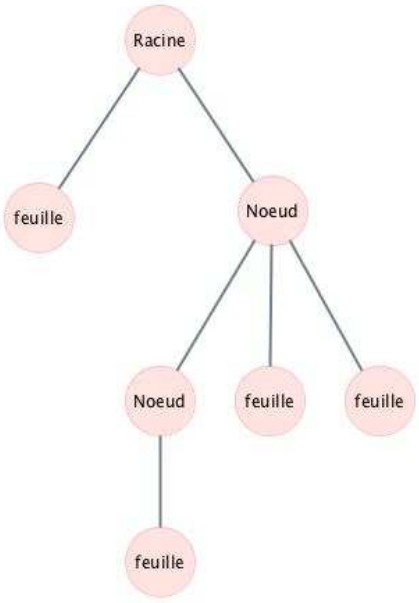
Structure de données
Les arbres

1 Définitions

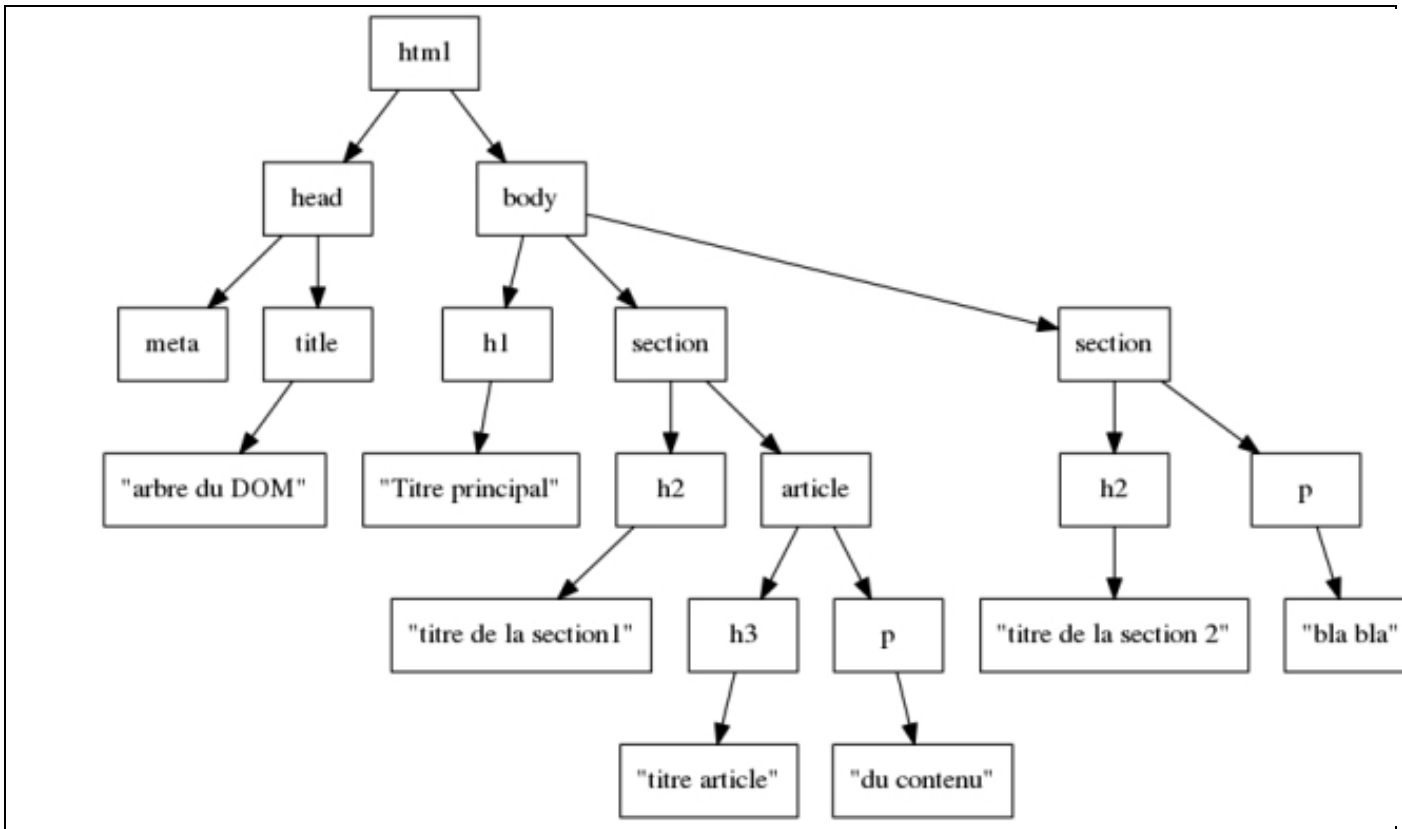
Un arbre en informatique est un type particulier de graphe. Comme nous n'avons pas encore vu à ce stade de l'année les graphes, cette définition n'est pas d'une grande utilité... On dira donc qu'un arbre est constitué :

- d'une racine, sommet de "départ" de l'arbre ;
- de nœuds , sommets intermédiaires de l'arbre ;
- de feuilles, sommets "finaux" de l'arbre ;
- et de branches, qui relient les éléments précédents entre eux.

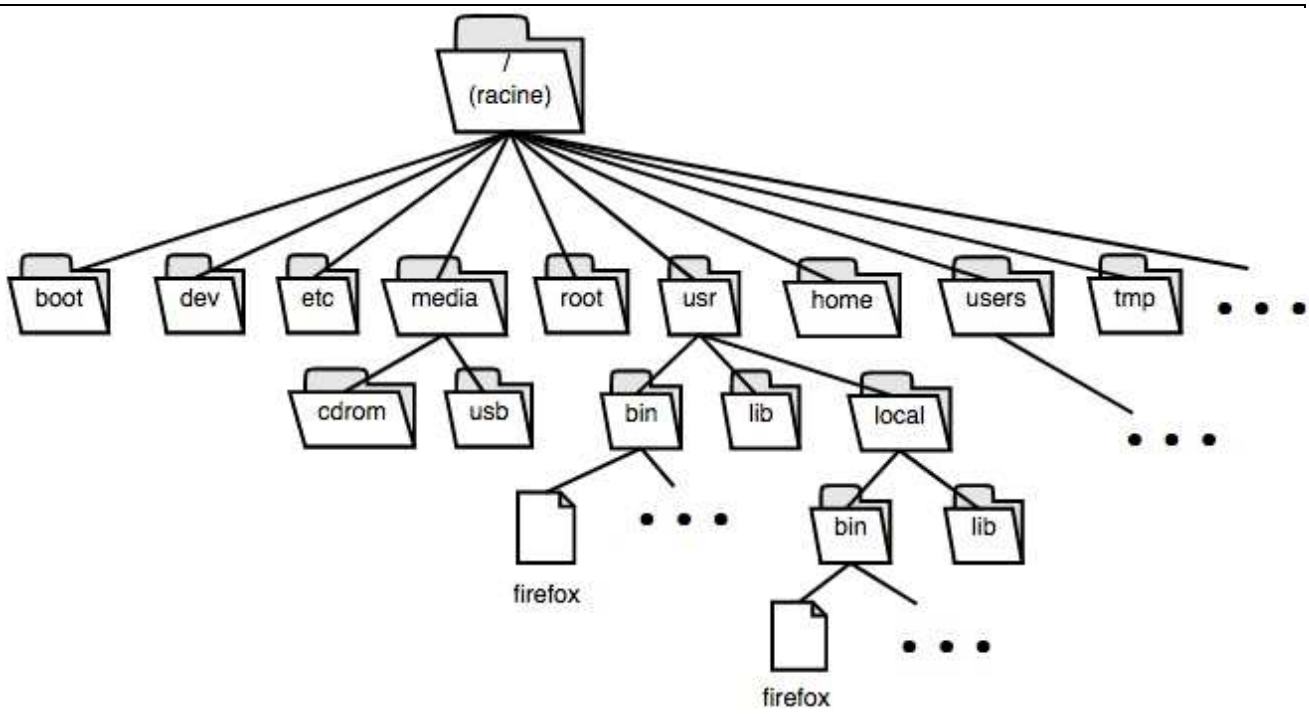


Les arbres informatiques ont ceci de particulier qu'ils poussent tête en bas (ils viennent de l'hémisphère sud). Les arbres sont des structures de données hiérarchiques, très utilisées en informatique. Ils sont orientés : la représentation standardisée, racine en haut, indique la relation "père-fils" entre les sommets. Lorsque deux sommets sont reliés par une branche, celui du haut est le père, et celui du bas est le fils. Un père peut avoir plusieurs fils, mais un fils ne peut pas avoir plusieurs pères (faites un dessin, on obtiendrait alors ce qu'on appelle un cycle dans le vocabulaire des graphes).

L'arborescence des fichiers ou d'un document html donne des exemples d'arbres en informatique.



Une arborescence html



Une arborescence de fichiers

1.1 Définitions complémentaires

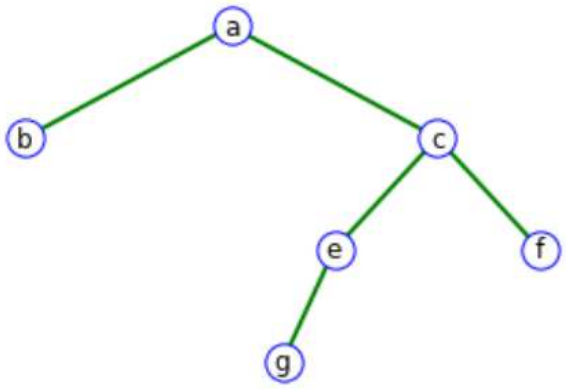
Les nœuds sont en général étiquetés, ci-contre les **étiquettes** sont les lettres a, b, c, etc…

La **taille** d'un arbre est le nombre de ses nœuds. Dans l'exemple ci-contre, la taille de l'arbre est 7.

La **hauteur** (ou **profondeur** ou **niveau**) d'un nœud X est définie comme le nombre de nœud à parcourir pour aller de la racine au nœud X. La hauteur de la racine est arbitrairement fixée à 1, ou 0 suivant les définitions.

La **hauteur** de l'arbre est la plus grande des hauteurs de ses nœuds

Exemple : le nœud ⓒ a pour hauteur 2, le nœud ⓕ a pour hauteur 3, le nœud ⓖ a pour hauteur 4 (avec 1 comme hauteur pour a). la hauteur de l'arbre est égale à la hauteur du nœud ⓖ, soit 4.



L'**arité**d'un nœud est le nombre de ses fils

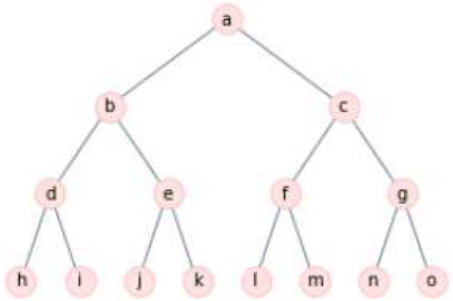
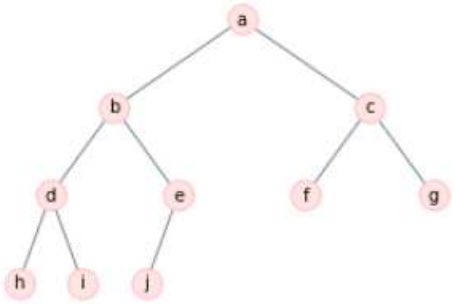
Exemple : ⓐ a pour arité 2, ⓑ a pour arité 0, ⓒ a pour arité 3

Un arbre peut être défini de manière récursive : un arbre est soit un arbre vide, soit une racine et une liste de sous-arbres (éventuellement vide, auquel cas c'est une feuille). *C'est la définition qui est probablement la plus pertinente pour la vision « informatique » des arbres.*

En terminale, nous étudierons principalement les **arbres binaires** : un arbre binaire est un arbre dont tous les nœuds sont d'arité au maximum 2. Autrement dit, chaque père a au plus deux fils, appelés sous-arbre gauche et sous-arbre droit.

Les arbres binaires se rencontrent par exemple dans les compétitions sportives comme un tournoi de tennis.

Trois cas particuliers :

Arbre dégénéré ou filiforme	 Arbre parfait	 Arbre complet ou presque complet : c'est un arbre dont tous les niveaux sont complètement remplis, sauf éventuellement le dernier. Les feuilles du dernier niveau sont le plus à gauche possible

A noter que ces définitions varient suivant les auteurs et les livres… notamment « parfait » et « complet » peuvent être intervertis !

Lien entre taille et hauteur. Si un arbre binaire est de taille *n* et de hauteur *h*, alors :

- le nombre de sommets est au moins égal à la hauteur, et strictement inférieur à $2^{hauteur}$:
  $h \leq n \leq 2^h$
- De manière équivalente, la hauteur est strictement plus grand que le logarithme en base deux de la taille, et inférieur ou égal à la taille :
  $log_2 n < h \leq n$

2 Une classe arbre binaire en Python

Le but est de créer une structure de données pour représenter un arbre binaire en Python. Les opérations sur les arbres binaires sont au minimum :

- Construction d'arbre vide
- Construction d'un arbre à partir d'un entier et de deux sous-arbres gauche et droit
- Test de vacuité
- Accès à la racine d'un arbre
- Accès au sous-arbre gauche
- Accès au sous-arbre droit

2.1 Implémentation de base

Une classe nœud suffit à créer un arbre, qui n'est rien d'autre qu'une suite récursive de nœuds. La classe ArbreBinaire du paragraphe suivant n'est donc pas obligatoire. (C'est ce qui est utilisé dans le travail sur Jupyter)

2.2 Implémentation avec arbre et nœuds

Cette implémentation plus complète permet une manipulation plus simple des arbres :

On crée une classe nœud, puis on utilise ce nœud pour construire un arbre

```
classNoeud:
def__init__(self,valeur,gauche,droit):
self.n = valeur
self.g = gauche
self.d = droit
```

```
classArbreBinaire:
def__init__(self,c):
self.r = c

defcreeVide():
returnArbreBinaire(None)

defcreeNGD(valeur,gauche=None,droit=None):
returnArbreBinaire(Noeud(valeur,gauche,droit))

defestVide(self):
returnself.risNone

defracine(self):
assert not(self.risNone),'Arbre vide'
returnself.r.n

deffilsGauche(self):
assertnot(self.risNone),'Arbre vide'
returnself.r.g

deffilsDroit(self):
assertnot(self.r is None),'Arbre vide'
returnself.r.d
```

3 Algorithmes sur les arbres

Ces algorithmes sont à comprendre plus qu'à retenir

3.1 Taille et hauteur

Calcul de la taille : retourne le nombre de sommets de l'arbre (racine + nœuds + feuilles)

Fonction récursive taille(*arbre*) :
Si*arbre* est vide
Retourner 0
Sinon
Retourner 1 + taille (*fils gauche*) + taille (*fils droit*)

Calcul de la hauteur : retourne la hauteur de l'arbre

Fonction récursive hauteur(*arbre*) :
Si*arbre* est vide
Retourner 0
Sinon
Retourner 1 + max(hauteur (*fils gauche*), hauteur(*fils droit*))

4 Mise en œuvre logicielle :

Utiliser le notebook Jupyter « **Travail sur les arbres** » afin de faire le travail demandé.
Pour utiliser un fichier Jupyter, le plus simple est d'installer Anaconda puis de lancer Jupyter.
L'environnement EduPython contient également Jupyter. Vous pouvez également installer Jupyter de façon autonome.
Enfin si vous ne voulez pas utiliser Jupyter (dommage) vous trouverez la version pdf du notebook.