Introduction :

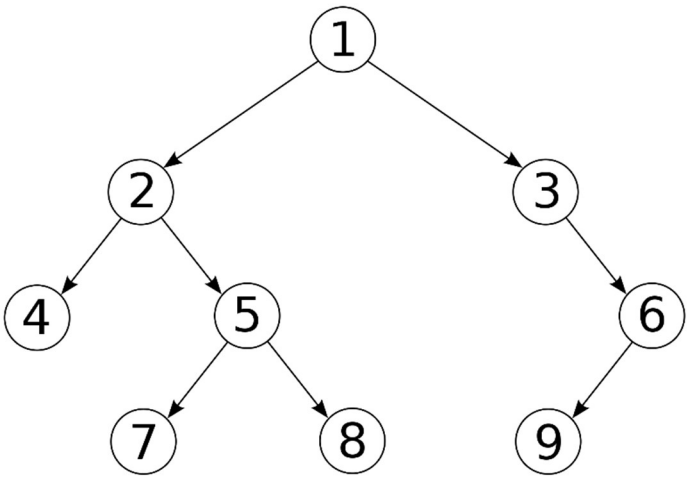
Un arbre binaire est une structure de données qui peut se représenter sous la **forme d'une hiérarchie** dont chaque élément est appelé **nœud**, le nœud initial étant appelé **racine**. Dans un arbre binaire, chaque élément possède au plus deux éléments **fils** au niveau inférieur, habituellement appelés **gauche et droit.** Du point de vue de ces éléments fils, l'élément dont ils sont issus au niveau supérieur est appelé **père**.

Au niveau le plus élevé, niveau 0, il y a un **nœud racine**. Au niveau directement inférieur, il y a au plus deux nœuds fils. En continuant à descendre aux niveaux inférieurs, on peut en avoir quatre, puis huit, seize, etc. c'est-à-dire **la suite des puissances de deux**. Un nœud n'ayant aucun fils est appelé **feuille.**

Le niveau d'un nœud, autrement dit la distance entre ce nœud et la racine, est appelé profondeur. La hauteur de l'arbre est la profondeur maximale d'un nœud.

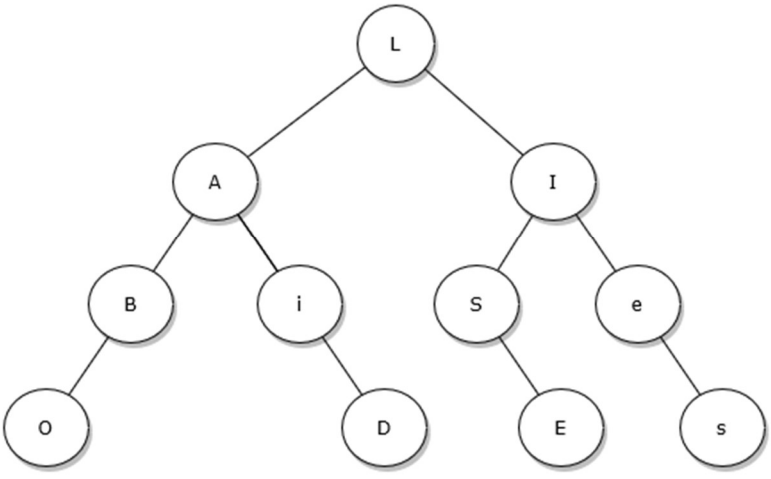
Représentation :

Voici un exemple simple d'arbre binaire :



Exercice 1 :

On dispose de cet arbre binaire :



Le but est de créer plusieurs fonctions afin d'effectuer des tâches simples par rapport à l'arbre binaire donné :

- Une fonction Arbre_vide(arbre) qui retourne vrai si l'arbre est vide.
- Une fonction Enfants(arbre, nœud) qui retourne les enfants d'un nœud sous la forme d'un uplet (fg,fd).
- Deux fonctions Fils_gauche(arbre, nœud) qui retournent le fils gauche d'un nœud et son homologue Fils_droit(arbre, nœud) s'ils existent.
- Une fonction Racine(arbre, nœud) qui retourne vrai si le nœud est la racine de l'arbre.
- Une fonction Est_feuille(arbre, nœud) qui retourne vrai si le nœud est une feuille.
- Une fonction Parent(arbre, nœud) qui retourne le parent du nœud ou False si c'est la racine
- Une fonction Frere(arbre, nœud) qui retourne vrai si le nœud a un frère gauche ou droit
- Testez plusieurs nœuds avec une boucle pour et une liste de nœud=['I','L','O','S']

1. Création de l'arbre, insertion de nœuds et vérification :

```
def Creation_arbre(r,profondeur):
    ''' r : la racine (str ou int). la profondeur de l'arbre (int)'''
    Arbre = [r]+[None for i in range(2**(profondeur+1)-2)]
    return  Arbre

def Insertion_noeud(arbre,n,fg,fd):
    '''Insère les noeuds et leurs enfants dans l'arbre'''
    indice = arbre.index(n)
    arbre[2*indice+1] = fg
    arbre[2*indice+2] = fd

def Arbre_vide(arbre):
    '''La fonction retourne vrai si l'arbre est vide'''
    condition = False
    if len(arbre) == 0:
        condition = True
    return condition
```

2. Recherche d'enfants (fils gauche et fils droit) :

```
def Enfants(arbre,noeud):
    '''La fonction retourne les enfants d'un nœud sous la forme d'un uplet
(fg,fd)'''
    pos = arbre.index(noeud)
    fg = arbre[2*pos+1]
    fd = arbre[2*pos+2]
    return (fg, fd)

def Fils_gauche(arbre,noeud):
    '''La fonction retourne le fils gauche d'un nœud'''
    pos = arbre.index(noeud)
    fgauche = arbre[2*pos+1]
    return fgauche

def Fils_droit(arbre,noeud):
    '''La fonction retourne le fils droit d'un nœud'''
```

```
    pos = arbre.index(noeud)
    fdroit = arbre[2*pos+2]
    return fdroit
```

3. Racine et feuille :

```
def Racine(arbre,noeud):
    '''La fonction retourne vrai si le nœud est la racine de l'arbre'''
    racine_cond = False
    if arbre[0] == noeud:
        racine_cond = True
    return racine_cond

def Est_feuille(arbre,noeud):
    '''La fonction retourne vrai si le nœud est une feuille'''
    feuille_cond = False
    if Enfants(arbre, noeud)[0] == None and Enfants(arbre, noeud)[1] ==
None:
        feuille_cond = True
    return feuille_cond
```

4. Parents et frères :

```
def Parent(arbre,noeud):
    '''La fonction retourne le parent du nœud ou False si c'est la
racine'''
    parent_noeud = False
    for element in arbre :
        if Enfants(arbre,element)[0] == noeud or Enfants(arbre,element)[1]
== noeud:
            parent_noeud = element
    return parent_noeud

def Frere(arbre,noeud):
    '''
    La fonction retourne vrai si le nœud a un frère gauche ou droit.
("frere_cond")
    Ainsi que le frère en question ! ("frere_element")
    '''
    frere_cond = False
    papa = Parent(arbre,noeud)
    if papa == False :
        papa = arbre[0]
    node_childrens = Enfants(arbre,papa)
    frere_element = None
    if node_childrens[0] != None and node_childrens[1] != None :
        frere_cond = True
        if node_childrens[0] == noeud :
            frere_element = node_childrens[1]
        else :
            frere_element = node_childrens[0]
    return (frere_cond,frere_element)
```

5. Appel des fonctions :

```
# Création de l'arbre :
arbre = Creation_arbre('L',4)

# Ajout des noeuds par niveau de gauche à droite :
Insertion_noeud(arbre,'L','A','I')
```

```
Insertion_noeud(arbre,'A','B','i')
Insertion_noeud(arbre,'I','S','e')
Insertion_noeud(arbre,'B','O',None)
Insertion_noeud(arbre,'i',None,'D')
Insertion_noeud(arbre,'S',None,'E')
Insertion_noeud(arbre,'e',None,'s')

# Pour vérifier les données initiales :
print('Taille du tableau',len(arbre))
print('Arbre=',arbre)

# Vérification du remplissage de l'arbre :
print('Arbre vide :',Arbre_vide(arbre))

# Liste des nœuds testés :
nodes = ['I','L','O','S']

# Boucle des différents tests :
for i in nodes :
    print("Noeud :",i)
    print(Enfants(arbre,i),'sont les enfants de',i)
    print(Fils_gauche(arbre,i),'est le fils gauche')
    print(Fils_droit(arbre,i),'est le fils droit')
    print(i,'est racine :',Racine(arbre,i))
    print(i,'est une feuille :',Est_feuille(arbre,i))
    print(Parent(arbre,i),'est le parent de',i)
    print(i,'a un frère :',Frere(arbre,i)[0]," il s'agit de
:",Frere(arbre,i)[1])
```

Nous avons donc créé les différentes fonctions, il ne nous reste plus qu'à créer l'arbre et y insérer les nœuds, afin de procéder aux différentes sélections !

6. Résultats dans la console :

```
"""
Résultat dans la console :

Taille du tableau 31
Arbre= ['L', 'A', 'I', 'B', 'i', 'S', 'e', 'O', None, None, 'D', None,
'E', None, 's', None, None, None, None, None, None, None, None, None,
None, None, None, None, None, None]
Arbre vide : False
Noeud : I
('S', 'e') sont les enfants de I
S est le fils gauche
e est le fils droit
I est racine : False
I est une feuille : False
L est le parent de I
I a un frère : True  il s'agit de : A
Noeud : L
('A', 'I') sont les enfants de L
A est le fils gauche
I est le fils droit
L est racine : True
L est une feuille : False
False est le parent de L
L a un frère : True  il s'agit de : A
```

```
23 Noeud : O
24 (None, None) sont les enfants de O
25 None est le fils gauche
26 None est le fils droit
27 O est racine : False
28 O est une feuille : True
29 B est le parent de O
30 O a un frère : False  il s'agit de : None
31 Noeud : S
32 (None, 'E') sont les enfants de S
33 None est le fils gauche
34 E est le fils droit
35 S est racine : False
36 S est une feuille : False
37 I est le parent de S
38 S a un frère : True  il s'agit de : e
39 """
```

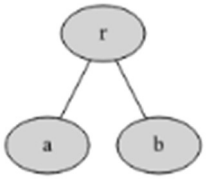
Grâce à l'exécution du code, on obtient donc aisément des informations très utiles comme :

- **Représentation de l'arbre sous forme d'un tableau** : ['L', 'A', 'I', 'B', 'i', 'S', 'e', 'O', None, None, 'D', None, 'E', None, 's', None, None, None, None, None, None, None, None, None, None, None, None, None, None, None, None]
- **Taille du tableau** : 31
- **Ainsi, que les enfants/parents/frères des différents nœuds sélectionnés.**

Exercice 2 :

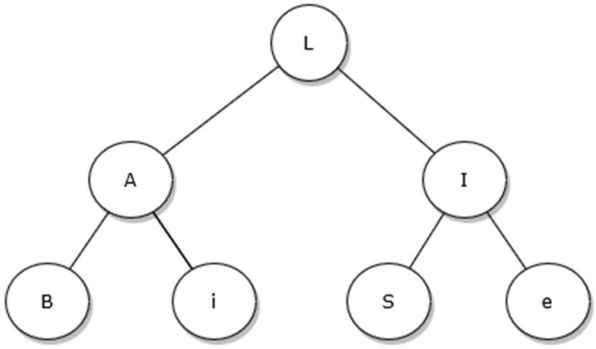
Comme vous l'avez sans doute constaté, il est assez fastidieux de représenter un arbre avec un unique tableau surtout pour un arbre très profond. Nous allons donc utiliser une seconde méthode, l'idée est de **représenter l'arbre avec un tableau contenant des tableaux !**

Exemple :



Cet arbre se représente par le tableau : **['r',['a',[],[]],['b',[],[]]]**

Dans la suite de l'exercice nous allons utiliser cet arbre binaire :



1. Création de l'arbre et insertion de nœuds :

```python
# Méthode récursive :

# Insertion
def Noeud(nom, fg =None, fd =None) :
    return {'racine': nom, 'fg' : fg, 'fd': fd}

# Définition des nœuds :
e = Noeud('e')
S = Noeud('S')
i = Noeud('i')
B = Noeud('B')
I = Noeud('I',S,e)
A = Noeud('A', B,i)
racine = Noeud('L', A, I)

# Création de l'arbre :
def Construit(arbre) :
    if arbre == None:
        return[]
    else:
        return[arbre['racine'],Construit(arbre['fg']),Construit(arbre['fd'])]

# Appel de la fonction seulement pour la racine, le reste se fait récursivement :
arbre=Construit(racine)
```

On utilise une manière tout à fait différente de la première, on se retrouve donc avec une représentation de l'arbre totalement différente !

Voici le résultat :

```
['L', ['A', ['B', [], []], ['i', [], []]], ['I', ['S', [], []], ['e', [], []]]]
```

2. Calcul de la hauteur de l'arbre :

L'idée est la suivante :

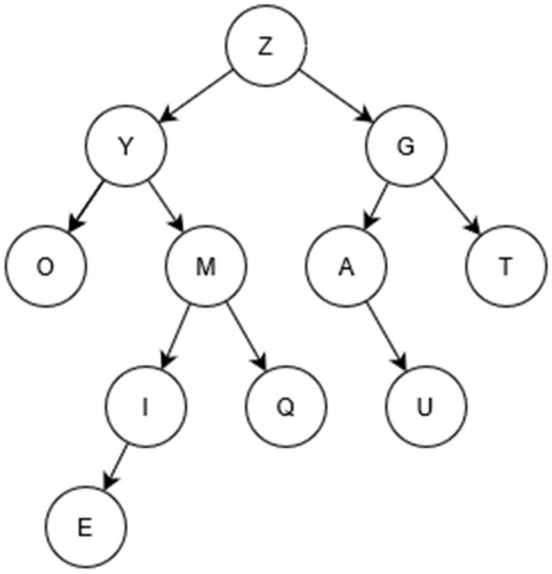
- Si l'arbre est vide la hauteur vaut -1.
- Sinon la hauteur vaut 1 auquel il faut ajouter le maximum entre les hauteurs des sous-arbres gauche et droit.
- Ces sous-arbres sont eux-mêmes des arbres dont il faut calculer la hauteur.

Une méthode récursive semble tout à fait adaptée à la situation.

```python
# Calcul de la hauteur de l'arbre :
def hauteur(arbre):
    if len(arbre) == 0 :
        return -1
    else :
        h1 = 1 + hauteur(arbre[1])
        h2 = 1 + hauteur(arbre[2])
        return max(h1,h2)
```

3. Modifications :

Notre script nous permet de modifier facilement l'arbre étudié, nous allons juste redéfinir les nœuds pour obtenir l'arbre binaire suivant :




```
# Définition des nœuds :
E = Noeud('E')
U = Noeud('U')
Q = Noeud('Q')
I = Noeud('I', E)
T = Noeud('T')
A = Noeud('A', None, U)
M = Noeud('M', I, Q)
O = Noeud('O')
G = Noeud('G', A, T)
Y = Noeud('Y', O, M)
racine = Noeud('Z', Y, G)
```

On obtient donc le tableau suivant :

```
['Z', ['Y', ['O', [], []], ['M', ['I', ['E', [], []], []], ['Q',
[], []]]], ['G', ['A', [], ['U', [], []]], ['T', [], []]]]
```

De plus, la console nous retourne bien une hauteur de 4 !

Pour une meilleure compréhension du programme, je vous fournis ci-joint le code source Python afin que vous puissiez le tester sur votre propre machine ! Vous pouvez aussi modifier vous-même l'arbre de test, et notamment les nœuds le composant !