Chapitre 1

Arbres binaires

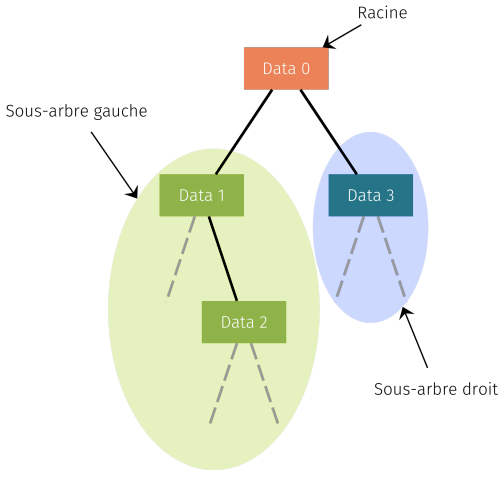
1 Généralités

1.1 Arbres

La structure d'arbre n'est pas linéaire : les éléments n'y sont pas rangés les uns à la suite des autres. C'est une structure *hiérarchique* : les éléments y sont organisés en *niveaux*.

Dans ce chapitre nous étudions les *arbres binaires*.

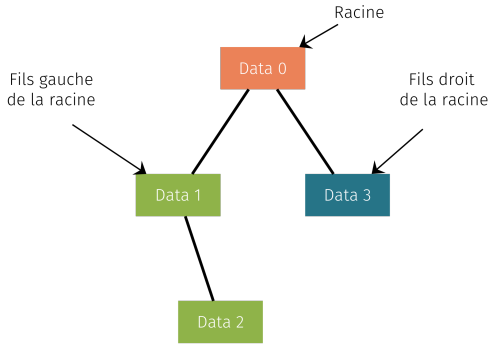
Définitions : arbre binaire, sous-arbre, racine



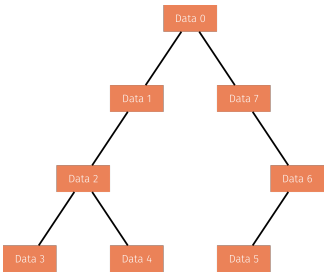
Un arbre binaire peut être vide. Sinon, il est composé d'au moins un nœud, appelé *racine* et le reste des nœuds peut être partagé en 2 sous-ensembles : le *sous-arbre gauche* et le *sous-arbre droit*.
Un nœud possède toujours un sous-arbre droit et un sous-arbre gauche, même si ceux-ci peuvent être vides (segments gris en pointillés).

Définition : fils

Quand le sous-arbre gauche d'un nœud est non-vide, sa racine est appelée *fils gauche* du nœud. On définit de même la notion de *fils droit*.



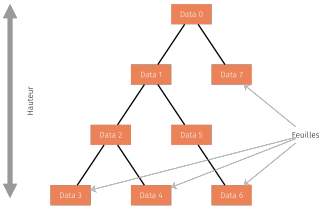
Définition : taille



La *taille* d'un arbre est le nombre de ses nœuds. Voici un arbre de taille 8.

Définition : hauteur et feuille

On appelle *feuille* tout nœud qui n'a ni fils droit ni fils gauche.
La *hauteur* d'un nœud est le nombre d'arêtes qui mène de la racine à ce nœud).
La hauteur de l'arbre est la plus grande des hauteurs des nœuds qui le composent.
Cet arbre est de hauteur 3.



Remarque

La définition de hauteur *n'est pas standard*. Selon celle-ci la hauteur de l'arbre vide n'est pas définie.

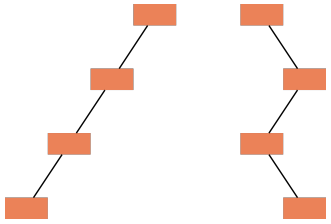
On choisit parfois de définir la hauteur d'un nœud comme le nombre de nœuds pour aller jusqu'à la racine incluse, et on fixe la hauteur de l'arbre vide à zéro.

Cela donne une hauteur qui est supérieure d'une unité par rapport à celle choisie dans ce cours.

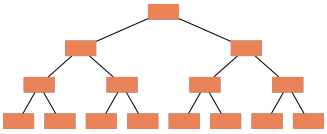
Définition : nœud interne, arbre dégénéré

Un nœud qui n'est ni la racine ni une feuille est dit *interne*.

Si tous les nœuds internes n'ont qu'un seul fils, l'arbre est dit *dégénéré*.



Définition : arbre parfait



Si tous les nœuds internes ont 2 fils et que toutes les feuilles sont à la même hauteur, on dit que l'arbre binaire est *parfait*.

2 Structure de données

2.1 Modèle théorique

Interface théorique

- *arbre_vide()* : crée un arbre binaire vide
- *racine(arbre)* : renvoie le nœud qui est la racine de l'arbre
- *gauche(arbre)* : renvoie le sous-arbre gauche de l'arbre
- *droit(arbre)* : renvoie le sous-arbre droit de l'arbre

– *contenu(nœud)* : renvoie l'élément stocké dans le nœud

On ne suivra pas complètement ce modèle théorique.

2.2 Implémentation en Python

Python

```python
class Node:
    def __init__(self, value, left=None, right=None):
        self.value = value
        self.left = left  # left child
        self.right = right  # right child
```

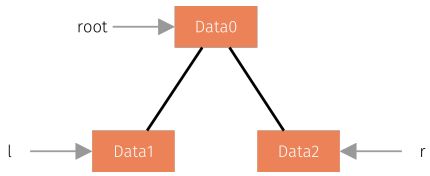
Dans ce modèle on ne définit que la classe `Node`. `Node(2)` renvoie un nœud sans fils droit ni fils gauche contenant la valeur 2.

Exemple

Ce code :

```
>>> l = Node('Data1')
>>> r = Node('Data2')
>>> root = Node('Data0', l, r)
```

Produit ceci :



Remarque

Une instance de la classe `Node` contient

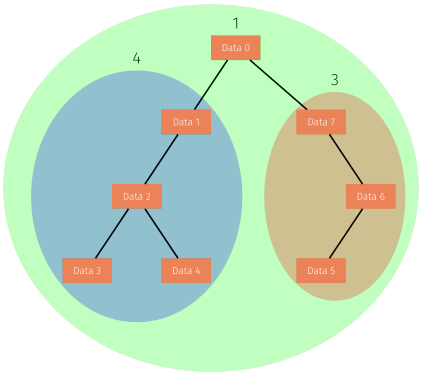
– une valeur ;

– 2 références à 2 autres instances de la classe `Node` (ou bien `None`).

Tout comme les listes chaînées, les arbres sont des structures qui se prêtent bien à la récursivité.

Exemple

La taille de l'arbre vide est 0, sinon c'est « 1 plus la taille du sous-arbre gauche plus la taille du sous-arbre droit ».



La taille de cet arbre est 1 + 4 + 3 = 8.

3 Parcours

3.1 Différents parcours

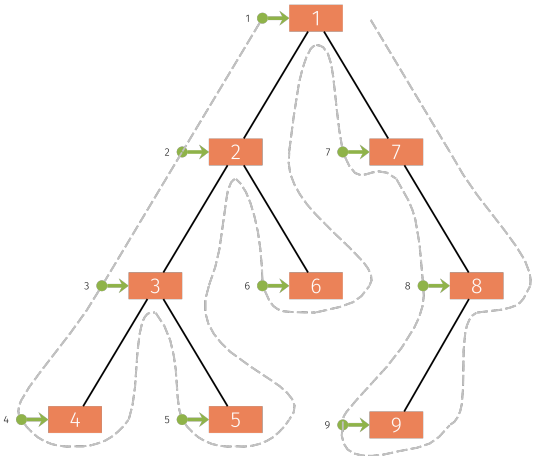
Pour parcourir un arbre binaire, on peut utiliser diverses méthodes récursives.
Parmi celles-ci, il en existe qui consistent à

- traiter le nœud courant ;
- parcourir le sous arbre gauche s'il est non vide ;
- parcourir le sous-arbre droit s'il est non vide.

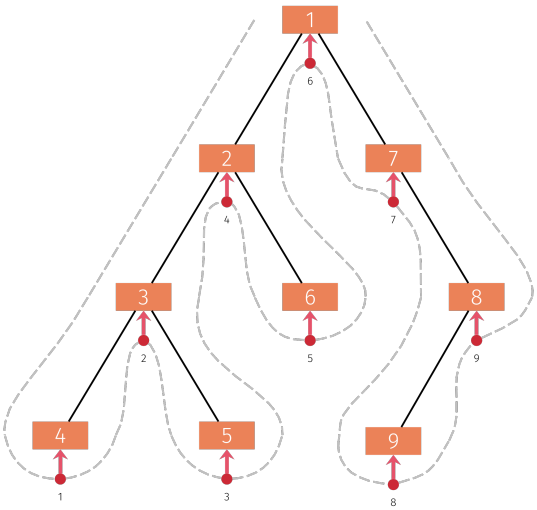
Et ceci *dans un ordre donné*.
Si on choisit de toujours parcourir le sous-arbre gauche avant le droit, cela nous donne 3 méthodes.

3.2 Préfixe, infixe, postfixe

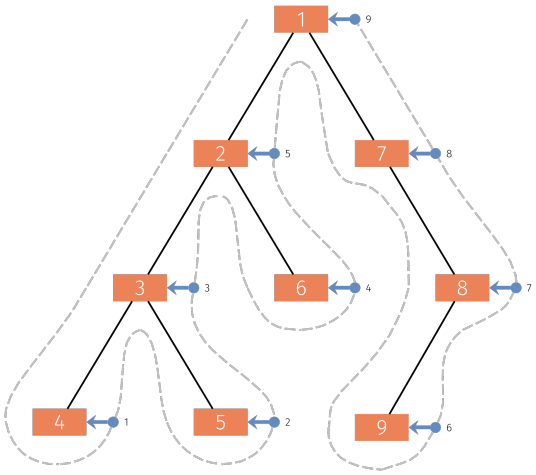
– **Parcours préfixe :** Traiter d'abord le nœud courant puis ensuite parcourir le sous-arbre gauche, puis le droit.



– **Parcours infixe :** Parcourir d'abord le sous-arbre gauche, puis traiter le nœud courant et ensuite parcourir le sous-arbre le droit.



– **Parcours postfixe ou suffixe :** Parcourir d'abord les sous-arbres gauche et droit puis traiter le nœud courant.



4 Exercices

Exercice 1 : construire des arbres

1. Dessiner tous les arbres binaires de taille 2.
2. Dessiner tous les arbres binaires de taille 3.

Exercice 2 : dénombrer des arbres

Sachant qu'il existe

- 1 arbre vide ;
- 1 arbre de taille 1 ;
- 2 arbres de taille 2 ;
- 5 arbres de taille 3 ;
- 14 arbres de taille 5.

Déterminer sans les construire le nombre d'arbres de taille 5.

Exercice 3 : une relation entre la hauteur et la taille d'un arbre

Soit h un entier naturel et A un arbre de hauteur h.

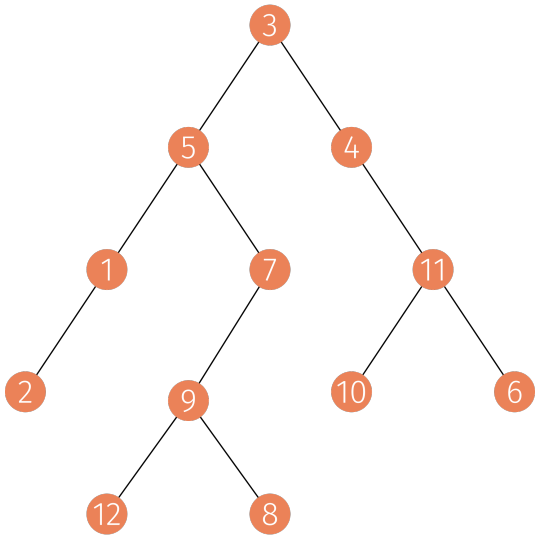
1. Combien, au minimum, A possède-t-il de nœuds ?

2. Combien, au maximum, A possède-t-il de nœuds ?

On en déduit l'encadrement suivant :
soit N la taille d'un arbre de hauteur h, alors on a

$$\leq N \leq$$

Exercice 4 : Parcours « à la main »

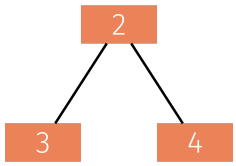


1. Écrire les valeurs de l'arbre dans l'ordre de son parcours préfixe.
2. Faire de même avec un parcours infixe.
3. Faire de même avec un parcours postfixe.

Exercice 5

1. Créer un fichier `Node.py` et implémenter la classe `Node` vue en cours.
2. Implémenter la méthode `__str__` qui utilise une sous-fonction récursive qui :
   - si on lui demande d'afficher `None` renvoie `''` ;
   - sinon (c'est qu'elle doit bien afficher un nœud) ouvre une parenthèse, affiche récursivement le sous-arbre gauche, puis affiche la valeur du nœud, le sous-arbre droit et enfin ferme la parenthèse.

sur l'arbre suivant



qui est créé par

```
a = Node(3)
b = Node(4)
c = Node(2, a, b)
```

`print(c)` devra renvoyer `'((3)2(4))'`

3. Implémenter la méthode d'instance `size` qui renvoie un `int` qui est la taille de l'arbre (s'aider du cours).
4. Comment trouver récursivement la hauteur d'un arbre ? Proposer une « méthode logique » et implémenter la méthode d'instance `height`, qui renvoie un `int` qui est la hauteur de l'arbre.
5. Implémenter la méthode d'instance `__eq__` qui renvoie `True` si deux arbres sont égaux et `False` sinon (trouver une méthode récursive).

Exercice 6 : Parcours

1. Ajouter à la classe `Node` une méthode d'instance `prefix` qui renvoie un `str` qui est la chaîne de caractères obtenue en concaténant toutes les valeurs des nœuds de l'arbre au cours de son parcours préfixe.
2. De même implémenter `infix` et `postfix`.