABR)

3.1 Arbre binaire de recherche

Un arbre binaire de recherche (ABR) est un arbre binaire dans lequel chaque nœud est étiqueté par une valeur, telle que chaque nœud du sous-arbre gauche ait une valeur inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une valeur supérieure (selon la mise en œuvre de l'ABR, on pourra interdire ou non des valeurs de nœuds égales).



Pour pouvoir faire des recherches efficaces d'éléments (en $O(\log n)$), il faut rééquilibrer l'arbre. Soit à chaque suppression d'élément, ou de temps en temps.

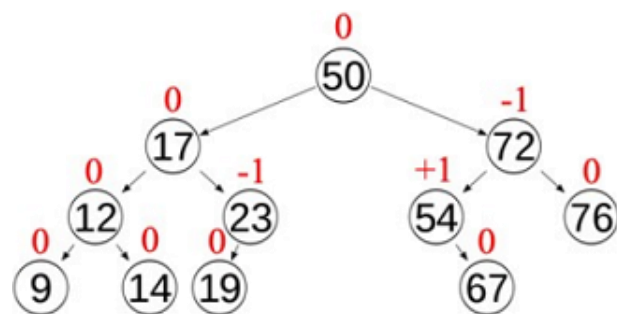


3.2 Arbre AVL

Les arbres AVL ont été historiquement les premiers arbres binaires de recherche automatiquement équilibrés. Dans un arbre AVL, les hauteurs des deux sous-arbres d'un même nœud diffèrent au plus de un. La recherche, l'insertion et la suppression sont toutes en $O(\log(n))$ dans le pire des cas.

La dénomination « arbre AVL » provient des noms de ses deux inventeurs russes, Georgy Maximovich Adelson-Velsky et Evgenii Mikhailovich Landis.

Le facteur d'équilibrage d'un nœud est la différence entre la hauteur de son sous-arbre droit et celle de son sous-arbre gauche. Un nœud/arbre dont le facteur d'équilibrage est +1, 0, ou -1 est considéré comme équilibré. Un nœud avec tout autre facteur est considéré comme déséquilibré et requiert un rééquilibrage.

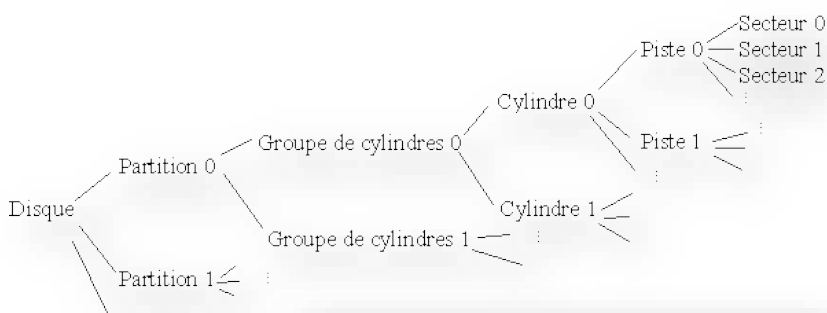


4. Applications des arbres binaires de recherche

4.1 Stockage et recherche de données

Utilisés dans beaucoup les applications de recherche où les données sont entrées/sorties en permanence.

Les arbres B (B-tree) qui sont des arbres équilibrés, sont mis en œuvre dans les mécanismes de gestion de bases de données, et systèmes de fichiers (NTFS, EXT, ...). Ils stockent les données sous une forme triée et permettent une exécution des opérations d'insertion et de suppression en temps toujours logarithmique.

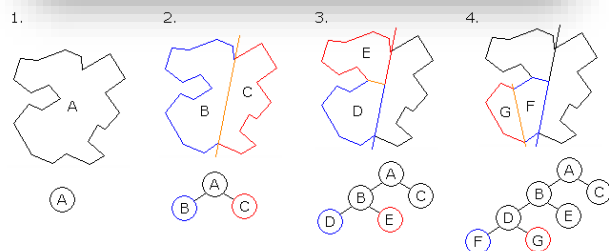


4.2 Les jeux vidéo

Utilisés dans presque tous les jeux vidéo 3D pour déterminer quels objets doivent être rendus, par des moteurs 3D tels que Doom, Quake, GoldSource, ...

Le principe est de subdiviser récursivement un espace, une scène de jeu, en deux. Cela permet une représentation des objets dans l'espace sous la forme d'une structure de données arborescente connue sous le nom d'arbre BSP.

La structure d'un arbre BSP donne rapidement des informations spatiales sur les objets dans une scène qui sont utiles pour le rendu, comme pour la gestion des interactions.



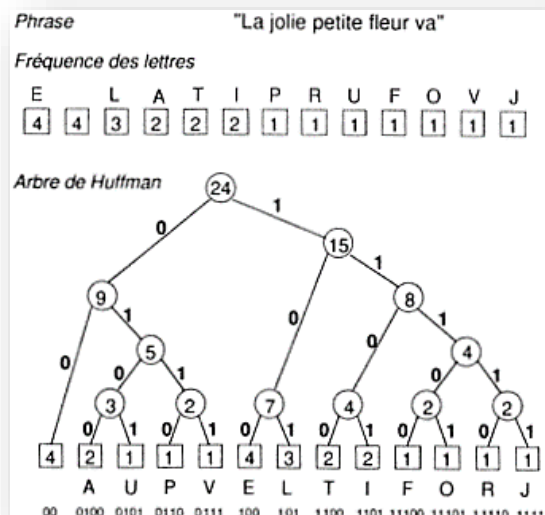
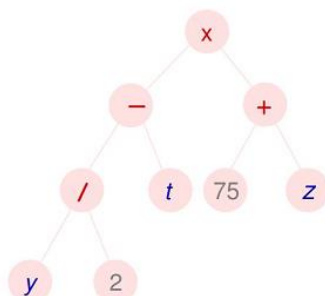
4.3 La compression des données

Utilisés dans les algorithmes de compression, tels que ceux utilisés par les .jpeg et .mp3, l'algorithme de compression sans perte de Huffman.

4.4 Analyse d'expressions mathématiques

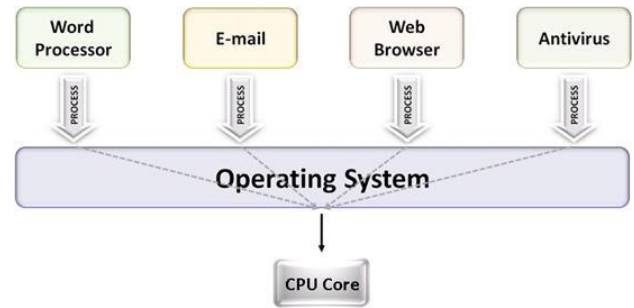
Utilisés par des compilateurs et calculatrices pour analyser des expressions mathématiques.

$$(y/2 - t) \times (75 + z)$$



4.5 Systèmes d'exploitation (ordonnancement des tâches processeurs)

Utilisés pour mettre en œuvre une priorité efficace (File d'attente prioritaire), qui à son tour est utilisée pour programmer des processus dans de nombreux systèmes d'exploitation, assurer la qualité du service dans les routeurs, ...



5. Implémentation des arbres binaires

Le Type Abstrait de Données arbre (binaire) peut être implémenté de différentes façons en informatique. Mais dans tous les cas on retrouve ces opérations (primitives) de base :

Opérations arbre binaire:

- **Constructeur : Arbre()**
Postconditions : L'arbre créé est un arbre vide
- **Procédure vider ()**
Postcondition : L'arbre ne contient plus aucuns éléments
- **Fonction estVide ()** : renvoie un booléen
Résultat : retourne vrai si l'arbre est vide, faux sinon
- **Procédure insérerElement (e)**
Postcondition : Si e n'existe pas dans l'arbre, alors un nouveau nœud contenant e est inséré.
Si e existe déjà dans l'arbre, alors l'arbre est inchangé
- **Procédure supprimerElement (e)** : renvoie tout type
Postcondition : l'élément e est recherché et supprimé de l'arbre. L'arbre reste inchangé si l'élément e n'est pas présent. Les propriétés de l'arbre sont conservées.
- **Fonction hauteur ()** : renvoie un entier
Résultat : retourne la hauteur de l'arbre (profondeur de sa plus longue branche), -1 s'il est vide
- **Fonction rechercherElement (e)** : renvoie un tuple (booléen, Arbre)
Résultat : un tuple indiquant si l'élément e est dans l'arbre (True ou False), et le nœud-arbre de l'élément (None si absent)

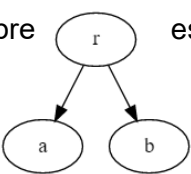
5.1 Implémentation d'un arbre par un tableau dynamique (list python)

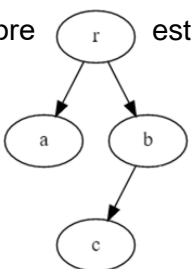
Une possibilité est de représenter chaque nœud par une liste composée de trois éléments : l'étiquette, deux listes représentant les sous arbres gauche (fils gauche), et droit (fils droit)

[étiquette, fils gauche, fils droit]

Exemples :

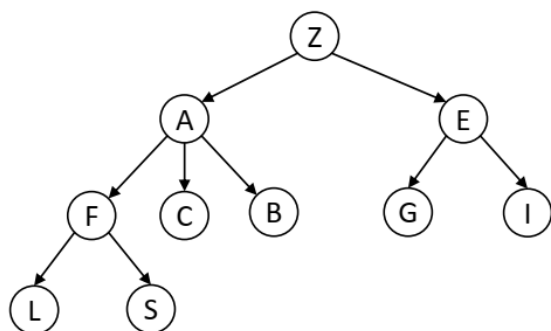
- L'arbre  est implémenté par la liste : ['r', [], []]

- L'arbre  est implémenté par la liste : ['r', ['a', [], []], ['b', [], []]]

- L'arbre  est implémenté par la liste : ['r', ['a', [], []], ['b', ['c', [], []], []]]

5.2 Implémentation d'un arbre par un tableau 2D

Exemple : Le nœud de la ligne a comme fils le nœud de la colonne



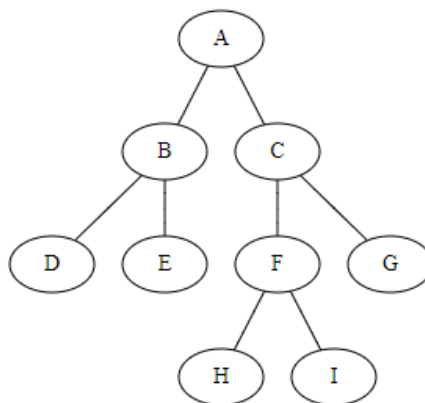
	A	B	C	E	F	G	I	L	S	Z
A	0	1	1	0	1	0	0	0	0	0
B	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	1	1	0	0	0
F	0	0	0	0	0	0	0	1	1	0
G	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0
Z	1	0	0	1	0	0	0	0	0	0

5.3 Implémentation par un dictionnaire python

Une autre solution pour représenter un arbre est d'utiliser un dictionnaire, l'arbre ci-contre serait encoder :

```

arbre = { 'A': ['B', 'C'],
          'B': ['D', 'E'],
          'C': ['F', 'G'],
          'D': [ None, None ],
          'E': [ None, None ],
          'F': ['H', 'I'],
          'G': [ None, None ],
          'H': [ None, None ],
          'I': [ None, None ] }
  
```



Remarque : Lorsqu'il manque un enfant (ou 2) à un nœud, on considère que l'enfant manquant est None

5.4 Implémentation d'un arbre par un classe en POO

Une solution est d'implémenter un arbre binaire à partir d'une classe Arbre modélisant un nœud seul. Cette classe comporte 3 attributs, voici le code :

```

class Arbre :
    def __init__(self, info=None, fg=None, fd=None)
        self.info = info
        self.fg = fg
        self.fd = fd
    .....
  
```

Un arbre vide est représenté par l'attribut info à None.

Exemple : arbre1 = Arbre() # arbre vide
 arbre2 = Arbre("r") # arbre avec un nœud (la racine) ->



Il reste ensuite à implémenter les méthodes **insérerElement**, **supprimerElement**, ...

A retenir

Arbre binaire	Chaque nœud admet au plus deux fils : un fils gauche et un fils droit
Arbre binaire de recherche (ABR)	Arbre étiqueté par des nombres où : <ul style="list-style-type: none"> La racine est supérieure ou égale à tous les nœuds du sous arbre gauche La racine est inférieure à tous les nœuds du sous arbre droit Les deux sous arbre de la racine sont eux-mêmes des ABR
Arbre AVL	Arbre binaire de recherche automatiquement équilibré. Les hauteurs des sous arbres gauche et droit diffèrent d'au plus 1. Recherche en $O(\log(n))$