Les arbres

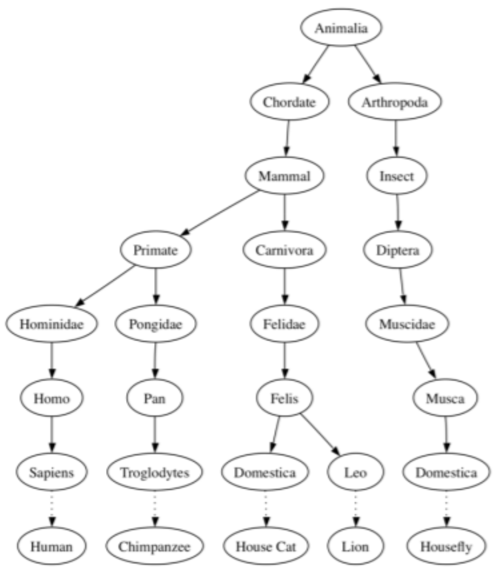
1 Les arbres

1.1 Introduction

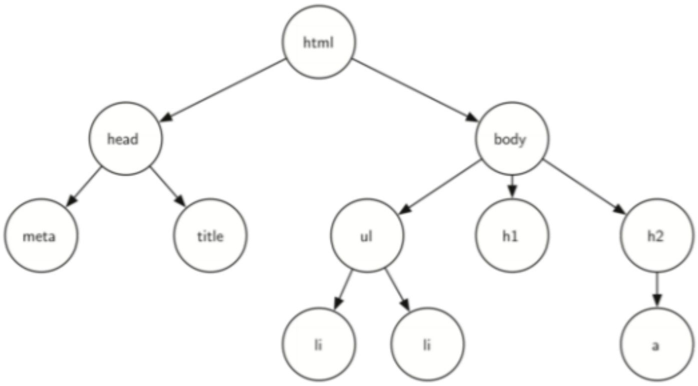
Il n'existe pas qu'une façon linéaire de représenter les données, comme les listes, les tableaux, les dictionnaires, les piles et les files. Nous pouvons également structurer les données de façon hiérarchique.

1.2 Exemples de situations où l'on rencontre des arbres

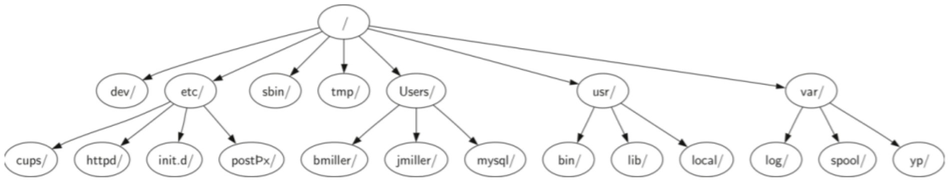
- En biologie



- Les balises d'une page web



- Les dossiers d'un ordinateur



1.3 Définition

Définition 9.1

Un **arbre** est une structure de données constituée de nœuds, qui peuvent avoir des enfants (et qui sont aussi des nœuds)

Définition 9.2

Le sommet de l'arbre est appelé **racine**.

Définition 9.3

Un nœud qui ne possède pas d'enfant est appelé une **feuille**.

Définition 9.4

Les nœuds autres que la racine et les feuilles sont appelés **nœuds internes**.

Définition 9.5

Une **branche** est une suite finie de nœuds consécutifs de la racine vers une feuille.

Remarque

Un arbre a donc autant de feuilles que de branches !

Définition 9.6

l'**arité** d'un arbre est le nombre maximal d'enfants qu'un nœud peut avoir.

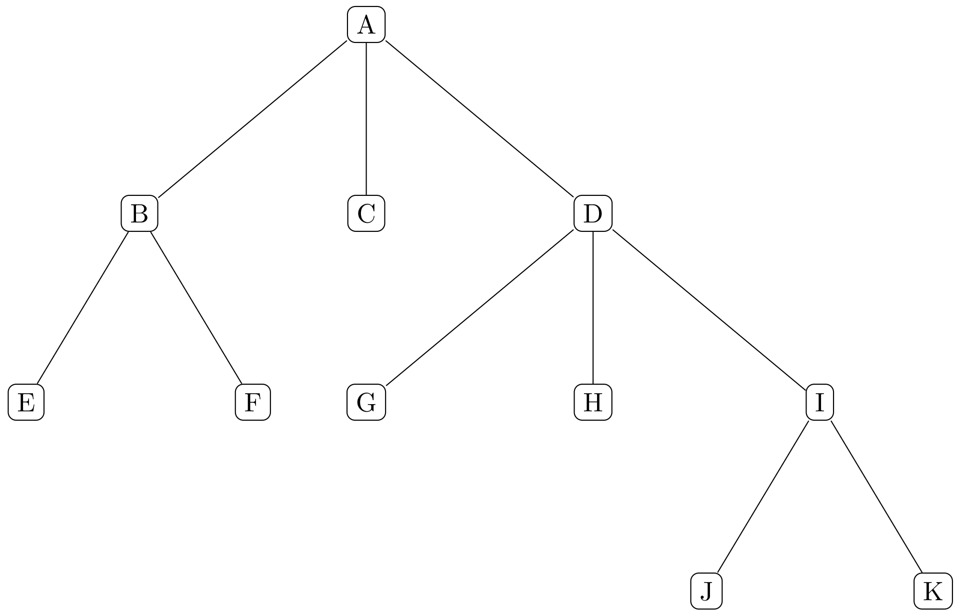
Définition 9.7

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

Définition 9.8

La **hauteur** d'un arbre est la profondeur à laquelle il faut descendre pour trouver la feuille la plus éloignée de la racine.

Exercice 9.1
On considère l'arbre ci-dessous.



Répondre aux questions suivantes :

- Nombre de nœuds ?
- Nombre de racine ? Laquelle ?
- Nombre de feuilles ? Lesquelles ?
- Nombre de branches ?
- Nombre de nœuds internes ?
- Quel est son arité ?
- Quelle est sa taille ?
- Quelle est sa hauteur ?

Remarque

En informatique, les arbres poussent vers le bas :-)

1.4 Représentation en Python d'un arbre, appelé aussi *arborescence*

1.4.1 Avec une classe

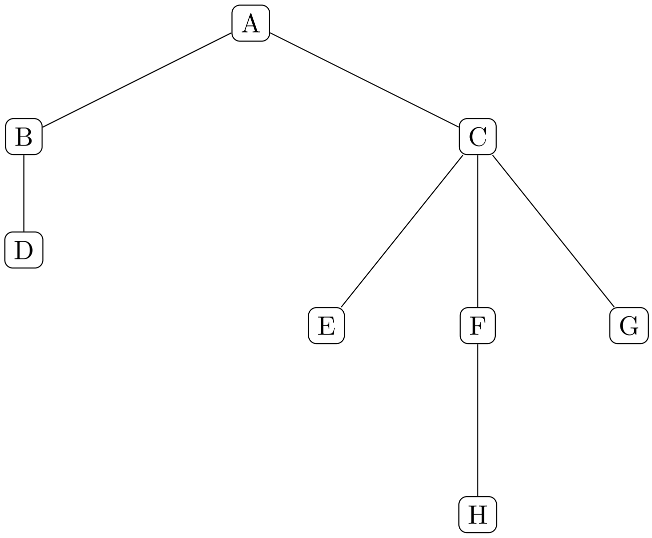
Pour représenter un arbre (une arborescence) en Python, on peut utiliser des objets, comme pour les listes chaînées.

L'objet de la classe contient deux attributs : un attribut valeur (dans lequel on stocke une valeur quelconque, appelée *étiquette* et un attribut fils dans lequel on stocke les fils sous la forme d'un tableau.

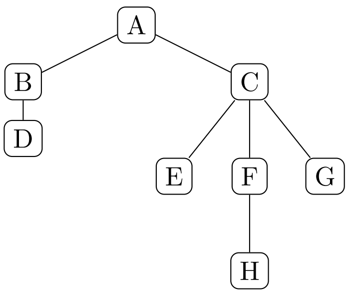
Exercice 9.2
Construire la classe Noeud. Pour les feuilles, on mettra [] pour le fils***

Exercice 9.3
Construire l'arbre donné en exemple plus haut.

Exercice 9.4
Construire l'arbre ci-dessous :



Exercice 9.5
Créer une fonction récursive **represente(arbre,p=0)** qui permet un affichage d'un arbre comme ceci :




```
 A
- B
-- D
- C
-- E
-- F
--- H
-- G
```

Représentation possible d'un arbre. Les tirets indiquent la profondeur
***

1.4.2 Avec un dictionnaire

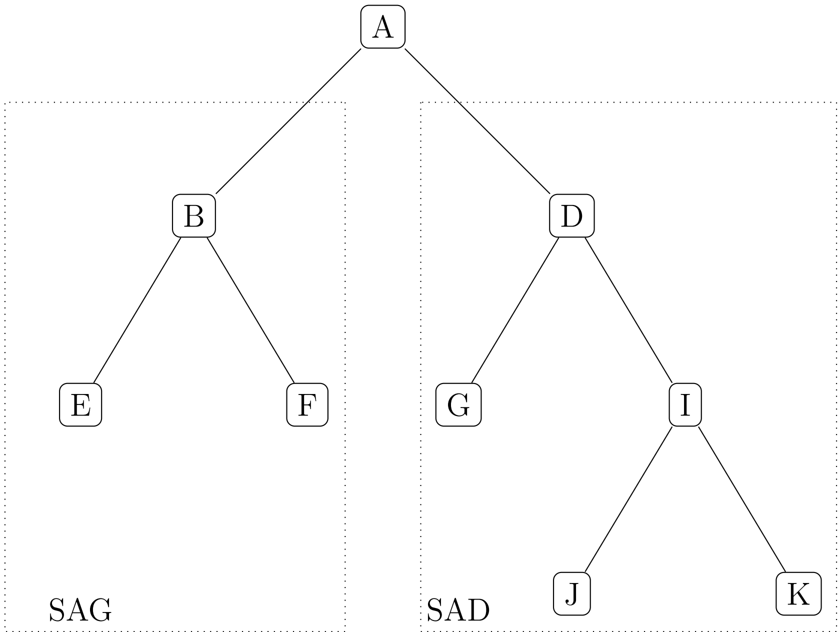
2 Les arbres binaires

2.1 Définition

Définition 9.9

Un arbre dont l'arité est 2 est un **arbre binaire**

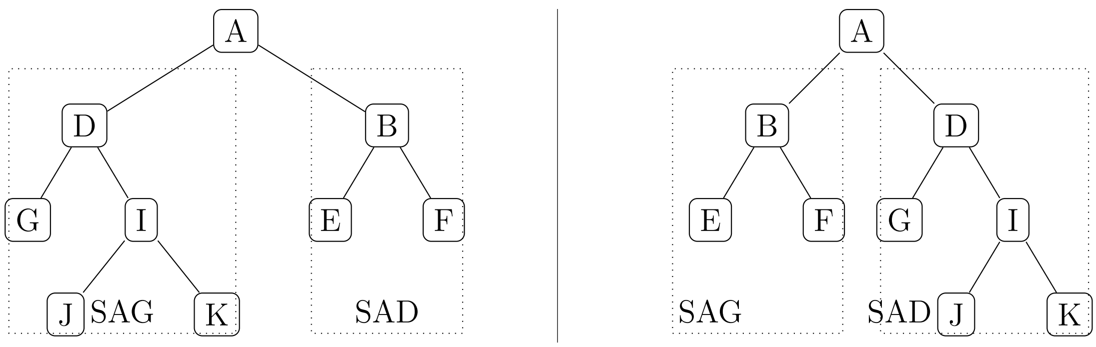
Les arbres binaires sont donc des arbres ou chaque nœud peut donner 0,1 ou 2 enfants.



On distingue généralement à partir du nœud racine 2 sous-arbres disjoints : Le sous-arbre gauche de l'arbre binaire (SAG) et le sous-arbre droit de l'arbre binaire (SAD).

Remarque

⚠ De ce fait, ces deux arbres ne sont pas identiques :



Exercice 9.6

Dessiner tous les arbres binaires possédant 3 nœuds.

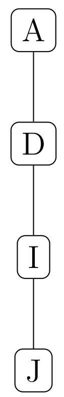
Exercice 9.7
Dessiner tous les arbres binaires possédant 4 nœuds.

Exercice 9.8
Sachant qu'il y a 1 arbre binaire vide, 1 arbre binaire contenant 1 nœud, 2 arbres binaires contennant 2 nœuds, 5 arbres binaires contenant 3 nœuds et 14 arbres binaires contenant 4 nœuds, calculer le nombre d'arbres binaires contenant 5 nœuds.
⚠ On cherche seulement ici à les dénombrer.***

2.2 Cas particuliers

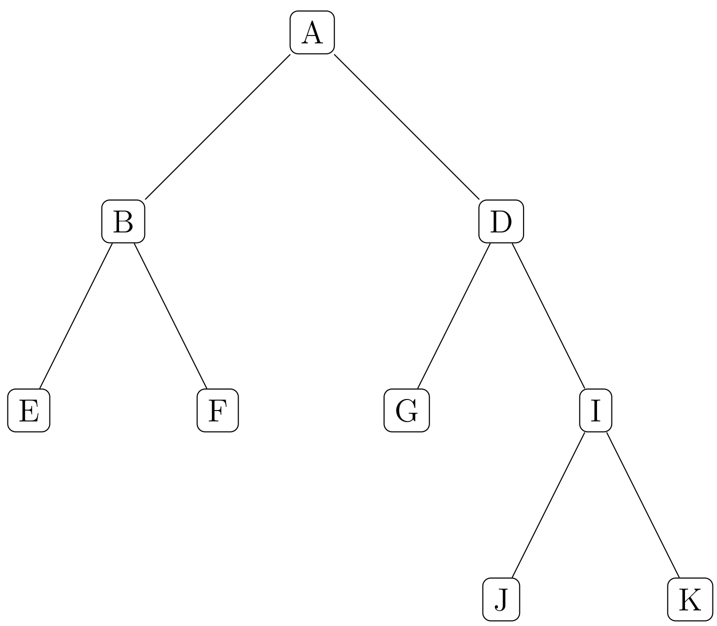
2.2.1 Arbre dégénéré ou filiforme

Définition 9.10
Un **arbre dégénéré** est un arbre dont les nœuds ne possèdent au plus un enfant.



2.2.2 Arbre localement complet

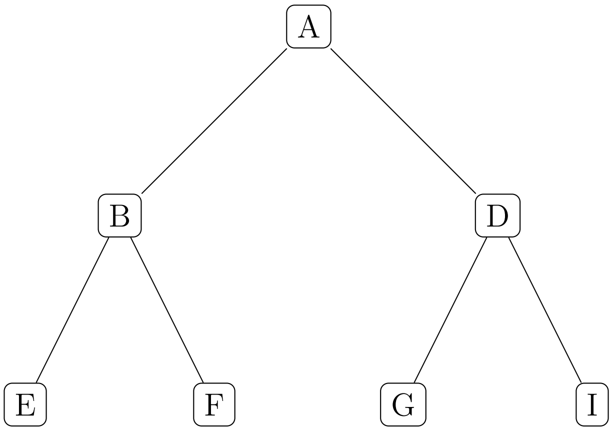
Définition 9.11
Un **arbre localement complet** est un arbre binaire dont chacun des nœuds possède soit deux enfants, soit aucun.



2.2.3 Arbre complet

Définition 9.12

C'est un arbre qui est localement complet et dont toutes les feuilles sont au niveau hiérarchique le plus bas.

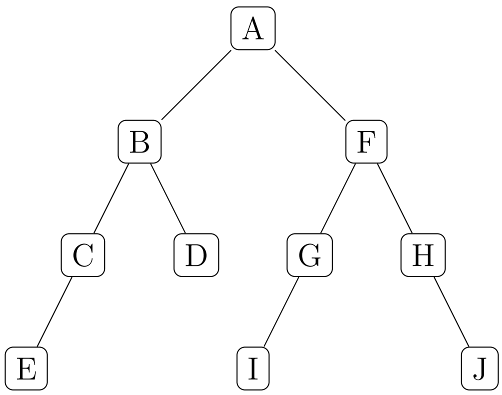


Exercice 9.9

- Combien de nœuds au maximum comporte un arbre localement complet de hauteur h ? Au minimum ?
- Combien de nœuds comporte un arbre complet de hauteur h ?

2.3 Notion de clé

À chaque nœud d'un arbre binaire, on associe une clé ("valeur" associée au nœud)



- Si on prend le nœud ayant pour clé A (le nœud racine de l'arbre) on a :
  - le sous-arbre gauche est composé du nœud ayant pour clé B, du nœud ayant pour clé C, du nœud ayant pour clé D et du nœud ayant pour clé E
  - le sous-arbre droit est composé du nœud ayant pour clé F, du nœud ayant pour clé G, du nœud ayant pour clé H, du nœud ayant pour clé I et du nœud ayant pour clé J
- si on prend le nœud ayant pour clé B on a :

  - le sous-arbre gauche est composé du nœud ayant pour clé C et du nœud ayant pour clé E
  - le sous-arbre droit est uniquement composé du nœud ayant pour clé D

- si on prend le noeud ayant pour clé G on a :
  - le sous-arbre gauche est uniquement composé du noeud ayant pour clé I
  - le sous-arbre droit est vide (NIL)

Remarque

Un arbre vide est noté NIL

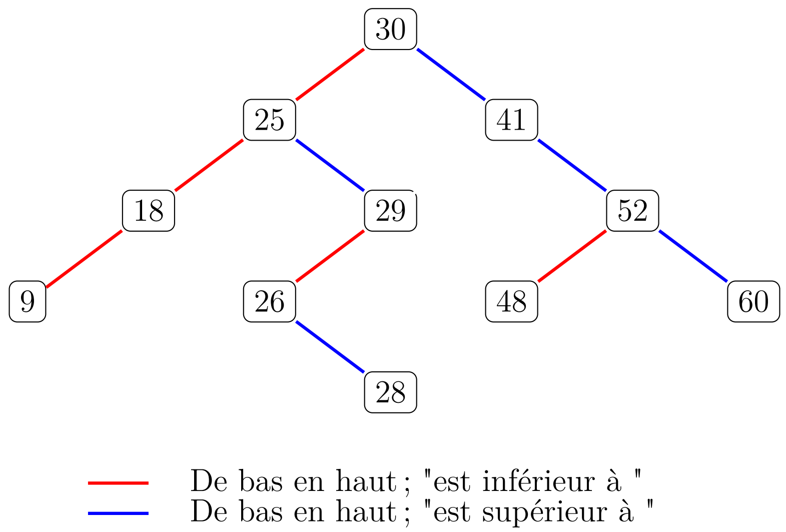
très important

Un sous-arbre (droite ou gauche) est un arbre (même s'il contient un seul nœud ou pas de nœud de tout (NIL)).

2.4 Les arbres binaires de recherche

Un arbre binaire de recherche est un cas particulier des arbres binaires qui doit satisfaire en plus deux conditions :

- Les clés de tous les nœuds du sous-arbre gauche d'un nœud X son inférieures ou égales à la clé de X
- Les clés de tous les nœuds du sous-arbre droit d'un nœud X sont strictement supérieures à la clé de X.



3 Algorithmes sur les arbres binaires

Notations : Soit T un arbre :

T.racine est le nœud racine de l'arbre T

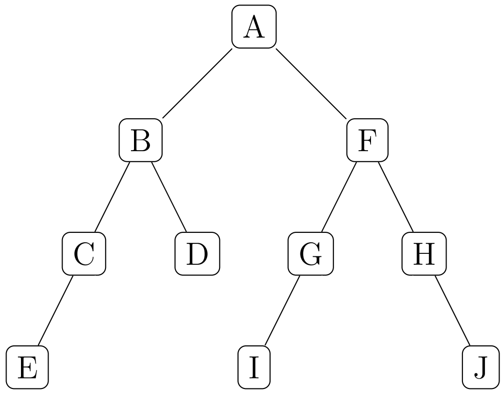
Soit un nœud x :

x.gauche correspond au sous-arbre gauche du nœud x

x.droit correspond au sous-arbre droit du nœud x

x.cle correspond à la clé du nœud x

3.1 Calcul de la hauteur d'un arbre

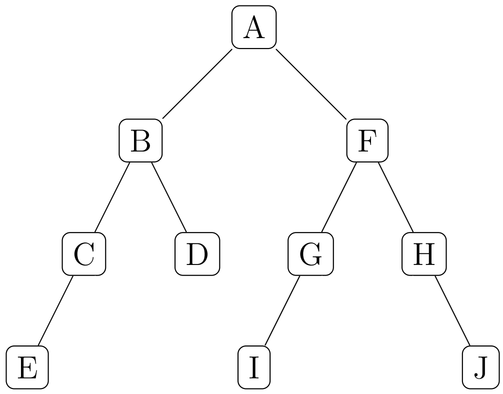


Appliquer cet algorithme à l'arbre ci-dessus.

```
VARIABLE
T : arbre
x : nœud
DEBUT
Function HAUTEUR(T)
    if T ≠ NIL then
        x ← T.racine
        renvoyer 1 + max(HAUTEUR(x.gauche), HAUTEUR(x.droit))
    else
        renvoyer 0
    end
end
FIN
```

3.2 Calcul de la taille d'un arbre

On considère de nouveau cet arbre :



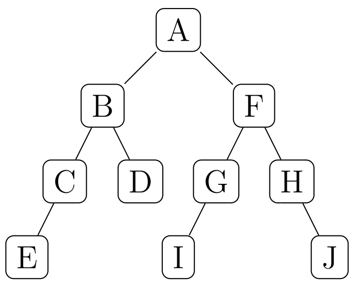
Appliquer cet algorithme à l'arbre ci-dessus.

```
VARIABLE
T : arbre
x : noeud
DEBUT ;
Function TAILLE(T)
    if  T ≠ NIL then
        x ← T.racine
        renvoyer 1 + TAILLE(x.gauche) + TAILLE(x.droit)
    else
        renvoyer 0
    end
end
FIN
```

3.3 Parcourir un arbre

Il existe plusieurs façons de parcourir un arbre !

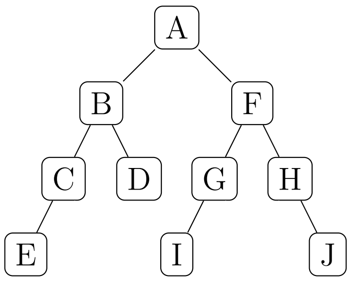
3.3.1 Parcours dans l'ordre infixe



```
VARIABLE
T : arbre
x : noeud
DEBUT
Function PARCOURS_ INF(T)
    if T ≠ NIL then
        x ← T.racine
        PARCOURS_ INF(x.gauche)
        affiche x.cle
        PARCOURS_ INF(x.droit)
    end
end
```

Dans quel ordre a été parcouru cet arbre ?

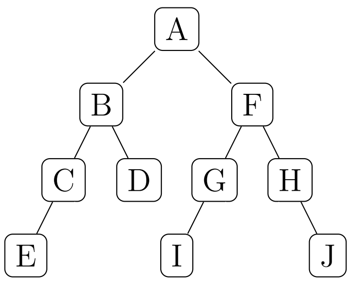
3.3.2 Parcours de l'arbre ordre préfixe




```
VARIABLE
T : arbre
x : nœud
DEBUT
Function PARCOURS_ PREFIXE(T)
    if T ≠ NIL then
        x ← T.racine
        affiche x.cle
        PARCOURS_ PREFIXE(x.gauche)
        PARCOURS_ PREFIXE(x.droit)
    end
end
```

Dans quel ordre a été parcouru cet arbre ?

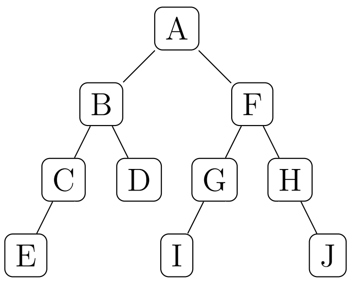
3.3.3 Parcours d'un arbre ordre suffixe



```
VARIABLE
T : arbre
x : nœud
DEBUT
Function PARCOURS_ SUFFIXE(T)
    if T ≠ NIL then
        x ← T.racine
        PARCOURS_ SUFFIXE(x.gauche)
        PARCOURS_ SUFFIXE(x.droit)
        afiche x.cle
    end
end
```

Dans quel ordre a été parcouru cet arbre ?

3.3.4 Parcourir un arbre en largeur d'abord



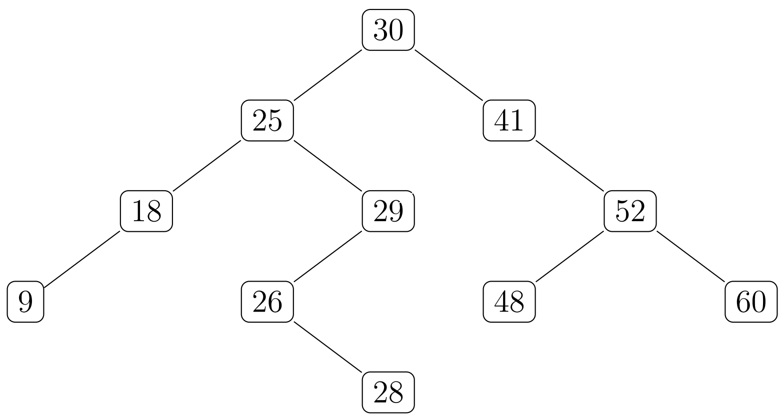

```
VARIABLE
T : arbre
Tg : arbre
Td : arbre x : nœud
f : file
DEBUT
Vider f # la file est vide
Function PARCOURS_ LARGEUR(T)
    enfiler(T.racine,f)
    f non vide  x ← défiler(f)
    afficher x.cle
    if x.gauche ≠ NIL then
        Tg ← x.gauche
        enfiler(Tg.racine,f)
    end
    if x.droit ≠ NIL then
        Td ← x.droite
        enfiler(Td.racine,f)
    end
end
```

Dans quel ordre a été parcouru cet arbre ?

Remarque

- Cet algorithme utilise une file FIFO
- Cet algorithme n'est pas récursif.

3.4 Recherche d'une clé dans un arbre binaire de recherche




```
VARIABLE
T : arbre
cle : clé
x : nœud
DEBUT
Vider f # la file est vide
Function RECHERCHER_ CLE(T,cle)
    if T = NIL then
        retouner Faux
    else
        x ← T.racine
        if x = cle then
            retourner Vrai
        end
        if x <cle then
            retourner RECHERCHER_ CLE (cle,x.droit)
        else
            retourner RECHERCHER_ CLE (cle,x.gauche)
        end
    end
end
```

Pour rechercher une clé dans un arbre binaire de recherche, on peut d'abord la comparer avec la racine. Si la clé est présente à la racine, on renvoie Vrai. Si la clé est inférieure à la racine, on cherche la clé dans le sous-arbre de gauche. Si la clé est supérieure à la racine, on cherche alors dans le sous-arbre de droite. Si la clé n'a pas été trouvée, on retourne Faux.

3.5 Insertion d'une clé dans un arbre binaire de recherche