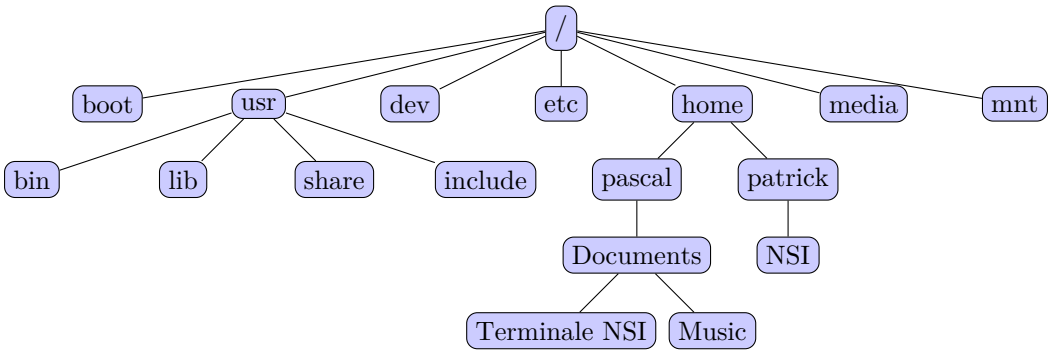
Les Arbres

1 Structures arborescentes

1.1 Introduction

Dans un chapitre précédent, nous avons exploré une première structure chaîné, la liste chaînée. Les structures arborescentes sont une autre famille de structures chaînés très présentes en informatique dont vous avez déjà rencontré un exemple l'année dernière : l'arborescence des fichiers d'un ordinateur.

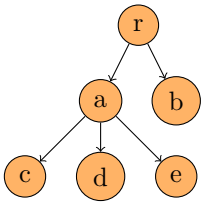


L'organisation sous forme d'arbre des fichiers d'un ordinateur permet notamment d'accéder en un petit nombre d'étape à n'importe quel fichier choisi parmi des dizaines de milliers, *pour peu qu'on aille dans la bonne direction.*

1.2 Définitions et vocabulaire

Les éléments d'un arbre sont appelés des **noeuds**. Le noeud au sommet de l'arbre est appelé **racine**. Un noeud peut avoir un ou plusieurs **fils** qui sont égalements des noeuds. Un noeud sans fils est une **feuille** (ou noeud externe), les autres noeuds sont des **noeuds internes**.
La **taille** d'un arbre est le nombre de noeuds qui le composent. La **hauteur** d'un arbre est le plus grand nombre de noeuds rencontrés en descendant du sommet jusqu'une feuille.

Exemple
On considère l'arbre ci-contre.



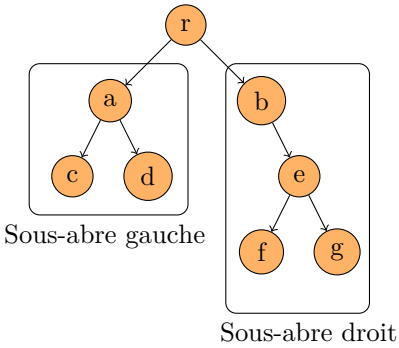
- Les noeuds sont `r`, `a`, `b`, `c`, `d` et `e`.
- La racine est le noeud `r`.
- Le noeud `r` a pour fils les noeuds `a` et `b`. Le noeud `a` a pour fils les noeuds `c`, `d` et `e`. Le noeud `b` n'a pas de fils.
- Les feuilles sont les noeuds `b`, `c`, `d` et `e`.
- La taille de l'arbre est 6.
- La hauteur de l'arbre est 3.

Remarque : La heuteur d'un arbre vide est 0.

2 Arbres binaires

2.1 Définitions

Un **arbre binaire** est un arbre dont chaque noeud possède au plus 2 fils généralement ordonnés, le fils gauche et le fils droit chacun respectivement racine du **sous-arbre gauche** et du **sous-arbre droit**.

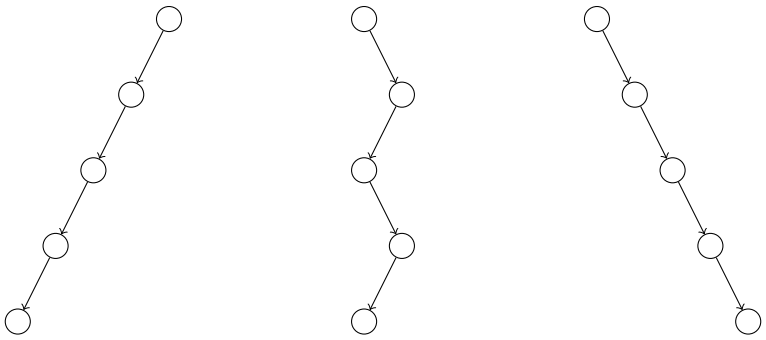


2.2 Hauteur d'un arbre bianire

Soit N la taille d'un arbre binaire et h sa hauteur, alors :

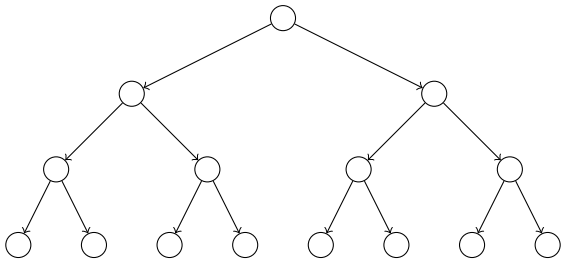
$$h \leq N \leq 2^h - 1$$

A gauche, l'égalité est atteinte avec des arbres formés d'un seul noeud à chaque niveau.



La hauteur est égale à la taille et l'arbre obtenu n'est pas différent d'une liste chaînée.

Pour l'autre côté, l'égalité est atteinte pour un arbre binaire **parfait** où toutes les feuilles ont exactement la même profondeur.



Le nombre de noeuds est $2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{h-1} = 2^h - 1$.

3 Représentation en Python

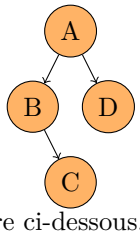
Il y a de nombreuses façon de représenter un arbre binaire en Python. Comme pour les listes chaînées, une façon habituelle consiste à représenter chaque noeud par objet d'une classe qui ici s'appelera `Noeud`.

```
class Noeud:
    '''un noeud d'un arbre binaire'''
    def __init__(self, v, g, d):
        self.valeur = v
        self.gauche = g
        self.droit = d
```

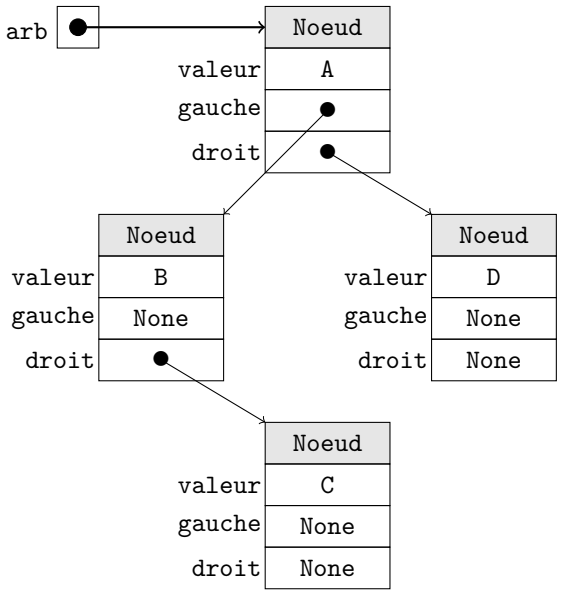
La classe contient trois attributs : un attribut `valeur` pour la valeur contenue par le noeud, un attribut `gauche` pour pour le sous arbre gauche et un attribut `droit` pour le sous arbre droit.

Pour construire un arbre, il suffit alors d'appliquer le constructeur de la classe `Noeud` autant de fois que nécessaire.

```
arb = Noeud('A',
      Noeud('B', None, Noeud('C', None, None)),
      Noeud('D', None, None)
```



On a ainsi créé quatre objets de la classe `Noeud` qui sont illustrés dans la figure ci-dessous.



4 Algorithme sur les arbres binaires

4.1 Taille et hauteur d'un arbre binaire

La définition d'un arbre binaire étant récursive, il est naturel d'utiliser un algorithme récursif pour calculer sa taille.

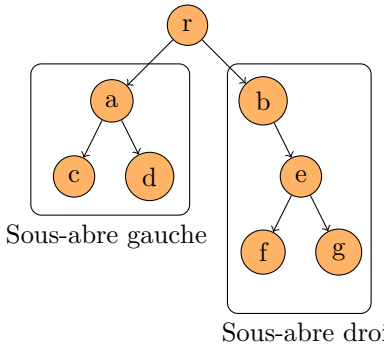
```
def taille(arb):
    '''calcul la taille d'un arbre binaire'''
```

Le cas de base (condition d'arrêt) est celui d'un arbre vide dont la taille est 0.

```
    if arb is None:
        return 0
```

La taille d'un arbre non vide, est la somme des tailles des deux sous-arbres gauche et droit plus 1 pour la racine.

```
    else:
        return 1 + taille(arb.gauche) + taille(arb.droite)
```



$$taille(\text{arbre}) = 1 + taille(\text{sous-arbre gauche}) + taille(\text{sous-arbre droit})$$

Ce qui donne la fonction `taille` suivante :

```
def taille(arb):
    '''calcul la taille d'un arbre binaire'''
    if arb is None:
        return 0
    else:
        return 1 + taille(arb.gauche) + taille(arb.droite)
```

De la même manière, si arbre est vide, sa hauteur est 0, et dans le cas contraire sa hauteur est 1 plus le maximum des hauteurs des sous-arbres droit et gauche. Ce qui donne la fonction `hauteur` suivante :

```
def hauteur(arb):
    '''calcul la hauteur d'un arbre binaire'''
    if arb is None:
        return 0
    else:
        return 1 + max(hauteur(arb.gauche),hauteur(arb.droite))
```

4.2 Parcours d'un arbre binaire

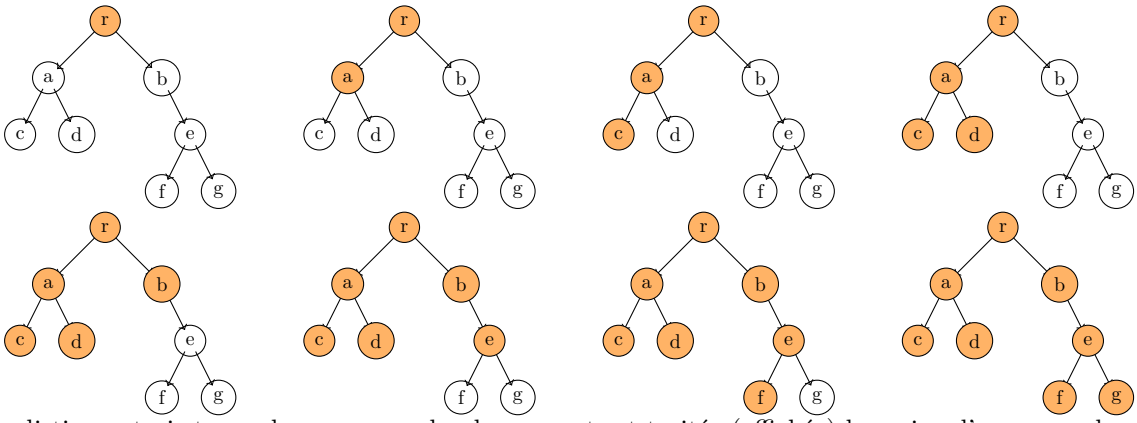
Les fonctions `hauteur` et `taille` parcours tous les noeuds de l'arbre ; l'odre dans lequel l'arbre est parcouru n'a pas d'importance. Si l'on veut afficher les valeurs des noeuds, l'odre dans lequel on rencontre les noeuds devient important.

4.2.1 Parcours en profondeur d'abord

Dans le cas où il est **parcouru en profondeur dabord**, on explore complètement une branche avant d'explorer la branche voisine à l'aide de l'algorithme suivant :

— si l'arbre est non vide, on parcours de manière récursive son sous-arbre gauche puis son sous-arbre droit ;

— sinon, l'algorithme se termine.



On distingue trois types de parcours selon le moment est traitée (affichée) la racine d'un sous-arbre :

— Dans un parcours préfixe, la racine est traité avant ses deux sous-arbres.

— Dans un parcours infixe, a racine est traité entre ses deux sous-arbres.

— Dans un parcours postfixe, la racine est traité après ses deux sous-arbres.

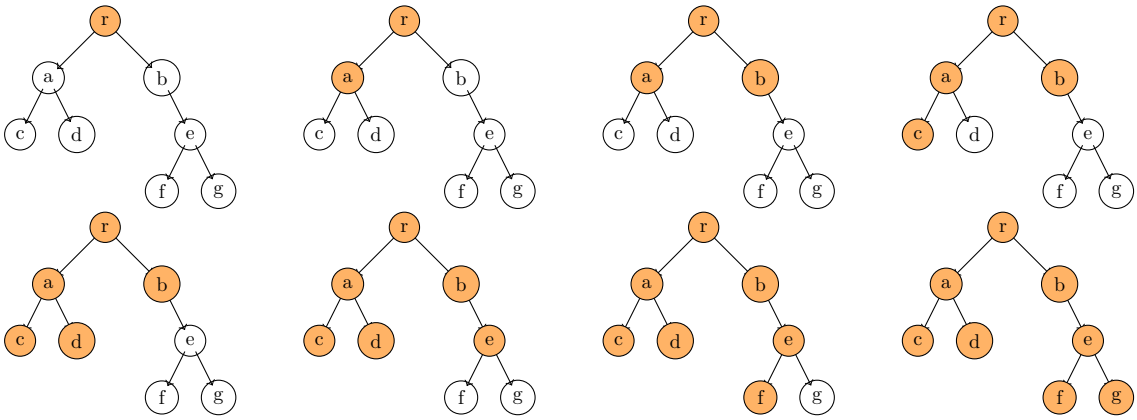
```
def parcours_prefixe(arb):
    '''affiche les noeuds de arb dans un parcours préfixe'''
    if arb is not None:
        print(arb.valeur, end=" ")
        parcours_prefixe(arb.gauche)
        parcours_prefixe(arb.droit)

def parcours_infixe(arb):
    '''affiche les noeuds de arb dans un parcours infixe'''
    if arb is not None:
        parcours_infixe(arb.gauche)
        print(arb.valeur, end=" ")
        parcours_infixe(arb.droit)

def parcours_postfixe(arb):
    '''affiche les noeuds de arb dans un parcours postfixe'''
    if arb is not None:
        parcours_postfixe(arb.gauche)
        parcours_postfixe(arb.droit)
        print(arb.valeur, end=" ")
```

Sur l'exemple d'illustration, on obtient les résultats suivants :

```
>>> parcours_prefixe(arb)
R A C D B E F G
>>> parcours_infixe(arb)
C A D R B F E G
>>> parcours_postfixe(arb)
C D A F G E B R
```

4.2.2 Parcours en largeur d'abord

Dans le cas d'un **parcouru en largeur d'abord**, on l'arbre étage par étage. On commence par la racine, puis les deux racines de chacun des sous-arbres, puis les quatres racines de chacun des quatres sous-arbres et ainsi de suite.



Auteur : Pascal Seckinger