Les Arbres

I- Terminologie de base

1- Définition :

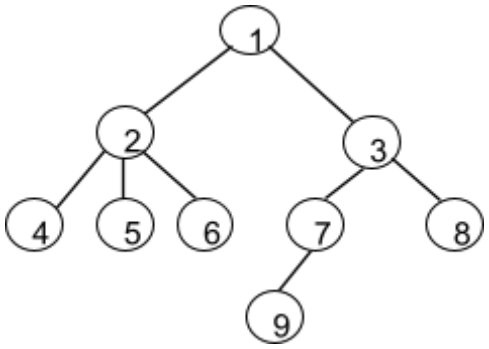
Un arbre est un ensemble d'éléments appelés **nœuds** reliés par des **arcs**.

Il existe une relation de parenté entre les nœuds. Un nœud **père** est situé au dessus de ses nœuds **fils**.

Un **père** est relié à ses **fils** par des **arcs**.

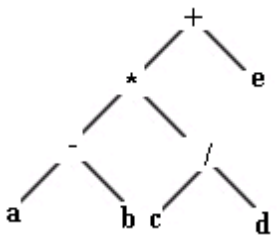
Le nœud le plus haut placé dont dérivent tous les autres nœuds est appelé la **racine**.

Exemples d'arbre :



- La **racine** est le nœud **1**.
- Le **père** est placé au dessus des **fils**.
- Le segment reliant un fils à son père est un **arc**.

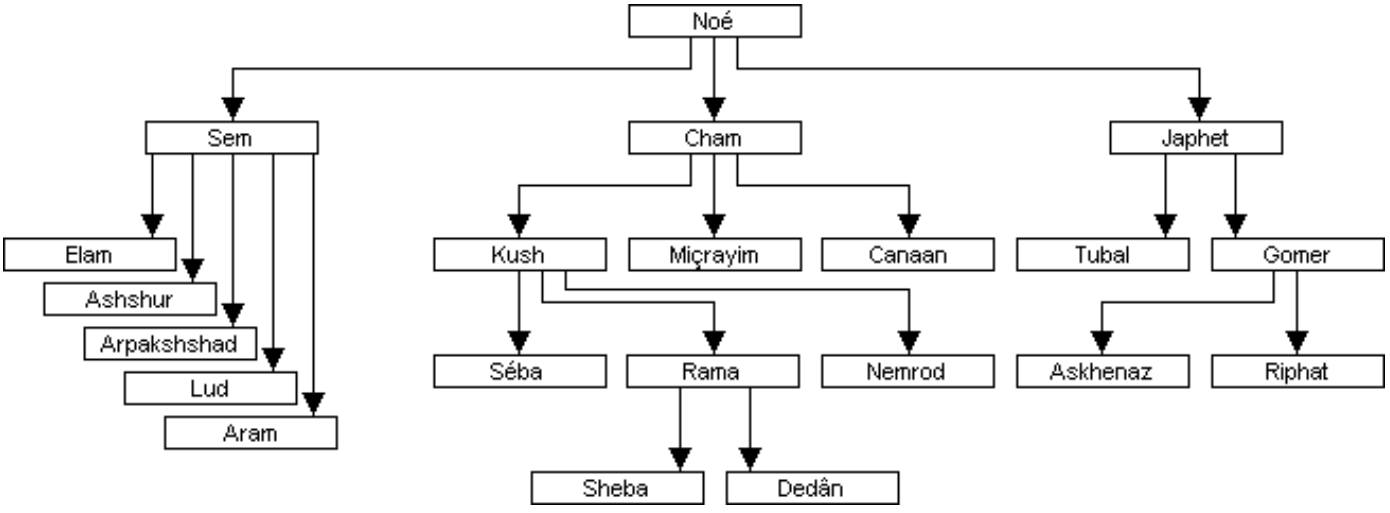
- **Arbre représentant une expression arithmétique**



((a-b)*(c/d))+e

- **Arbre généalogique :**

L'arbre généalogique suivant représente quelques descendants mâles de Noé



2- Définition récursive d'un arbre:

Un **nœud unique** est un arbre. Dans ce cas il est aussi la **racine** de l'arbre.
Si **n** est un nœud et $\mathbf{A_1, A_2, \ldots A_k}$ sont des arbres de racines respectives $\mathbf{n_1, n_2, \ldots, n_k}$ alors on peut construire un arbre en associant comme **père** unique aux nœuds $\mathbf{n_1, n_2, \ldots, n_k}$, le nœud **n**. dans ce nouvel arbre, **n** est la racine et $\mathbf{A_1, A_2, \ldots A_k}$ sont les sous **arbres** de cette racine. Les nœuds $\mathbf{n_1, n_2, \ldots, n_k}$, sont appelés les **fils** du nœud **n**.

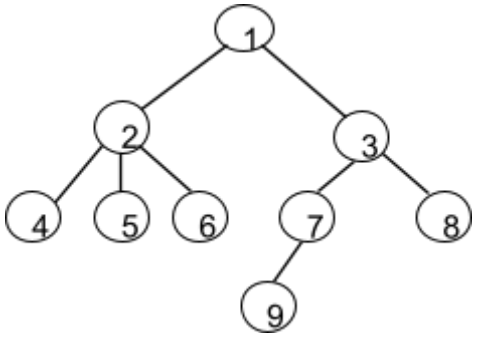
3- Chemin entre nœuds.

Si $\mathbf{n_1, n_2, \ldots, n_j}$, est une suite de nœuds telle que $\mathbf{n_i}$ est le père de $\mathbf{n_{i+1}}$ pour **i** allant de **1** à **j-1**, alors cette suite est appelée **chemin entre le nœud** $\mathbf{n_1}$ et le nœud $\mathbf{n_j}$.

La **longueur** du chemin est le nombre de **nœuds - 1** (=nombre d'arcs).

4- Ascendants et descendants

- S'il existe un chemin entre les nœuds **a** et **b** alors **a** est dit **ascendant** de **b** (ou **ancêtre**), **b** et dit **descendant** de **a**.
- Tout **nœud** est un de ses **descendants** et un de ses **ascendants**.
- Tout **nœud ascendant** ou **descendant** d'un nœud différent de lui-même est un **ascendant** ou **descendant propre**.
- Dans un arbre seule la **racine** n'a pas **d'ascendant propre**.
- Un nœud sans descendant propre est appelé **feuille**.
- Des nœuds ayant le même père sont appelés **frères**.
- Un nœud qui n'est pas une feuille est un **nœud interne**.
- Un arbre sans nœud est un **arbre vide**.



- 1, 3, 7, 9 est un chemin de **longueur 3**.
- 1, 3, 6 n'est pas un chemin
- Les **descendants propres** de 3 sont : 7, 8 et 9
- Les **fils** de 3 sont : 7 et 8
- Le **père** de 3 est : 1
- Les **ascendants propres** de 7 sont : 3 et 1
- Les **feuilles** de l'arbre sont : 4, 5, 6, 8 et 9
- 4, 5 et 6 sont **frères**

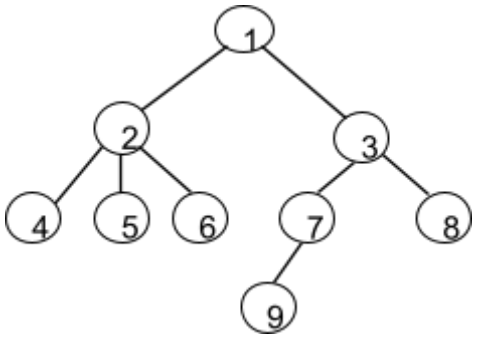
5- Hauteur et profondeur

La **hauteur** d'un nœud dans un arbre est la longueur du plus long chemin que l'on peut mener entre ce nœud et une feuille de l'arbre.

La **hauteur** d'un arbre est la hauteur de sa racine.

La **profondeur** d'un nœud est la longueur du chemin entre la racine et ce nœud.

Dans l'exemple :

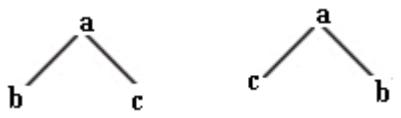


- La **hauteur** de 3 est : 2
- La **profondeur** de 3 est : 1
- La **hauteur** de l'arbre est : 3

6- **Ordre des nœuds d'un arbre.**

Les fils d'un arbre sont habituellement ordonnés de gauche à droite.

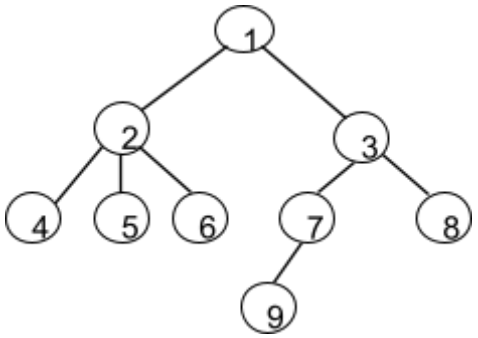
Les deux arbres ci-dessous sont différents :



L'ordre gauche-droit peut être prolongé pour comparer des nœuds qui ne sont ni ascendants ni descendants.

Si **b** et **c** sont deux frères, que **b** et à gauche de **c**, tout descendant de **b** est à gauche de tout descendant de **c**.

Dans l'exemple :



- 2 est à gauche de 3
- 5 est à gauche de 7
- 4 est à gauche de 9
- 3 n'est ni à gauche ni à droite de 9.

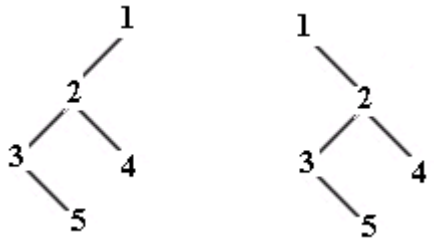
II- Arbres binaires (AB)

1- **Définition :**

Un arbre binaire est un arbre dont chaque nœud a au **maximum deux fils**.

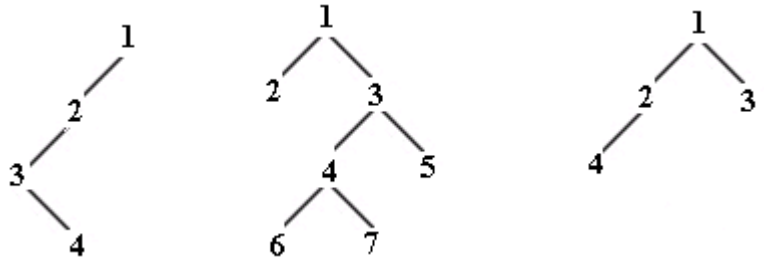
Les arbres binaires distinguent le **fils gauche** du **fils droit**.

Ces deux arbres binaires sont différents :



Dans un arbre binaire **dégénéré** ou **filiforme** chaque nœud a un seul fils.

Un arbre **binaire complet** est un arbre binaire dont chaque nœud a **deux fils** ou **est une feuille**.



Filiforme *Complet* *pas complet*

2- **Parcours d'un arbre binaire.**

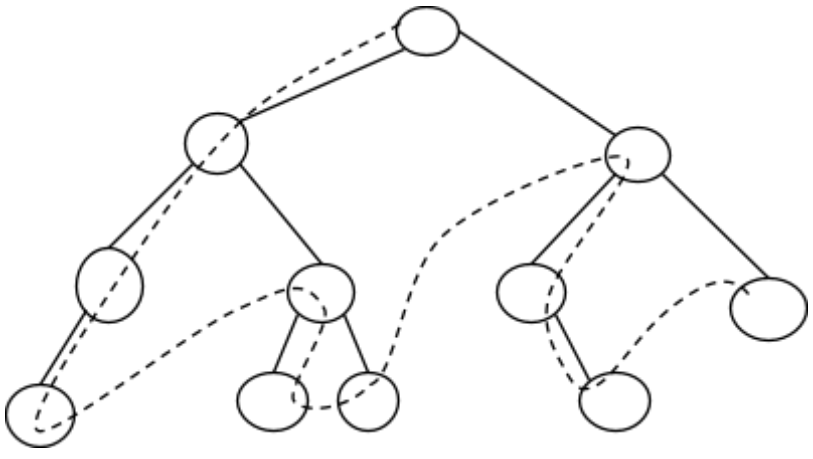
❖ Parcours en **profondeur d'bord**

Le parcours le plus simple à programmer est le parcours dit en **profondeur d'bord**.

a- **Parcours préfixe.**

Le parcours **préfixe** est décrit récursivement :

- Visiter la **racine**
- Visiter le sous-arbre **gauche** en parcours **préfixe**
- Visiter le sous-arbre **droit** en parcours **préfixe**



Un affichage **préfixe** donnerait :

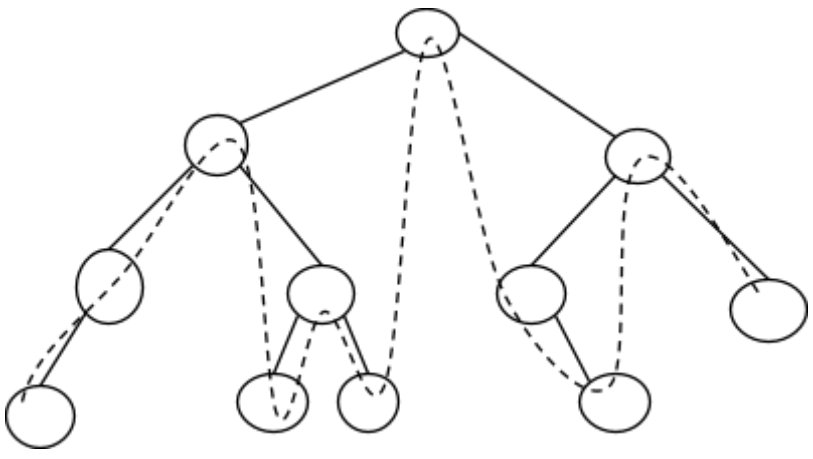
12 - 1 - 91 - 32- 67 – 45- 50 - 7- 61 - 40 - 82

```
affichagePréfixe(a) :
    Si non Vide(a) :
        afficher(val(a))
        affichagePréfixe (filsGauche(a))
        affichagePréfixe (filsDroit(a))
```

b- **Parcours infixe.**

Le parcours **infixe** est décrit récursivement :

- Visiter le sous-arbre **gauche** en parcours **infixe**
- Visiter la **racine**
- Visiter le sous-arbre **droit** en parcours **infixe**



Un affichage **infixe** donnerait :

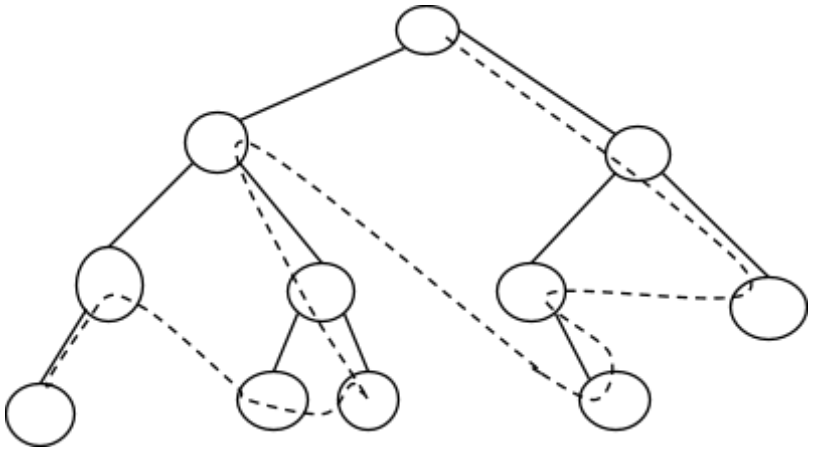
32 - 91 - 1 - 45 - 6 7 - 50 - 12 - 61 - 40 - 7 - 82

```
affichageInfixe(a) :
    Si non Vide(a) :
        affichageInfixe (filsGauche(a))
        afficher(val(a))
        affichageInfixe (filsDroit(a))
```

c- **Parcours postfixe.**

Le parcours **postfixe** est décrit récursivement :

- Visiter le sous-arbre **gauche** en parcours **postfixe**
- Visiter le sous-arbre **droit** en parcours **postfixe**
- Visiter la **racine**



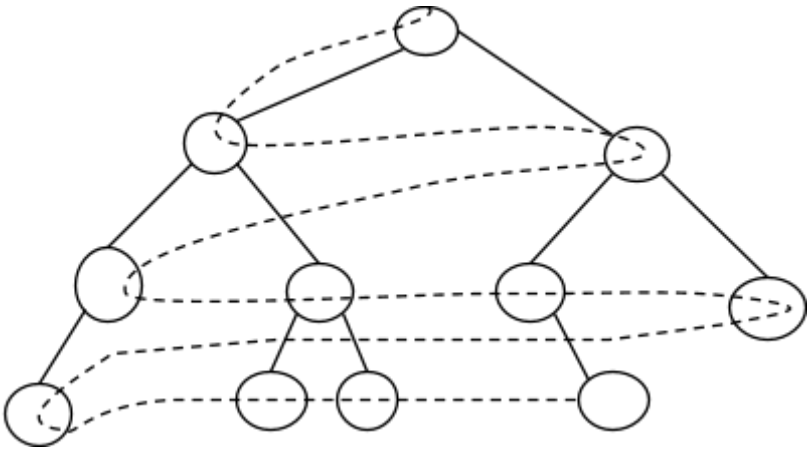
Un affichage **postfixe** donnerait :

32- 91 - 45 - 50 - 67 - 1 - 40 - 61 - 82 - 7 - 12

```
affichagePostfixe(a) :
    Si non vide(a) :
         affichagePostfixe (filsGauche(a))
         affichagePostfixe (filsDroit(a))
         afficher(val(a))
```

❖ **Parcours en largeur.**

- visite les nœuds **niveau** par **niveau** depuis la **racine**:
- Peut être décrit facilement en utilisant une **File** .



Un affichage **en largeur** donnerait :

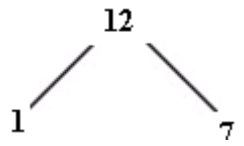
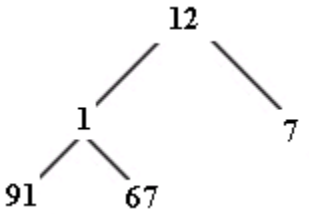
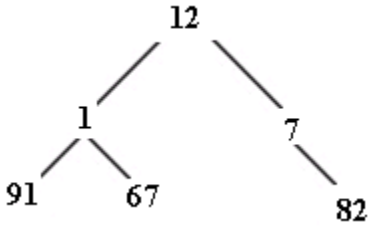
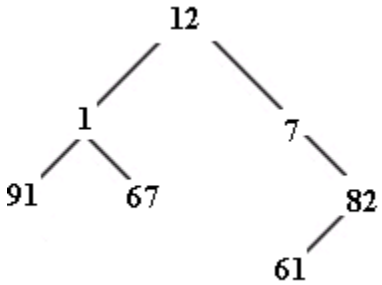
12 - 1 - 7- 91- 67- 61- 82- 32 - 45 - 50 - 40

```
affichageLargeur(a):
     F : File              #File FIFO
     Enfiler(a,F)          #enfiler la racine a dans la file F
     tantque non vide(F):
          n=Défiler(F)
          si non vide(n):
               Afficher(val(n))
               Enfiler(filsGauche(n),F) #enfiler le fils gauche de n dans
F
               Enfiler(filsDroit(n),F)  #enfiler le fils droit de n dans F
```

3- **Mise en œuvre des arbres binaires.**

En **Python**, on peut représenter un **arbre vide** par une **liste vide []** et un arbre **non vide** par une liste comprenant **3** éléments **[clé , filsGauche , filsDroit]**.

Exemple :

A= [] #A est un arbre vide	
A= [12,[],[]] #A est un arbre qui contient un seul nœud (12)	12
A=[12, [1, [], []] , [7, [], []]]	
A=[12, [1, [91,[],[]], [67,[],[]]] , [7, [], []]]	
A=[12, [1, [91,[],[]], [67,[],[]]] , [7, [], [82,[],[]]]]	
A=[12, [1, [91,[],[]], [67,[],[]]] , [7, [], [82,[61,[],[]] ,[]]]]	

❖ **Les fonctions de base sur la manipulation des arbres binaires.**

#Fonction qui détermine si un arbre a est vide ou non.

```
def vide(a):
    return a==[]
```

#Fonction qui retourne la valeur de la racine d'un arbre (étiquette).

```
def val(a):
   if a!=[]:
      return a[0]
   else:
      return None
```

#Fonction qui retourne le fils gauche de l'arbre a .

```
def filsGauche(a):
   if not vide(a):
      return a[1]
   else:
     return []
```

#Fonction qui retourne le fils Droit de l'arbre a .

```
def filsDroit(a):
    if not vide(a):
       return a[2]
    else:
       return []
```

#Affichage préfixe d'un arbre

```
def prefixe(a):
     if not Vide(a):
          print (val(a), end=' ')
          prefixe(filsGauche(a))
          prefixe(filsDroit(a))
```

#Affichage préfixe en utilisant un traitement itératif (en utilisant une Pile)

```
def prefixeIter(a):
     P=[]            #Pile vide
     P.append(a)     #empiler la racine dans la pile P
     while not vide(P):
          n=P.pop() #dépiler p et récupérer la tète (nœud n)
          print(val(n),end=' ')
          if not vide(filsDroit(n)) :
               P.append(filsDroit(n))    #empiler le fils droit de n dans P
          if not vide(filsGauche(n)) :
               P.append(filsGauche(n))   #empiler le fils gauche de n dans P
     print()
```

#Affichage infixe d'un arbre

```
def infixe(a):
     if not vide(a):
          infixe(filsGauche(a))
          print (val(a), end=' ')
          infixe(filsDroit(a))
```

#Affichage postfixe d'un arbre

```
def postfixe(a):
     if not vide(a):
          postfixe(filsGauche(a))
          postfixe(filsDroit(a))
          print (val(a), end=' ')
```

#parcourir un arbre en largeur

```
def parcoursLargeur(a):
     F=[] #File FIFO
     F.append(a) #enfiler la racine a dans la file F
     while not vide(F):
          n=F[0] ; F=F[1:] #défiler F (récupérer la tète)
          print(val(n),end=' ')
          if not vide(filsGauche(n))):
             F.append(filsGauche(n))  #enfiler le fils gauche de n dans F
          if not vide(filsDroit(n))):
               F.append(filsDroit(n)) #enfiler le fils droit de n dans F
     print()
```

#déterminer si un nœud est une feuille

```
def estFeuille(a):
     if vide(a) : return False
     return a[1]==[] and a[2]==[]
```

#hauteur d'un arbre

- un arbre vide est de hauteur : 0
- un arbre qui ne contient qu'un seul nœud (racine) est de hauteur : 0
- un arbre non vide et qui contient plus qu'un nœud a pour hauteur : 1 + la hauteur maximale entre ses fils.

```
def hauteur(a):
     if vide(a)or estFeuille(a):
          return 0
     else:
          return 1 + max(hauteur(filsGauche(a)),hauteur(filsDroit(a)))
```

#calculer le nombre de nœuds d'un arbre

```
def nombreNoeuds(a):
  if vide(a):
          return 0
  else:
    return 1+ nombreNoeuds(filsGauche(a)) + nombreNoeuds(filsDroit(a))
```

#calculer le nombre de feuilles dans un arbre

```
def nombreFeuilles(a):
  if vide(a):
     return 0
  elif estFeuille(a):
     return 1
  else:
     return nombreFeuilles(filsGauche(a))+ nombreFeuilles(filsDroit(a))
```

#déterminer si un nœud est un nœud interne

```
def estNoeudInterne(a):
     if vide(a) : return False
     return not estFeuille(a)
```

#calculer le nombre de nœuds internes d'un arbre

```
def nombreNoeudInternes(a):
     if vide(a):
          return 0
     elif estFeuille(a):
          return 0
     else:
          return 1+ nombreNoeudInternes(filsGauche(a)) +
nombreNoeudInternes(filsDroit(a))
```

#fonction de recherche dans un arbre binaire quelconque

```
def existe(a, x):
     if vide(a):
          return False
     else:
          if val(a) ==x:
               return True
          else:
               return existe(filsGauche(a),x) or existe(filsDroit(a),
x)
```

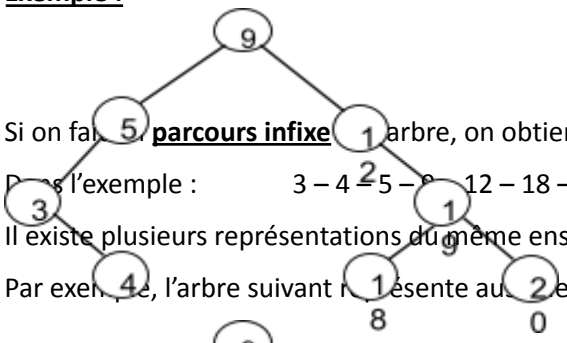
III- Arbres binaires de recherche (ABR)

1- Définition

Un **arbre binaire de recherche** (**ABR**) est un arbre étiqueté tel que pour tout nœud **n** :

- Tout nœud du sous arbre **gauche** a une valeur **inférieure ou égale** à la valeur de **n**.
- Tout nœud du sous arbre **droit** a une valeur **supérieure** à la valeur de **n**.

Exemple :

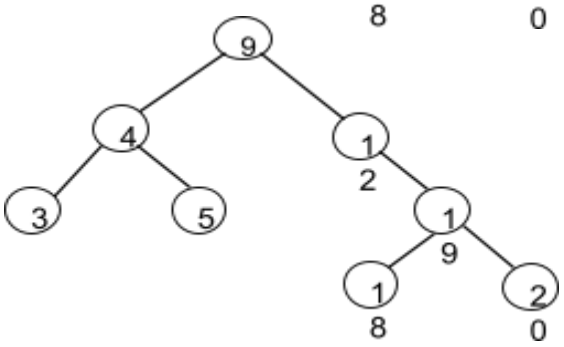


Si on fait un **parcours infixe** de l'arbre, on obtient la liste des valeurs des nœuds triée en ordre croissant :

Dans l'exemple : 3 – 4 – 5 – 9 – 12 – 18 – 19 – 20

Il existe plusieurs représentations du même ensemble d'éléments par des arbres binaires de recherche.

Par exemple, l'arbre suivant représente aussi le même ensemble de chiffres :



2- **Recherche dan un ABR**

a- **Recherche d'un élément.**

Pour rechercher un élément x :

- On compare l'élément x à la valeur de la racine.
- S'il est égale l recherche est terminée.
- S'il est plus petit on poursuit dans le sous arbre gauche
- S'il est plus grand on poursuit dans le sous arbre droit
- Si le sous arbre dans le quel on cherche est vide c'est que l'élément n'appartient pas à l'ensemble.

La fonction s'écrit comme suit de façon **itérative** :

```
def rechercheIter(v,a):
     while not vide(a) and v!=val(a):
          if v< val(a):
               a=filsGauche(a)
          else:
               a=filsDroit(a)

     if vide(a): return False
     return True
```

On peut aussi l'écrire d'une façon **récursive**

```
def rechercheRec(v,a):
     if vide(a):
          return False
     if v==val(a):
          return True
     if v<val(a):
          return rechercheRec(v,filsGauche(a))
     else:
          return rechercheRec(v,filsDroit(a))
```

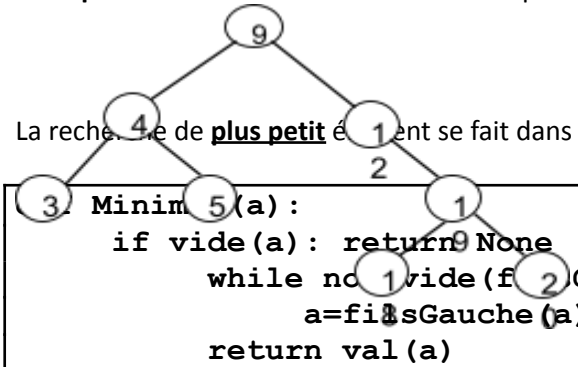
b- **Recherche du plus grand élément.**

Le plus grand élément est toujours dans le fils droit puisque le fils droit contient tous les éléments plus grands que le nœud.

S'il n'y a pas de fils droit, le plus grand élément est la valeur du nœud racine.

```
def Maximum(a):
     if vide(a): return None
     while not vide(filsDroit(a)):
          a=filsDroit(a);
     return val(a)
```

Exemple : sur l'arbre suivant rechercher le plus grand élément.



La recherche de **plus petit** élément se fait dans le **fils gauche** de l'arbre.

```
def Minimum(a):
     if vide(a): return None
          while not vide(filsGauche(a)):
               a=filsGauche(a);
          return val(a)
```

3- **Insertion dans un ABR**

```
def insertion(v,a):
     if vide(a):
          a+= [v,[],[]]
     else:
          if v<=val(a):
               insertion(v,filsGauche(a))
          else:
               insertion(v,filsDroit(a))
```