Implémentation de données arborescentes

Implémentation récursive

De la même façon que les structures de données séquentielles, les données arborescentes sont des types abstraits de données qui peuvent être implémentée informatiquement de différentes façons.
Nous avons vu qu'il était possible de représenter un arbre binaire avec une listes dont la longueur dépendra de la hauteur de l'arbre. Cette représentation n'est pas très efficace car elle nécessite de coder de façon séquentielle une structure arborescente. Une solution plus efficace consiste à définir un arbre de façon récursive.

Mise en évidence de la récursion

En effet un arbre peut être vu comme un nœud comportant deux sous arbres gauche et droit (voir figure 1). Il s'agit donc d'une structure récursive puisque les sous arbres sont aussi des arbres
Le but de cet exercice est de définir une classe permettant d'implémenter un arbre binaire de façon récursive

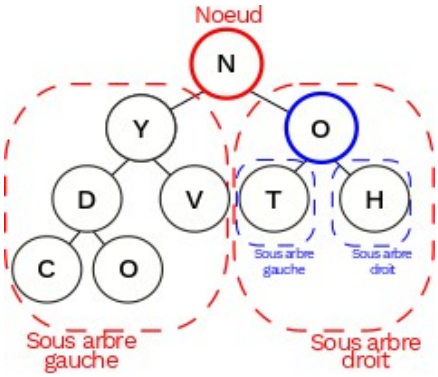

Figure 1: Représentation récursive d'un arbre

Expression minimale du TAD Arbre

Les primitives de base du type abstrait de donnée Arbre sont les suivantes :

CREER_ARBRE(d, gauche, droit) qui retourne un objet de type arbre.
La valeur de la racine est définie par d ; gauche et droit correspondent au contenu des sous arbres gauche et droit. Dans le cas d'une feuille, gauche et droit prennent la valeur None.

NOEUD(A) qui retourne la valeur du nœud de l'arbre A

SAG(A) qui retourne le sous arbre gauche de l'arbre A

SAD(A) qui retourne le sous arbre droit de l'arbre A

EST_FEUILLE(A) qui renvoie un booléen Vrai si le nœud est une feuille.

Exploitation des primitives

On considère l'arbre défini sur la figure 2.

1. **Entourer** sur la figure 2 le sous arbre défini par l' instruction suivante :

```
CREER_ARBRE('Y', CREER_ARBRE('D', CREER_ARBRE('C', None, None) , CREER_ARBRE('O', None, None) ) , CREER_ARBRE('V', None, None) )
```

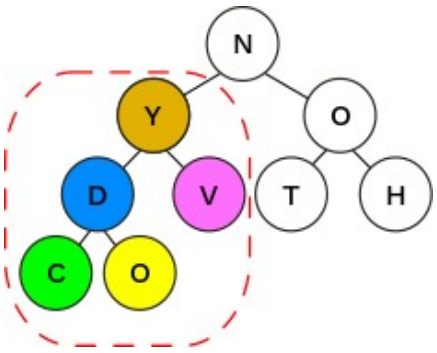

Figure 2: Arbre binaire étudié

2. **Compléter** cette instruction pour définir complètement l'arbre représenté sur la figure 2.

CREER_ARBRE('N', CREER_ARBRE('Y', CREER_ARBRE('D', CREER_ARBRE('C', None, None) , CREER_ARBRE('O', None, None)) , CREER_ARBRE('V', None, None)) , CREER_ARBRE('O', CREER_ARBRE('T', None, None), CREER_ARBRE('H', None, None)))

Pour la suite cet arbre sera nommé abr.

3. **Indiquer** les valeurs renvoyées par les instructions suivantes :

NOEUD(SAD(abr)) ➞ renvoie 'O'

EST_FEUILLE(SAG(SAG(abr))) ➞ Renvoie False (Noeud D n'est pas une feuille)

EST_FEUILLE(SAD(SAD(abr)) ➞ Renvoie True (H est une feuille)

4. **Indiquer** les instructions qui permettent d'effectuer les opération suivantes :

➔ Tester si le nœud T de abr est une feuille.

EST_FEUILLE(SAG(SAD(abr))) ➞ Renvoie True (T est une feuille)

➔ Renvoyer la valeur du sous arbre droit du sous arbre gauche de l'arbre abr.

NOEUD(SAD(SAG(abr))) ➞ renvoie 'V'

Programmation objet de l'implémentation récursive d'un arbre.

La programmation objet incomplète du TAD défini précédemment est la suivante :

```
class ArbreBin :
    def __init__(self,d, gauche=None, droit=None) :
    '''Construit un arbre constitue d'un noeud d'étiquette , d'un sous arbre gauche
    et un sous arbre droit.
    gauche = None et droit = None par defaut -> construction d'une feuille)'''
          self.__data = d
          self.__sag = gauche
          self.__sad = droit
    def est_feuille(self) :
    '''Retourne vrai si le noeud est une feuille'''
          return self.__sag==None and self.__sad==None
    def sag(self) :
          '''retorune le sous arbre gauche'''
          return self.__sag
    def sad(self) :
    '''retorune le sous arbre droit'''
          return self.__sad
    def noeud(self) :
    '''retorune l'etiquette du noeud'''
          return self.__data
```

Code disponible sur .\donnee\NSI\arbres\code\ArbreBin.py

Nombre de nœuds de l'arbre

On souhaite définir une méthode nbre_noeuds() qui renvoie le nombre de nœuds internes (sans les feuilles) d'un arbre binaire équilibré. Cette recherche peut se faire récursivement puisque ce nombre est égale au nombre de nœuds du sous arbre gauche plus le nombre du sous arbre droit plus un (lui même).

5. **Indiquer** la condition d'arrêt de cette fonction récursive nbre_noeuds()

Condition d'arrêt : Si le nœud est une feuille il faut renvoyer 0

6. **Programmer** la méthode nbre_noeuds() et **tester** son fonctionnement avec l'arbre de la figure 2 (variable abr)

```
def nbre_noeuds(self, debug = False) :
    ''' retourne le nombre de noeuds de l'arbre'''
    if debug : print('-> nbre de noeuds de ',self.__data)
    if self.est_feuille() :
          if debug : print (' <- Sous arbre ',self.__data , 'retourne 0')
          return 0
    else :
          if debug :
                res = 1 + self.sag().nbre_noeuds(True) +  \
                      self.sad().nbre_noeuds(True)
                print (' <- Sous arbre ',self.__data , 'retourne : ', res)
```

```
            return res
        else : return 1 + self.sag().nbre_noeuds() + self.sad().nbre_noeuds()
```

```
def nbre_noeuds(self, debug = False) :
    ''' retourne le nombre de noeuds de l'arbre'''
    if debug : print('-> nbre de noeuds de ',self.__data)
    if self.est_feuille() :
          if debug : print (' <- Sous arbre ',self.__data , 'retourne 0')
          return 0
    else :
          if debug :
                res = 1 + self.sag().nbre_noeuds(True) +  \
                      self.sad().nbre_noeuds(True)
                print (' <- Sous arbre ',self.__data , 'retourne : ', res)
                return res

          else : return 1 + self.sag().nbre_noeuds() + self.sad().nbre_noeuds()
```

```
>>> abr.nbre_noeuds(True) # (nombre de nœud en mode debug)
-> nbre de noeuds de  N
-> nbre de noeuds de  Y
-> nbre de noeuds de  D
-> nbre de noeuds de  C
   <- Sous arbre  C retourne 0
-> nbre de noeuds de  O
   <- Sous arbre  O retourne 0
   <- Sous arbre  D retourne :  1
-> nbre de noeuds de  V
   <- Sous arbre  V retourne 0
   <- Sous arbre  Y retourne :  2
-> nbre de noeuds de  O
-> nbre de noeuds de  T
   <- Sous arbre  T retourne 0
-> nbre de noeuds de  H
   <- Sous arbre  H retourne 0
   <- Sous arbre  O retourne :  1
   <- Sous arbre  N retourne :  4
4
```

Nombre de nœuds de feuilles

On souhaite définir une méthode nbre_feuilles() qui renvoie le nombre de feuilles d'un arbre binaire équilibré.

7. **Programmer** la méthode nbre_feuilles() et **tester** son fonctionnement avec l'arbre de la figure 2 (variable abr)

```
def nbre_feuilles(self) :
    ''' retourne le nombre de feuilles de l'arbre'''
    if self.est_feuille() : return 1
    else : return self.sag().nbre_feuilles() + self.sad().nbre_feuilles()
```

Structures de l'arbre

8. **Programmer** une méthode hauteur() qui renvoie la hauteur de l'arbre. **Tester** son fonctionnement avec l'arbre de la figure 2 (variable abr)
9. **Programmer** une méthode feuilles() qui renvoie la liste des étiquettes des feuilles de l'arbre. **Tester** son fonctionnement avec l'arbre de la figure 2 (variable abr)

```
def feuilles(self):
    '''retourne la liste des étiquettes des feuilles de l'arbre'''
    if self.est_feuille() : return [self.noeud()]
    else : return self.sad().feuilles() + self.sag().feuilles()

def hauteur(self) :
    '''Renvoie la hauteur de l'arbre'''
    if self.est_feuille() : return 0
    else : return 1+ max(self.sag().hauteur(), self.sad().hauteur())
```