

Les structures que l'on a vu jusque maintenant (tableaux, listes, piles, files) sont des structures dites **séquentielles** car les éléments stockés le sont les uns à la suite des autres.

À l'inverse, une structure **arborescente** est une structure partant d'un unique point (appelé **racine**) et développant plusieurs **branches** (on parle aussi parfois de fils) se scindant elles même à chaque étape.

On connaît déjà de telles structures :

- le système de fichiers dans un système d'exploitation ;
- les arbres de probabilités.
- les arbre génétiques / phylogénétiques.

1 Définition et propriétés des arbres binaires

Un arbre binaire est une structure récursive comportant des noeuds (les valeurs) liés les uns aux autres. Plus précisément, un arbre binaire peut être :

- un arbre vide ;
- composé d'un noeud comportant :
 - une valeur ;
 - un sous-arbre gauche ;
 - un sous arbre droit.

On appelle un noeud dont les deux sous-arbres sont vides une **feuille**.

En somme, un arbre binaire peut être vu comme une liste chaînée ayant à chaque cellule deux cellules suivantes. Exemples et exercices 70 et 71 (fiche A3) :

La **taille** d'un arbre binaire est son nombre de noeuds, sa **hauteur** est le plus grand nombre de noeud étant rencontré en descendant de la racine à chaque feuille.

On peut définir ces deux notions récursivement :

Taille :

- L'arbre vide a une taille de 0 ;
- La taille d'un arbre non-vidé est de 1 plus la somme des tailles de ses sous arbres.

Hauteur :

- L'arbre vide a une hauteur de 0 ;
- La hauteur d'un arbre non-vidé est de 1 plus la plus grande des hauteurs de ses sous arbres.

Exemples :

Si A est un arbre de taille N et de hauteur h , on a l'inégalité suivante :

$$h \leq N \leq 2^h - 1$$

Les cas d'égalités dans cette inégalité ont lieu pour certains arbres que l'on appelle :

- **peignes** lorsque $h = N$;
- **arbres parfaits** lorsque $N = 2^h - 1$.

Dans un peigne, chaque noeud a au plus un sous arbre non-vide ce qui lui confère une structure équivalente à celle d'une liste chaînée.

Un arbre parfait correspond à un arbre de hauteur donnée rempli avec le maximum de noeuds possible.

Exemples :

2 Représentation en Python

Nous allons créer un module permettant de travailler avec des arbres en python. On crée pour cela une classe **Noeud** :

```
1 class Noeud:
2     def __init__(self, g, v, d):
3         """Un noeud d'un arbre binaire"""
4         self.gauche = g
5         self.valeur = v
6         self.droite = d
7
8 def taille(a):
9     """renvoie le nombre de noeuds present dans l'arbre a"""
10    if a is None :
11        return 0
12    else :
13        return 1 + taille(a.gauche) + taille(a.droite)
14
15 def hauteur(a):
16     """renvoie la hauteur de l'arbre a"""
17    if a is None :
18        return 0
19    else :
20        return 1 + max(hauteur(a.gauche), hauteur(a.droite))
```

Sur le site, récupérer le module du chapitre et créer les arbres suivant grâce au constructeur `Noeud(g,v,d)` :

3 Parcours d'un arbre

Lorsque l'on veut parcourir (récursivement) les valeurs d'un arbre – par exemple pour en afficher les valeurs, l'ordre dans lequel on fait les appels récursifs modifie fondamentalement l'ordre global du parcours. On peut citer trois types de parcours :

- le **parcours infixe** : on affiche les valeurs du sous arbre gauche, puis la valeur du noeud, puis les valeurs du sous-arbre droit ;
- le **parcours préfixe** : on affiche la valeur du noeud, puis les valeurs du sous arbre gauche, puis les valeurs du sous-arbre droit ;
- le **parcours postfixe** : on affiche les valeurs du sous arbre gauche, puis les valeurs du sous-arbre droit, puis la valeur du noeud ;

```
1 def parcours_infixe(a):
2     """affiche les noeuds de a en suivant un parcours infixe"""
3     if not(a is None) :
4         parcours_infixe(a.gauche)
5         print(a.valeur)
6         parcours_infixe(a.droite)
7
8 def parcours_prefixe(a):
9     """affiche les noeuds de a en suivant un parcours prefixe"""
10    if not(a is None) :
11        print(a.valeur)
12        parcours_prefixe(a.gauche)
13        parcours_prefixe(a.droite)
14
15 def parcours_postfixe(a):
16     """affiche les noeuds de a en suivant un parcours postfixe"""
17     if not(a is None) :
18         parcours_postfixe(a.gauche)
19         parcours_postfixe(a.droite)
20         print(a.valeur)
```

Tous les algorithmes présentés ici ont une complexité linéaire en la taille des arbres.