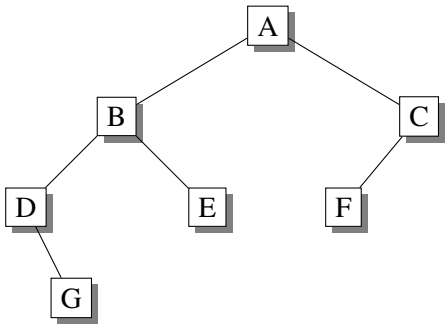
Spécialité NSI en terminale
Arbres binaires et algorithmes

1 Arbres binaires

1.1 Implémentation

Afin de pouvoir tester les différents programmes dans la suite, nous allons utiliser deux manières d'implémenter un arbre.

Prenons un exemple avec la représentation ci-dessous :



Nous pouvons écrire cet arbre sous la forme : `a = ['A', ['B', ['D', [], ['G', [], []]], ['E', [], []]], ['C', ['F', [], []], []]]`.

Une deuxième manière est d'utiliser une classe `Arbre` :

```
class Arbre:
    def __init__(self, val):
        self.valeur = val
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, val):
        self.gauche = Arbre(val)

    def ajout_droit(self, val):
        self.droit = Arbre(val)

b = Arbre('A')
b.ajout_gauche('B')
b.ajout_droit('C')
b.gauche.ajout_gauche('D')
b.gauche.ajout_droit('E')
b.gauche.gauche.ajout_droit('G')
b.droit.ajout_gauche('F')
```

1.2 Taille d'un arbre binaire

Il s'agit de déterminer le nombre total de nœuds.

Principe de l'algorithme :

- si l'arbre est vide, il y a 0 nœud ;

- sinon, nous comptons le nœud racine plus le nombre de nœuds du sous-arbre gauche et le nombre de nœuds du sous-arbre droit.

Pour la première implémentation, un programme en Python s'écrit sous la forme :

```
def taille1(arbre):
    if arbre == []:
        return 0
    else:
        return 1 + taille1(arbre[1]) + taille1(arbre[2])
```

Pour la deuxième implémentation, un programme en Python s'écrit sous la forme :

```
def taille2(arbre):
    if arbre is None:
        return 0
    else:
        return 1 + taille2(arbre.gauche) + taille2(arbre.droit)
```

Il est possible d'avoir une seule fonction `taille` pour les deux implémentations. C'est une question d'interface.

On définit pour cela trois fonctions : `vide`, `gauche` et `droit`.

```
def vide(arbre):
    return (arbre == []) or (arbre is None)

def gauche(arbre):
    if isinstance(arbre, list): # teste si arbre est une liste
        return arbre[1]
    else:
        return arbre.gauche

def droit(arbre):
    if isinstance(arbre, list):
        return arbre[2]
    else:
        return arbre.droit
```

La fonction `taille` peut alors prendre une forme unique :

```
def taille(arbre):
    if vide(arbre):
        return 0
    else:
        return 1 + taille(gauche(arbre)) + taille(droit(arbre))
```

Le résultat est identique pour les deux définitions de l'arbre exemple.

```
>>> taille(a)
7
>>> taille(b)
7
```

1.3 Hauteur d'un arbre binaire

Il s'agit de déterminer le nombre maximal de nœuds se trouvant entre la racine et une feuille.
Principe de l'algorithme :

- si l'arbre est vide, il contient 0 nœud ;
- sinon, nous comptons le nœud racine plus le maximum entre le nombre de nœuds du sous-arbre gauche et le nombre de nœuds du sous-arbre droit.

Un programme en Python peut s'écrire sous la forme suivante :

```
def hauteur(arbre):
    if vide(arbre):
        return 0  # ou -1 suivant la définition utilisée de la hauteur
    else:
        return 1 + max(hauteur(gauche(arbre)), hauteur(droit(arbre)))
```

Cette fonction est utilisable avec les deux implémentations de l'arbre exemple.

```
>>> hauteur(a)
4
>>> hauteur(b)
4
```

Dans le cadre d'une définition à l'aide d'une classe, une autre manière est envisageable : définir deux méthodes `taille` et `hauteur` dans la classe.

```
class Arbre:
    def __init__(self, val):
        self.valeur = val
        self.gauche = None
        self.droit = None

    def ajout_gauche(self, val):
        self.gauche = Arbre(val)

    def ajout_droit(self, val):
        self.droit = Arbre(val)
```

```
    def taille(self):
        tg = self.gauche.taille() if self.gauche else 0
        td = self.droit.taille() if self.droit else 0
        return 1 + tg + td

    def hauteur(self):
        hg = self.gauche.hauteur() if self.gauche else 0
        hd = self.droit.hauteur() if self.droit else 0
        return 1 + max(hg, hd)
```

Pour l'utilisation, nous définissons le même arbre que précédemment.

```
b = Arbre('A')
b.ajout_gauche('B')
b.ajout_droit('C')
b.gauche.ajout_gauche('D')
b.gauche.ajout_droit('E')
b.gauche.gauche.ajout_droit('G')
b.droit.ajout_gauche('F')
```

On exécute le programme dans l'interpréteur puis on utilise les deux méthodes :

```
>>> b.taille()
7
>>> b.hauteur()
4
```