

Structures de données : les arbres

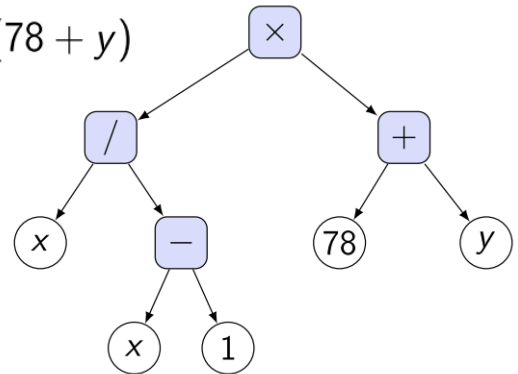
Résumé¹

Les arbres sont un nouveau type abstrait de données très utilisés dans beaucoup d'applications informatiques. Ils permettent d'effectuer des recherches de manière beaucoup plus efficaces que les listes chaînées vues précédemment.

Nous nous intéresserons particulièrement à des arbres particuliers appelés arbres binaires.

Puis dans un deuxième cours aux arbres binaires de recherche.

$$\frac{x}{(x-1)} \times (78 + y)$$



Sommaire

1 Les arbres : structure arborescente	2
1.1 Quelques exemples de données arborescentes	2
1.2 Quelques définitions relatives aux arbres.....	3
1.3 Définition récursive d'un arbre	3
2 Les arbres binaires	3
2.1 Représentation des arbres avec 3 puis 4 nœuds	3
2.2 Dénombrement du nombre d'arbres à n nœuds	4
2.3 Hauteur et taille d'un arbre.....	5
2.4 Information 'organisée' par un arbre.....	5
2.5 Quelques définitions	5
2.6 Intérêts des arbres.....	6
3 Parcours en profondeur d'arbres binaires	6
3.1 Introduction définition d'un parcours.....	6
3.2 Les trois type de parcours possible :	6
4 Parcours en largeur d'arbres binaires	7
4.1 Principe.....	7
4.2 Algorithme de parcours en largeur ou BFS Breadth First Search	8
5 Implémentation des arbres en Python	8
5.1 Construction d'arbres binaires	8
5.2 Impression des arbres.....	9
5.3 Parcours en profondeur	10
5.4 Parcours en profondeur.....	10



¹ La figure est issue du document arbre <http://www.dil.univ-mrs.fr/~gcolas/algo1.html>

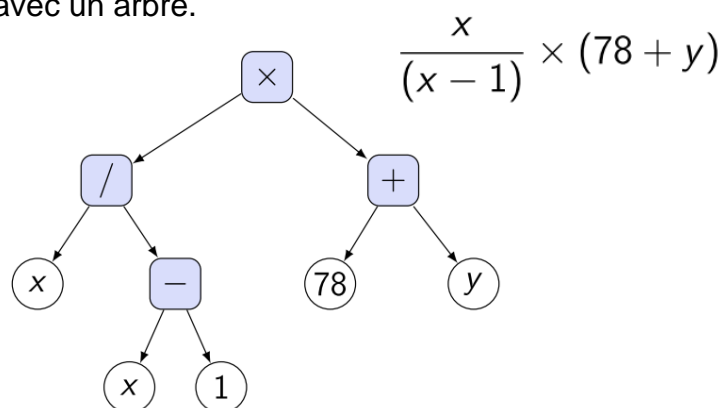
1 Les arbres : structure arborescente

Une structure arborescente est une représentation hiérarchique des données, cette représentation possède un point de départ unique puis se ramifie en branches successives. Il y a un nombre quelconque de ramifications qui se déploient à partir du point de départ appelé nœud racine. Attention les arbres en informatique sont représentés poussant vers le bas.

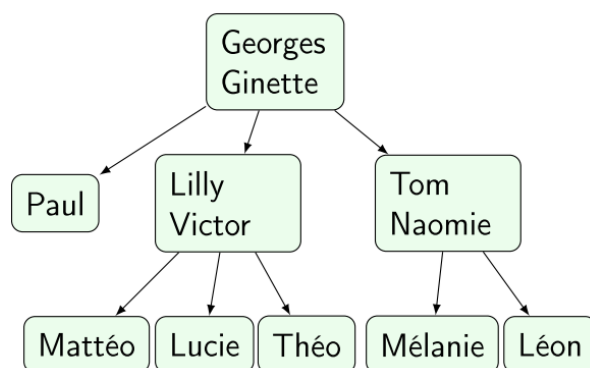
1.1 Quelques exemples de données arborescentes

Arbre syntaxique²

Représentation d'une expression arithmétique avec un arbre.



Arbre généalogique³

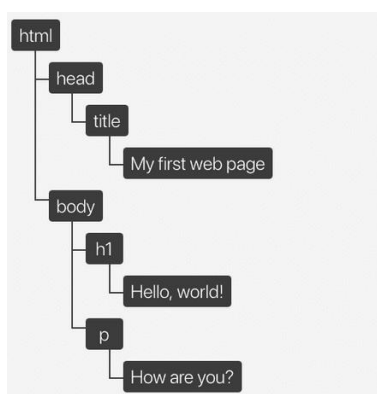


Le DOM Document Object Model⁴

Le DOM (Document Object Model) est une interface pour les pages web. C'est une API (Application Programming Interface) permettant aux programmes de lire et de manipuler le contenu de la page, sa structure et ses styles.

Le DOM est une représentation du document HTML source. Comme nous le verrons plus loin, il comporte quelques différences, mais il s'agit pour l'essentiel d'une conversion de la structure et du contenu du document HTML en un modèle objet utilisable par divers programmes.

Exemple :



```
<!doctype html>
<html lang="en">
  <head>
    <title>My first web page</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
    <p>How are you?</p>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="fr">
  <head>
    <meta charset="UTF-8" />
    <title>Hello HTML</title>
  </head>
  <body>
    <h1>Hello HTML</h1>
    <p>Bye now.</p>
  </body>
</html>
```

Q1. Représenter le DOM de la page HTML décrite ci-dessus :



² <http://www.dil.univ-mrs.fr/~gcolas/algo1.html> Pour les illustrations

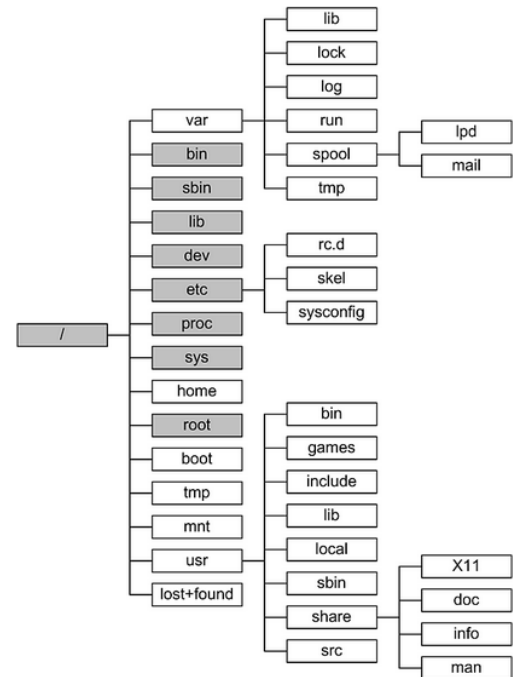
³ Ibid.

⁴ <https://la-cascade.io/le-dom-cest-quoi-exactement/>

Arborescence de fichiers

Présenté de façon arborescente, le système de fichiers Linux est une hiérarchie de répertoires ayant pour racine unique / (slash).

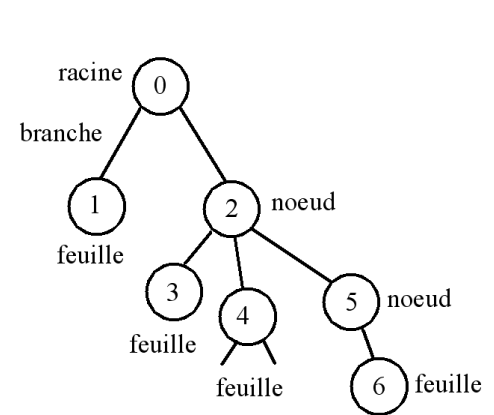
Lors de l'installation d'une distribution Linux, il est possible de créer, en plus de la partition principale contenant /, des partitions dédiées à certains répertoires de l'arborescence. Cependant, les répertoires indispensables au démarrage du système doivent être sur la même partition que / et ne peuvent donc pas être installés sur une partition séparée ; ces répertoires essentiels apparaissent en grisé dans le schéma suivant.



1.2 Quelques définitions relatives aux arbres

Définitions

- Un arbre peut être vide et ne contenir aucun nœud. \perp
- Un arbre non vide possède un nœud de départ appelé racine.
- Un nœud a des fils qui sont eux aussi des arbres.
- Si tous les fils d'un nœud sont vides, alors le nœud est qualifié de feuille.
- Les nœuds portent des valeurs, ce sont les données que l'on veut stocker.
- Si tous les nœuds de l'arbre ont n fils, alors l'arbre est dit n-aire.
- Un arbre dont tous les nœuds possèdent au maximum n fils a une arité de n.
- On appelle degré d'un nœud le nombre de fils que possède ce nœud.



Propriétés

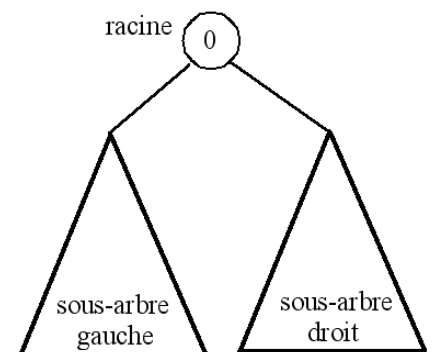
Chaque nœud à un père et un seul excepté le nœud racine qui n'a pas de père. Les feuilles d'un arbre sont les nœuds qui ne contiennent pas de fils.

1.3 Définition récursive d'un arbre

On peut définir n'importe quel arbre non vide comme étant un nœud qui possède deux sous arbres comme fils. Un sous arbre gauche et un sous arbre droit.

Un sous arbre peut éventuellement être vide.

Cette construction est récursive puisque pour tous les nœuds d'un arbre on peut utiliser cette représentation.



2 Les arbres binaires

Un arbre binaire est un arbre où tous les nœuds possèdent au plus deux fils.

2.1 Représentation des arbres avec 3 puis 4 nœuds

Q2. Représenter tous les arbres binaires ayant 3 nœuds.

Q3. Représenter tous les arbres binaires ayant 4 nœuds.



2.2 Dénombrement du nombre d'arbres à n nœuds

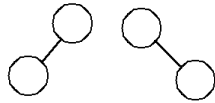
Principe du dénombrement

Le dénombrement de tous les arbres possédant n nœuds est donné par une suite de nombres dénommés les nombres de catalan ou C(n).

Pour démarrer la suite nous avons :

$$C(0) = 1 \quad C(2) = 2$$

$$C(1) = 1$$



Q4. Dénombrer tous les arbres possédant 4 nœuds.

Q5. Dénombrer tous les arbres possédant 5 nœuds.

Dénombrement de tous les arbres possédant 3 nœuds

	0		On enlève le nœud racine
0	2	}	On dénombre toutes les répartitions de nœuds possibles entre le sous arbre gauche et le sous arbre droit.
1	1		
2	0		

$$C(3) = C(0)*C(2) + C(1)*C(1) + C(2)*C(0)$$
$$C(3) = 2*1 + 1*1 + 2*1 = 5$$

Algorithme de calcul des nombres de Catalan

On considère un tableau C [] de N+1 éléments numérotés de 0 à N. L'algorithme ci-dessous permet de calculer la suite des nombres de Catalan. Ces nombres seront rangés dans le tableau C [] dans l'ordre. On a au départ C[0] = 1 puis par récurrence pour k variant de 0 à N-1:

Calcul des nombres de Catalan

$$C(0) = 1$$

$$C(N) = C(0) \cdot C(N-1) + C(1) \cdot C(N-2) + \dots + C(k) \cdot C(N-1-k) + \dots + C(N-1) \cdot C(0).$$

Voilà l'algorithme en pseudo code

```
C[ 0 ] ← 1
pour i de 1 à N
    cumul = 0
    pour k de 0 à i-1
        cumul ← cumul + C[ k ]*C[ i - 1 - k ]
    fin pour
    C[ i ] ← cumul
fin pour
afficher max
```

Script python

```
## Programme principal calcul des nombres de Catalan
try:
    N = 6
    liste_catalan = catalan(N)
    resultat_attendu = [1, 1, 2, 5, 14, 42, 132]
    if liste_catalan != resultat_attendu:
        raise ValueError
except ValueError:
    print("Le calcul est faux")
    print("Résultat attendu : ", resultat_attendu)
    print("Votre résultat : ", liste_catalan)
else:
    print("Le résultat est correct")
    print("Liste C(%2d) = " % (N), liste_catalan)
finally:
    print("Fin du traitement")
```



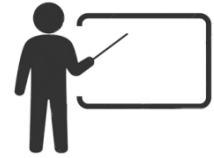
Votre fonction à réaliser

```
def catalan(N):
    '''
    Calcul de la suite de Catalan de n = 0 ... N
    En entrée le nombre N
    En sortie une liste C contenant les valeurs de la suite
    '''
    if N < 0:
        return None
    return C
```



Script_ARBRES_1. Compléter le script ci-dessous avec le code de la fonction catalan(N)

Script_ARBRES_1.py

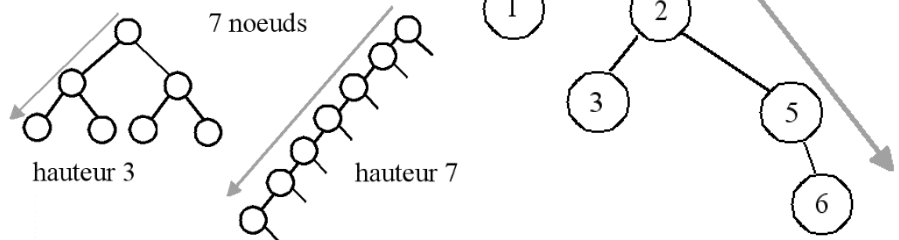


Q6. Expliquez le fonctionnement des instructions try - except - else - finally

2.3 Hauteur et taille d'un arbre

La hauteur d'un arbre est le plus grand nombre de nœuds obtenus en descendant depuis la racine jusqu'à chacune des feuilles, les deux extrémités comptent.

Une question se pose avec n nœuds quelle est la hauteur minimale et maximale ?



Nous voyons sur la figure que dans un cas extrême où tous les nœuds n d'un arbre n'ont qu'un seul fils la hauteur $h = n$.

Pour un arbre binaire dans le cas où tous les nœuds sont complets couche après couche la hauteur vérifie : $h \geq \log_2 n$.

La taille d'un arbre est le nombre de nœuds qui le compose.⁵

2.4 Information 'organisée' par un arbre

L'information organisée par un arbre est contenue dans les nœuds.

2.5 Quelques définitions⁶

Arbre binaire plein

Un arbre binaire est plein si chaque nœud a 0 ou 2 enfants. Nous pouvons également dire qu'un arbre binaire plein est un arbre binaire dans lequel tous les nœuds, à l'exception des feuilles, ont deux enfants.

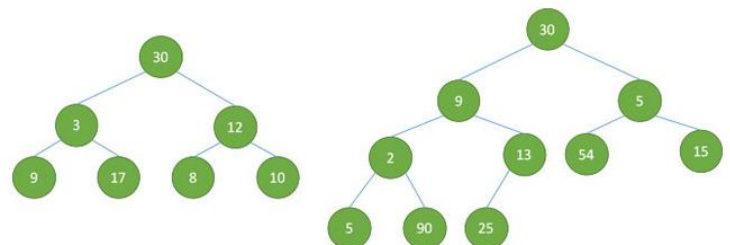
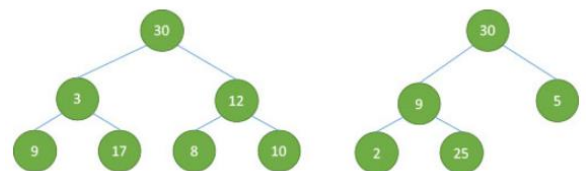
Dans un arbre binaire plein le nombre de feuille est égal au nombre de nœuds internes plus 1.

Arbre binaire complet

Un arbre binaire est complet si tous les niveaux sont complètement remplis, sauf peut-être le dernier niveau.



Et le dernier niveau a toutes les clés aussi gauches que possible.



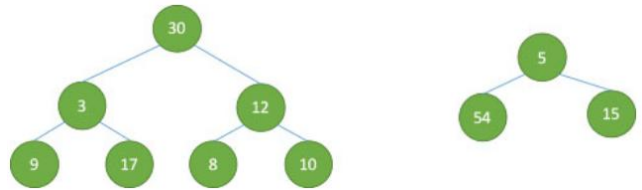
⁵ Attention ici dans certains documents la taille d'un arbre est le nombre total de nœuds obtenu sans compter les feuilles.

⁶ Les exemples sont issus de <https://developpement-informatique.com/article/172/types-darbre-binaire>

Arbre binaire parfait

Un arbre binaire est parfait si tous les nœuds internes ont deux enfants et toutes les feuilles sont au même niveau.

Un arbre binaire parfait de hauteur h a $2^h - 1$ nœud.

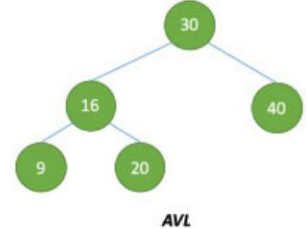


Arbre binaire équilibré

Un arbre binaire est équilibré si sa hauteur est $O(\log n)$, où n est le nombre de nœuds.

Arbre binaire équilibré AVL

C'est un arbre équilibré où la différence entre les hauteurs des sous-arbres gauche et droit soit de 1.



2.6 Intérêts des arbres

Le type abstrait arbres permet d'être efficace à la fois en insertion d'éléments et en recherche. Pour rappel :

	Insertion d'un élément	Recherche de l'ième élément
Pour les listes chaînées	$O(1)$	$O(n)$
Pour les listes	$O(n)$	$O(1)$

Ce sera l'objet de la suite de cette partie : Les arbres binaires de recherche.

3 Parcours en profondeur d'arbres binaires

3.1 Introduction définition d'un parcours

Parcourir un arbre consiste à cheminer de nœud en nœud à partir de la racine de telle manière que l'ensemble des nœuds soit parcourus. Pour chacun des nœuds il est possible de faire trois actions différentes :

- traiter le contenu du nœud : dans nos exemples le traitement consistera à afficher le nom du nœud,
- parcourir à gauche : on descend dans le sous arbre gauche si il existe,
- parcourir à droite : on descend dans le sous arbre de droite si il existe.

Le parcours en profondeur est bien adapté à un traitement par un algorithme récursif.

3.2 Les trois type de parcours possible⁷ :

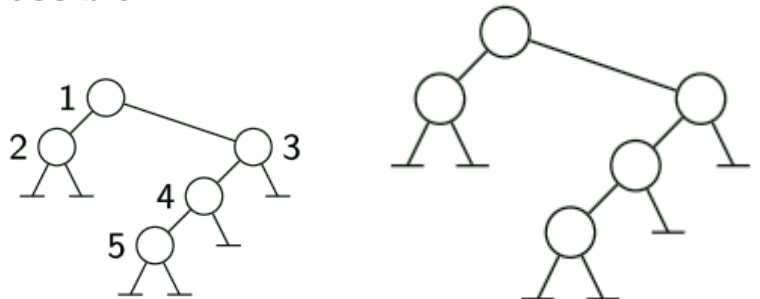
Pour l'arbre donné ci-contre :

Le parcours préfixe

Traitement du nœud

Parcours préfixe du sous arbre gauche

Parcours préfixe du sous arbre droit



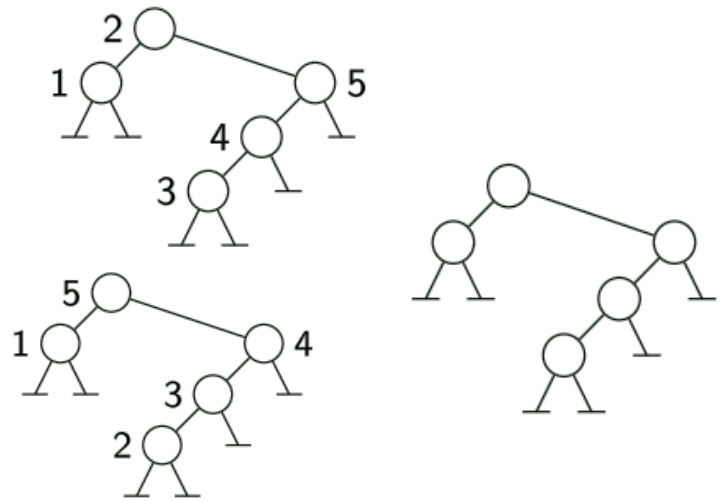
⁷ Les exemples sont issus de <https://www.lri.fr/~hivert/COURS/CFA-L3/06-Arbres.pdf>

Le parcours infixe

Parcours infixe du sous arbre gauche
Traitement du nœud
Parcours infixe du sous arbre droit

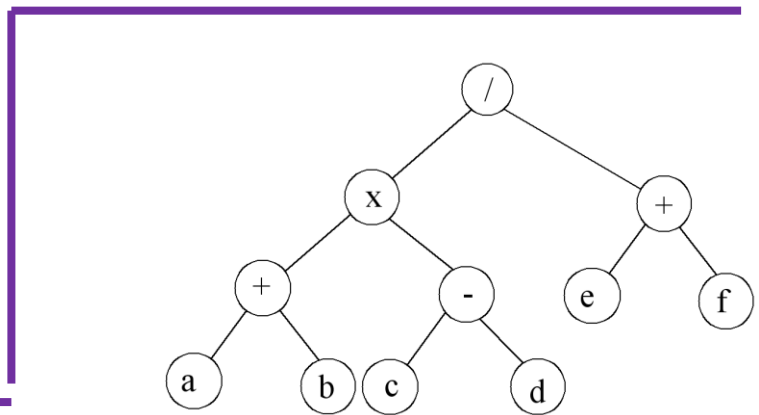
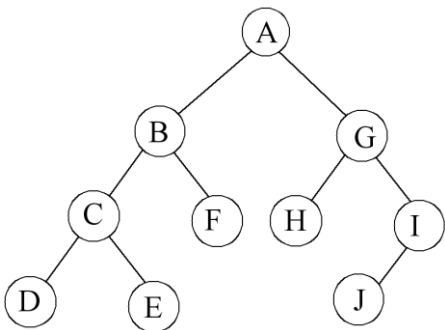
Le parcours postfixe

Parcours postfixe du sous arbre gauche
Parcours postfixe du sous arbre droit
Traitement du nœud



- Q7.** Donner le résultat du parcours Préfixe sur l'arbre ci-dessous.
- Q8.** Donner le résultat du parcours Infixe sur l'arbre ci-dessous.
- Q9.** Donner le résultat du parcours Posfixe sur l'arbre ci-dessous.

L'arbre binaire des exercices Q7-Q9



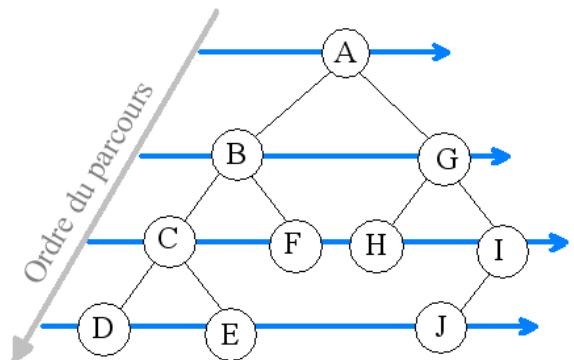
- Q10.** Reprendre les mêmes question avec l'arbre ci-dessus représentant une expression arithmétique.
- Q11.** Pour le parcours infixe de la question précédente on reprend le parcours avec ajout de parenthèses (ouvrante en rencontrant le nœud racine du sous arbre pour la première fois et fermante lorsqu'on le rencontre pour la dernière fois, avec exception sur les sous-arbres constitués d'une feuille). Que retrouve-t-on ?

4 Parcours en largeur d'arbres binaires

4.1 Principe

Le parcours en largeur d'un arbre consiste à descendre 'niveau' par 'niveau' jusqu'aux feuilles. Ce parcours n'est pas adapté à une démarche récursive.

De plus pour 'sauter' d'un sous arbre à un autre il nous faut mémoriser les nœuds à parcourir dans une logique de type FIFO, c'est donc une file qui va être utilisée.



4.2 Algorithme de parcours en largeur ou BFS *Breadth First Search*

Voilà un algorithme BFS de l'arbre A. Cet algorithme utilise un type abstrait de données 'File' muni des opérations : `enfiler(x,file)`, `s=defiler(file)`, et `estVide(file)`.

Pour l'arbre A on peut déterminer son noeud racine ainsi que les filsGauche et filsDroit.

```
/* On initialise file avec une File vide */
file ← File
/* On enfiler le premier noeud */
enfiler(racine(A),file)
/* On répète le traitement tant que la file n'est pas vide */
tant que la file n'est pas vide
    noeud ← defiler(file)
    /* Traiter noeud par exemple pour nous imprimer sa valeur */
    traiter(noeud)
    /* Si le fils gauche du noeud n'est pas vide on l'enfile */
    si filsGauche(noeud) existe
        enfiler(filsGauche(noeud),file)
    /* Si le fils droit du noeud n'est pas vide on l'enfile */
    si filsDroit(noeud) existe
        enfiler(filsDroit(noeud),file)
```

Les noeuds seront bien traités dans leur ordre d'arrivée dans la file donc par 'niveau'.

5 Implémentation des arbres en Python

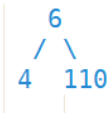
Il existe plusieurs méthodes pour coder des arbres binaires. Nous utiliserons une construction basée sur une classe Arbre définissant un noeud avec une valeur, un fils gauche et un fils droit.

5.1 Construction d'arbres binaires

La classe Arbre

```
class Noeud:
    def __init__(self, x, fils_gauche=None, fils_droit=None):
        self.val = x
        self.fg = fils_gauche
        self.fd = fils_droit
```

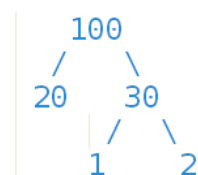
Création de l'arbre ci-dessous :



```
...
mon_arbre_bis = Noeud(6,Noeud(4),Noeud(110))
```

Création d'un premier arbre

Q12. Codez la création de l'arbre `mon_arbre` ci-contre :



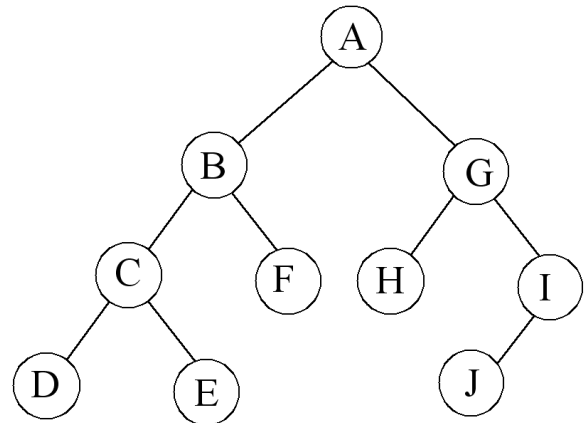
Création d'un nœud avec deux sous-arbres

```
def nouveau_noeud(x, arbre_gauche, arbre_droit):
    '''
    Créé un noeud de valeur x avec un sous arbre gauche
    et un sous arbre droit
    '''
    return Noeud(x, arbre_gauche, arbre_droit)
```

Q13. Fusionner les deux arbres `mon_arbre` et `mon_arbre_bis` dans un nœud de valeur '111'. L'arbre `mon_arbre` sera le fils gauche et l'arbre `mon_arbre_bis` sera le fils droit du nœud '111'.

Définition d'un arbre binaire plus complexe

Q14. Donner les instructions permettant de créer l'arbre binaire ci-contre :

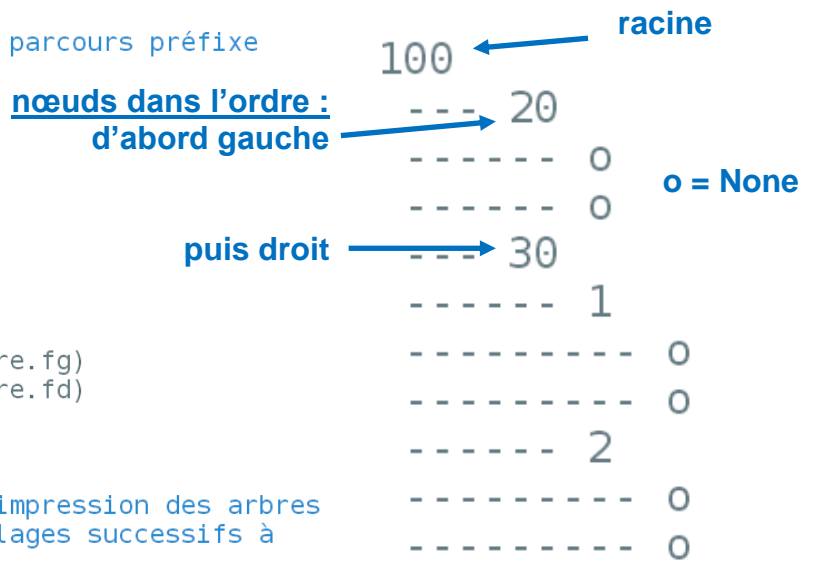


5.2 Impression des arbres


Pour pouvoir travailler sur nos arbres et vérifier le bon fonctionnement de nos opérations voilà un script python réalisant l'impression des arbres définis comme ci-dessus.

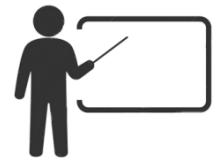
```
def imprime_arbre_niveau(niveau, arbre):
    '''
    Affiche les éléments de a dans un parcours préfixe
    Traiter la racine d'abord
    '''
    decalage = '---'*niveau
    if arbre is None:
        print(' ', decalage, 'o')
        return
    if niveau==0:
        print(decalage, arbre.val)
    else:
        print(' ', decalage, arbre.val)
    imprime_arbre_niveau(niveau+1, arbre.fg)
    imprime_arbre_niveau(niveau+1, arbre.fd)

def imprime_arbre(arbre):
    '''
    Appel de la fonction récursive d'impression des arbres
    Le niveau 0 sert à gérer les décalages successifs à
    l'affichage
    '''
    imprime_arbre_niveau(0, arbre)
```

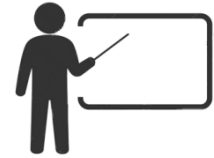




Script_ARBRES_2. Compléter le script ci-dessous pour créer et afficher les arbres.
 Script_ARBRES_2.py



Compléter le script ARBRES_2 avec les fonctions calculant la taille et la hauteur de l'arbre. On utilisera une implémentation récursive.




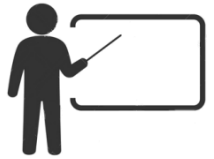
5.3 Parcours en profondeur

Voilà le code de la fonction parcours préfixe :

```
def parcours_prefixe(arbre):
    """
    Affiche les éléments de a dans un parcours préfixe
    Traiter la racine d'abord
    """
    if arbre is None:
        return
    print(arbre.val)
    parcours_prefixe(arbre.fg)
    parcours_prefixe(arbre.fd)
```



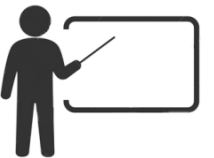
Script_ARBRES_3. En vous inspirant de la fonction parcours préfixe ci-dessus, Compléter dans le script les fonctions parcours infixe et postfixe.  Script_ARBRES_3.py



5.4 Parcours en profondeur



Script_ARBRES_4. Compléter la fonction parcours_BFS dans le script  Script_ARBRES_4.py



Compléter la fonction du parcours_BFS(arbre)

```
def parcours_BFS(arbre):
    """
    Parcours BFS de l'arbre
    Utilisation d'une file.
    A compléter
    """
    file = File()
```

