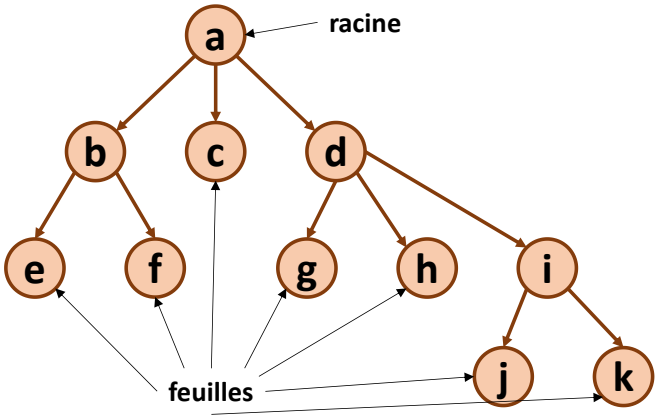
Structures de données hiérarchiques : arbres binaires

I./ Généralités sur les arbres binaires :

Un *arbre* est une *structure de données hiérarchique* constituée de *nœuds*. Chaque nœud peut avoir un ou plusieurs *fils*. Le sommet de la structure est un nœud appelé « *racine de l'arbre* ». Les nœuds n'ayant pas de fils sont les *feuilles*. Par convention, les arbres en tant que structures de données poussent vers le bas :



Les nœuds qui ne sont pas des feuilles, ni la racine de l'arbre, sont appelés « *nœuds internes* ». Une *branche* est une suite de nœuds reliant la racine de l'arbre à l'une de ses feuilles. Un arbre a donc autant de branches que de feuilles. Dans notre exemple, **a** est la racine, **b**, **d** et **i** sont des nœuds internes, **e**, **f**, **c**, **g**, **h**, **j** et **k** sont des feuilles.

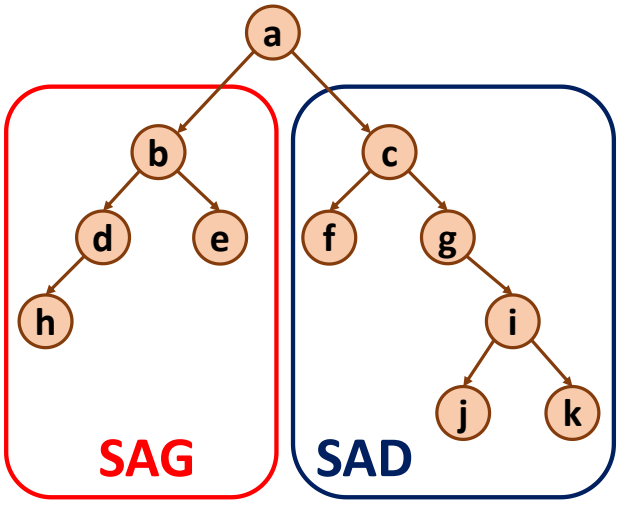
La *taille* d'un arbre est le nombre de nœuds qui le compose. Dans notre exemple, la taille de l'arbre est de 11.

L'*arité* d'un arbre est le nombre maximal de fils qu'un nœud peut avoir. Dans notre exemple, l'arité de l'arbre est de 3. (On peut aussi définir l'arité d'un nœud particulier).

La *hauteur d'un nœud* est le nombre minimal d'arêtes qu'il faut parcourir pour rejoindre ce nœud en partant de la racine plus un. La hauteur de la racine de l'arbre est de 1. La hauteur de l'arbre vide est 0. Dans notre exemple, la hauteur du nœud **i** est de 3.

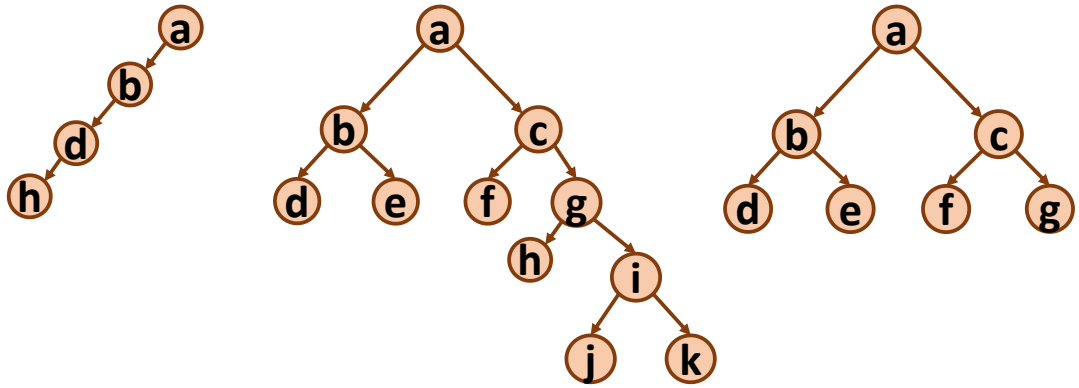
La *hauteur de l'arbre* est égale à celle du nœud qui a la plus grande hauteur. Dans notre cas, la hauteur de l'arbre est de 4.

Un arbre binaire est un arbre particulier dont les nœuds peuvent avoir un maximum de 2 fils. Un arbre binaire est donc un arbre d'arité égale à 2. Un nœud particulier peut donc avoir 0 fils, 1 fils ou 2 fils. On différencie les fils « gauches » des fils « droits ».



A partir du nœud racine, on peut définir deux sous-arbres disjoints, le sous-arbre gauche (SAG) et le sous-arbre droit (SAD).

Arbres binaires particuliers :



- *Arbre binaire dégénéré* ou *arbre binaire filiforme* : c'est un arbre binaire dont les nœuds ont 1 ou 0 fils. Ce type d'arbre a donc une seule feuille et est constitué d'une seule branche. (Cf. arbre de gauche sur le schéma ci-dessus).
- *Arbre binaire localement complet* : c'est un arbre binaire dont les nœuds ont 2 ou 0 fils. (Cf. arbre du milieu sur le schéma ci-dessus).
- *Arbre binaire complet* : c'est un arbre binaire localement complet dont les hauteurs de toutes les feuilles sont égales à la hauteur de l'arbre. On dit aussi que toutes les feuilles sont au niveau hiérarchique le plus bas. (Cf. arbre de droite sur le schéma ci-dessus).
- *Arbre binaire vide* : c'est un arbre binaire de taille 0. Il ne contient donc aucun nœud. Attention à ne pas confondre l'arbre vide avec l'arbre composé uniquement de son nœud racine, qui lui a une taille égale à 1.

II./ Une première implémentation des arbres binaires en Python :

Comme nous l'avons vu précédemment, un arbre binaire particulier peut être considéré comme étant un nœud racine muni d'un sous-arbre gauche et d'un sous-arbre droit. On peut par exemple utiliser une liste de listes pour représenter un arbre binaire, de la façon suivante :

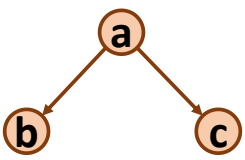
```
[ 'etiquette du noeud racine', [sous-arbre gauche], [sous-arbre droit]]
```

L'arbre vide sera représenté par une liste vide.

Soit par exemple un arbre composé d'un unique nœud, le nœud racine :

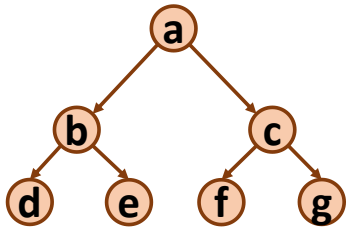
Le SAG de ce nœud '**a**' est l'arbre vide. De même pour le SAD de ce nœud. On peut donc représenter cet arbre binaire de la façon suivante : `[ 'a', [], [] ]`

Soit l'arbre binaire suivant : le nœud racine '**a**' possède un SAG composé d'un nœud '**b**' ayant un SAG et un SAD vide, et un SAD composé d'un nœud '**c**' ayant un SAG et un SAD vide :



Sa représentation sera : `[ 'a', [ 'b', [], [] ], [ 'c', [], [] ] ]`

Exercice : quelle serait la représentation sous forme de liste de listes de l'arbre binaire suivant ?



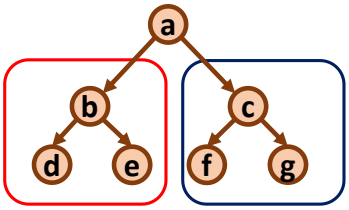
Pour implémenter la structure de données « arbre binaire », il faut disposer des fonctions suivantes :

- CREER_ARBRE_VIDE() : qui va renvoyer un arbre vide : **`[]`**
- CREER_ARBRE_FEUILLE(étiquette) : qui va renvoyer un arbre composé d'un seul nœud muni d'une étiquette et ayant un SAG et un SAD vides : **`[ étiquette, [], [] ]`**
- CREER_ARBRE(étiquette, SAG, SAD) : qui va renvoyer l'arbre composé d'un nœud portant une étiquette et dont les SAG et SAD sont donnés en argument : **`[ étiquette, SAG, SAD ]`**
- EST_VIDE(arbre) : qui renvoie un booléen indiquant si l'arbre passé en argument est vide.
- RACINE(arbre) : qui renvoie l'étiquette du nœud racine de l'arbre passé en argument.
- GAUCHE(arbre) : qui renvoie le SAG de l'arbre passé en argument.
- DROITE(arbre) : qui renvoie le SAD de l'arbre passé en argument.

Tapez et exécutez le code suivant :

```
def creer_arbre_vide():
    return []

def creer_arbre_feuille(etiquette):
    return [ etiquette, [], [] ]

def creer_arbre(etiquette, SAG, SAD):
    return [ etiquette, SAG, SAD ]

def racine(arbre):
    return arbre[0]

def gauche(arbre):
    return arbre[1]

def droit(arbre):
    return arbre[2]

def est_vide(arbre):
    return arbre == []

def hauteur(arbre):
    if arbre != []:
        return 1 + max(hauteur(gauche(arbre)), hauteur(droit(arbre)))
    else:
        return 0

def taille(arbre):
    if arbre != []:
        return 1 + taille(gauche(arbre)) + taille(droit(arbre))
    else:
        return 0
```

Soit l'arbre binaire suivant :



Son SAG est un arbre ayant comme racine le nœud '**b**' et comme SAG et SAD les feuilles '**d**' et '**e**'. On peut donc le créer de la façon suivante :

```
SAG = creer_arbre('B', creer_arbre_feuille('D'), creer_arbre_feuille('E'))
```

De la même façon, on peut créer son SAD de la façon suivante :

```
SAD = creer_arbre('C', creer_arbre_feuille('F'), creer_arbre_feuille('G'))
```

On crée alors l'arbre souhaité à l'aide de : `arbre = creer_arbre('A', SAG, SAD)`

On pouvait aussi directement créer cet arbre à l'aide de la commande suivante, mais cela est nettement moins lisible !

```
arbre=creer_arbre('A',creer_arbre('B',creer_arbre_feuille('D'),creer_arbre_feuill
e('E')),creer_arbre('C',creer_arbre_feuille('F'),creer_arbre_feuille('G')))
```

Exercice : quelle est la particularité des deux fonctions **`hauteur`** et **`taille`** ? En les appliquant à l'exemple d'arbre binaire précédent, étudiez leur fonctionnement.

III./ Réalisation d'une classe « Arbre_binaire » :

Pour les raisons évoquées dans le chapitre précédent (Files et Piles), on voudrait maintenant réaliser une classe « **`Arbre_binaire`** ». Voici le début de l'implémentation de cette classe :

```
class Arbre_binaire:


    def __init__(self, etiquette = None, gauche = None, droit = None):
        self.__etiquette = etiquette
        self.__gauche = gauche
        self.__droit = droit
        return None
```

Les « **`None`** » correspondent aux valeurs par défaut. Pour créer un arbre vide, on pourra donc juste faire :

```
arbre_vide = Arbre_binaire()
```

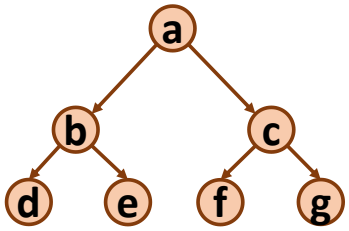
Un arbre vide aura donc son étiquette ainsi que ses SAG et SAD à une valeur **`None`**.

Pour un arbre composé d'un unique nœud racine, on pourra faire :

```
arbre_feuille = Arbre_binaire('étiquette')
```

Un arbre composé uniquement d'un nœud racine aura donc juste une étiquette, ses SAG et SAD auront une valeur **`None`**.

Pour pouvoir créer l'arbre suivant :



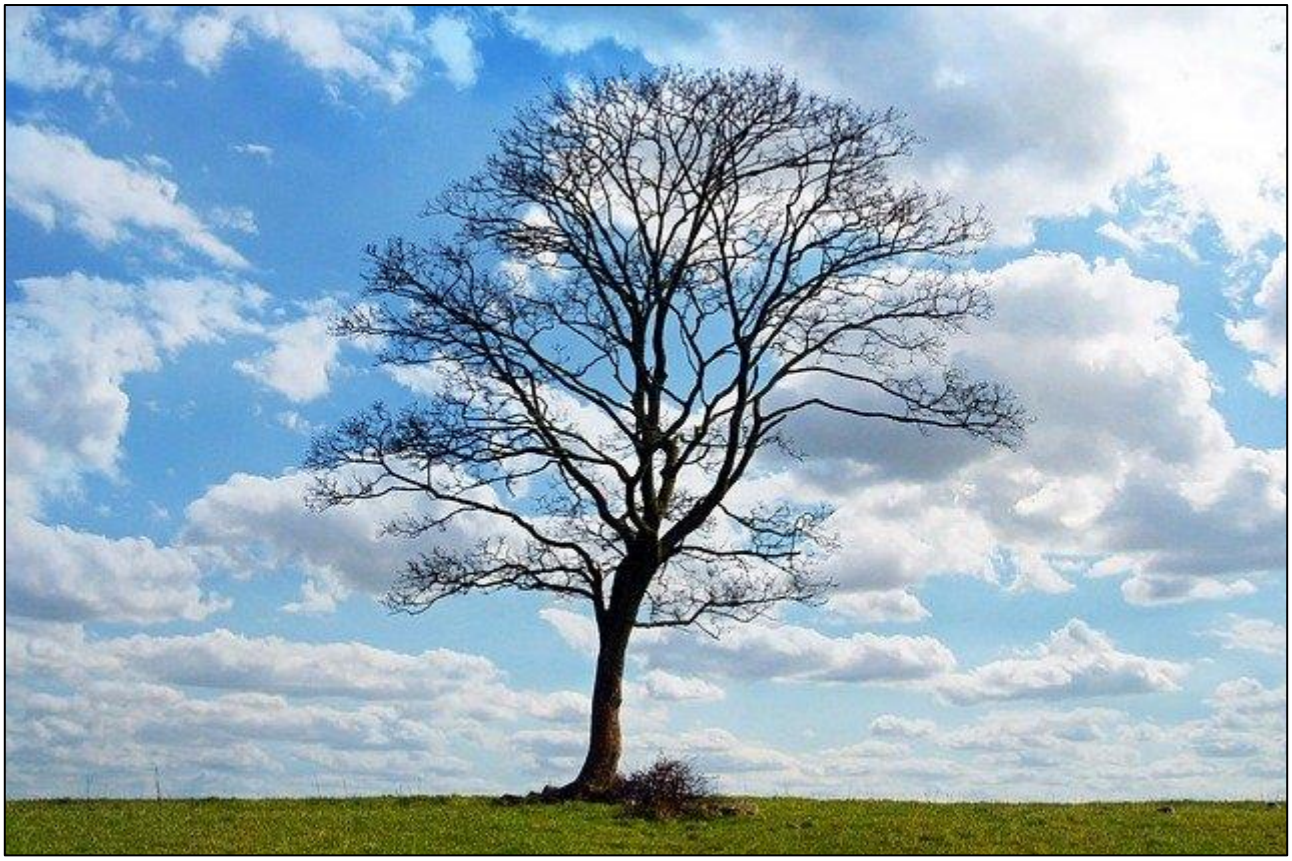
On pourra utiliser la commande :

```
arbre=Arbre_binaire('A',Arbre_binaire('B',Arbre_binaire('D'),Arbre_binaire('E')),
Arbre_binaire('C',Arbre_binaire('F'),Arbre_binaire('G')))
```

Exercice : complétez la classe « **Arbre_binaire** » en rajoutant les méthodes publiques suivantes :

- Les getters et setters permettant de récupérer et de modifier les attributs **__etiquette**, **__gauche** et **__droit**.
- **est_vide()**, qui renverra un booléen **True** si l'arbre est vide, **False** sinon.
- **hauteur()** qui renverra la hauteur de l'arbre binaire.
- **taille()** qui renverra la taille de l'arbre binaire.

Nous verrons l'utilisation des arbres binaires dans un prochain chapitre.



2025
Bruno PEYROT