Les arbres

Aziza EL OUAAZIZI
Cours SMIA S4
Faculté Polydisciplinaire de Taza
Université Sidi Mohammed Ben Abdellah

Introduction

Les arbres sont des structures hiérarchiques qui possèdent des liens ou des pointeurs vers d'autres arbres.

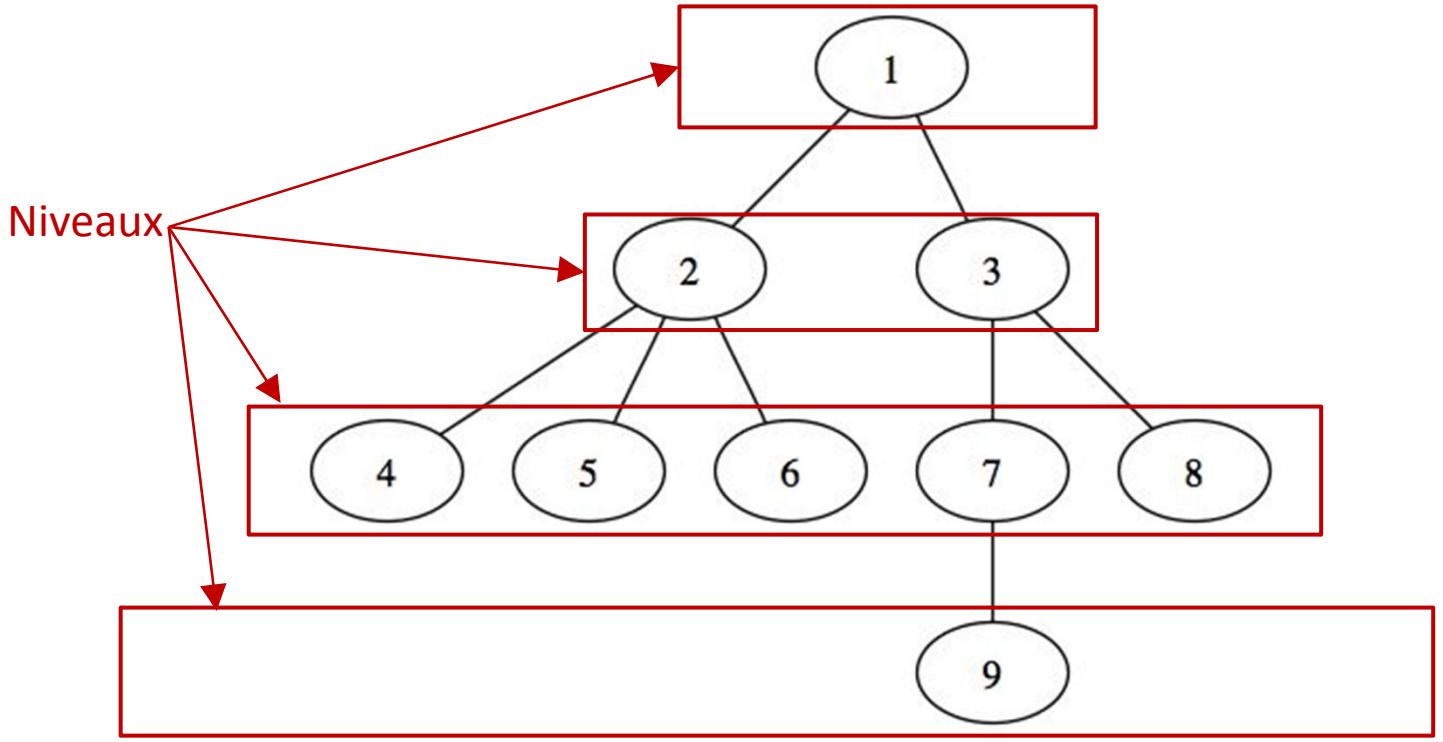
- Peuvent être définit de manière récursive.
- Utilisés dans l'implémentation des systèmes de bases de données, et dans l'organisation du système de fichiers d'un système d'exploitation ou dans n'importe quel système hiérarchique.
- Donnent de très bonnes performances s'ils sont utilisés correctement.

On distingue deux types d'arbres:

- les arbres enracinés
- les arbres non enracinés.

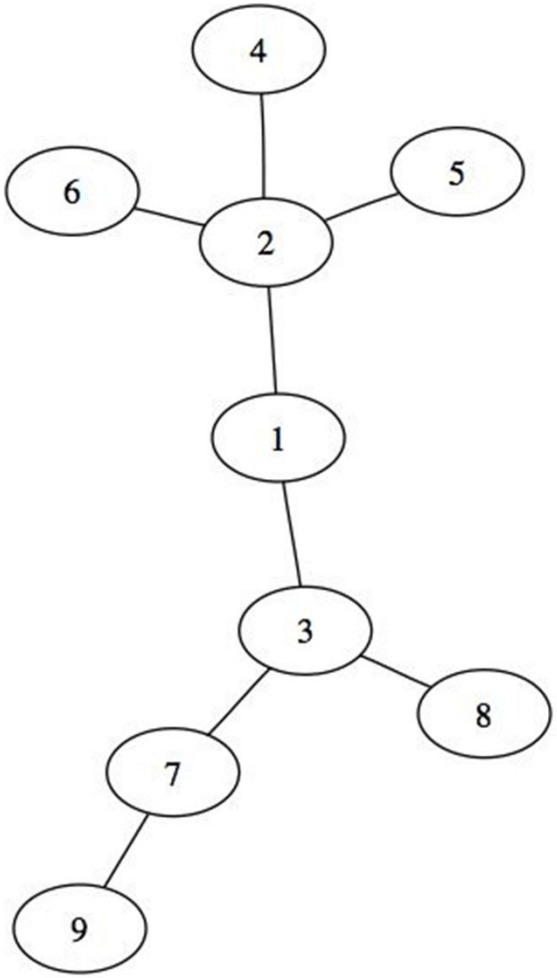
Arbre enraciné

Un arbre hiérarchique dans lequel on peut établir des niveaux. Il ressemble un arbre généalogique tel qu'on le conçoit couramment.



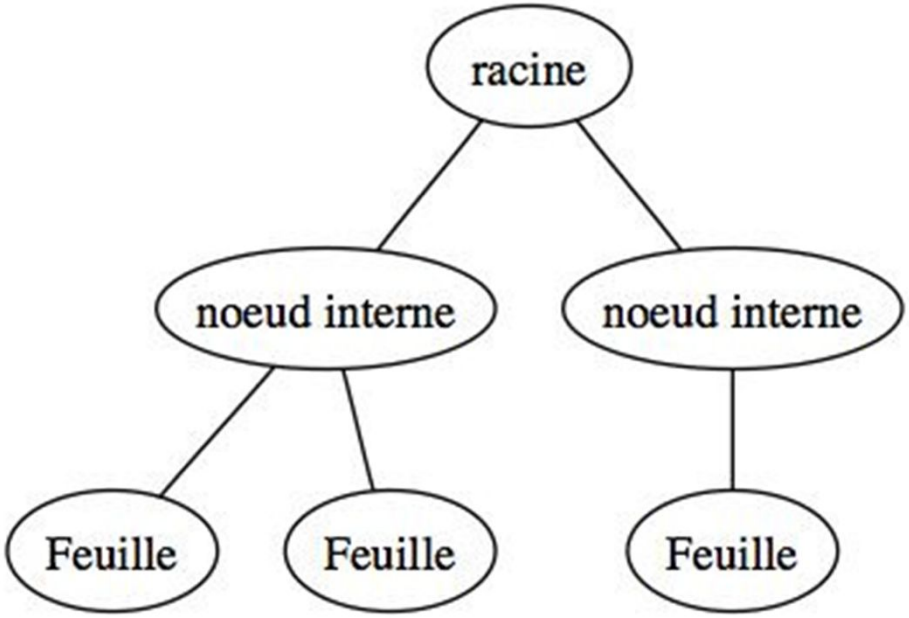
Arbre non enracinées

- un arbre qui n'a pas de relation d'ordre ou de hiérarchie entre les éléments qui le compose.
- On peut passer d'un arbre non enraciné à un arbre enraciné. Il suffit de choisir un élément comme sommet de l'arbre et de l'organiser de façon à obtenir un arbre enraciné.



Terminologie: Nœud, racine et feuille

- Chaque élément d'un arbre se nomme **un nœud**. Les nœuds sont reliés les uns aux autres par des relations d'ordre ou de hiérarchie.
- Il existe un nœud qui n'a pas de père, c'est **la racine** de l'arbre.
- Un nœud qui n'a pas de fils est appelé **une feuille**.
- Parfois on appelle une feuille un **nœud externe** tout autre nœud de l'arbre sera alors appelé un nœud interne.



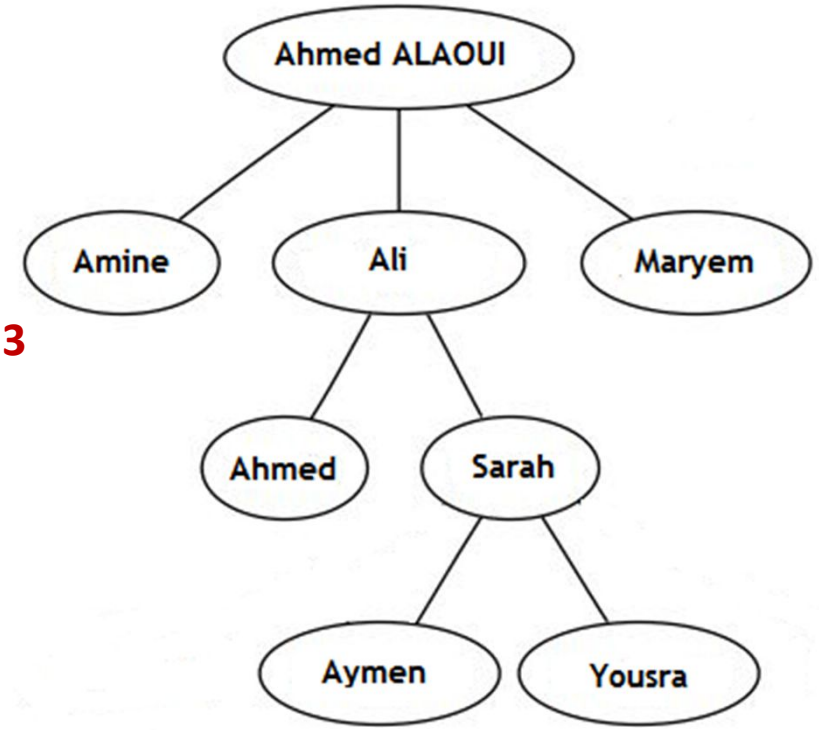
N.B: *On dit qu'un nœud admet un père s'il possède un nœud qui lui est supérieur dans cette hiérarchie.*
Un nœud peut avoir éventuellement un ou plusieurs fils

Terminologie: Arité d'un arbre

- Un arbre dont les nœuds comportent au maximum **n** fils est appelé un arbre **n-aire** ou arbre d'arité **n**
- On appelle **degré d'un nœud**, le nombre de fils que possède ce nœud.

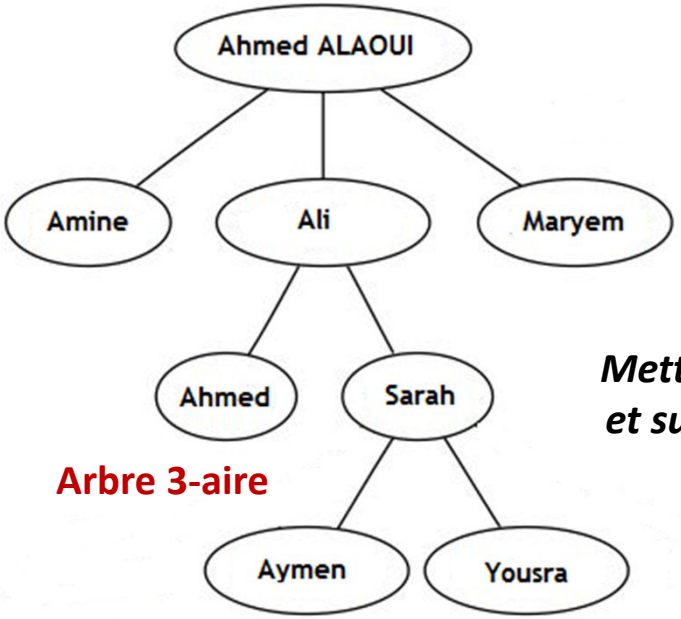
Exemple d'arbre d'arité 3

ou arbre 3-aire



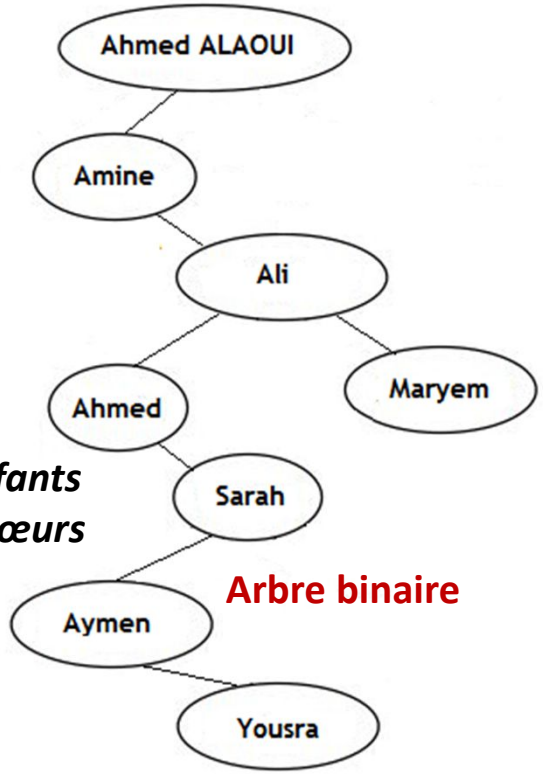
Arbre binaire

- Un cas particulier des arbres enracinés est l'arbre dit **binaire** où chaque nœud possède au maximum 2 fils: un fils gauche et un fils droit.
- Un arbre binaire est un cas basique: les algorithmes de manipulation d'un tel arbre peuvent s'appliquer sur n'importe quel arbre par simple généralisation .
- Il est possible de passer d'un arbre n-aire à un arbre binaire



Arbre 3-aire

Mettre sur les fils gauches les enfants et sur les fils droits les frères et sœurs



Arbre binaire

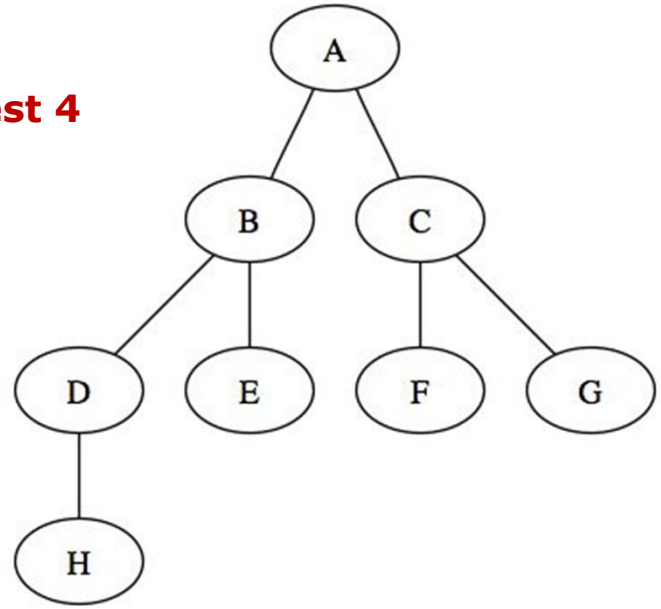
Terminologie: Taille et hauteur d'un arbre

- **Taille d'un arbre;** nombre de nœud interne qui le compose. C'est à dire le nombre nœud total moins le nombre de feuille de l'arbre.
- **Profondeur d'un nœud;** sa distance en termes de nœuds par rapport à la racine. Par convention, la racine est de profondeur 0.
- **Hauteur d'un l'arbre:** profondeur maximale de ses nœuds: la profondeur à laquelle il faut descendre dans l'arbre pour trouver le nœud le plus loin de la racine.

- ✓ **La taille de l'arbre de l'exemple est 4**

- ✓ **Le nœud A est de profondeur 0**
- ✓ **Le nœud B est de profondeur 1**
- ✓ **Le nœud F est de profondeur 2**
- ✓ **Le nœud H est de profondeur 3**

- ✓ **La hauteur de l'arbre est 3**

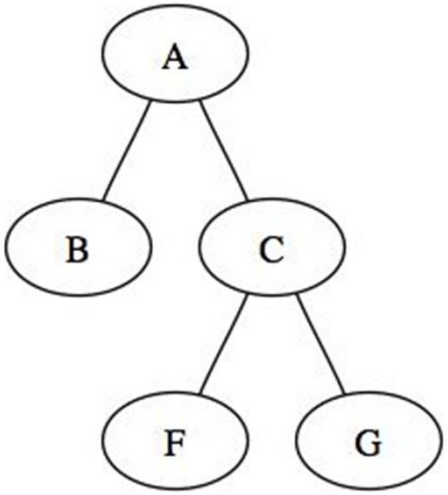


Arbre binaire localement complet

- Un arbre binaire dont chacun des nœuds possèdent soit 0 soit 2 fils. Dans ce cas tout les nœuds internes auront tous deux fils.
- Dans ce type d'arbre, on peut établir une relation entre la taille de l'arbre et le nombre de feuille:
  - un arbre binaire localement complet de taille **n** aura **n+1** feuilles.

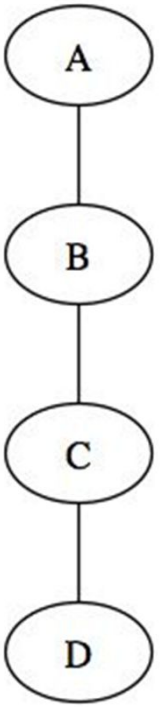
✓ **La taille de l'arbre de l'exemple est 2**

✓ **Le nombre de feuilles est 3=2+1**



Arbre dégénéré

- **Un arbre dégénéré** (appelé aussi filiforme) est un arbre dont les nœuds ne possèdent qu'un et un seul fils. Cet arbre est donc tout simplement une liste chaînée.
- Ce type d'arbre est à éviter, puisqu'il n'apportera aucun avantage par rapport à une liste simplement chaînée.
- Un arbre dégénéré de **taille n** a une **hauteur** égale à **n**.
- L'arbre de l'exemple est un arbre dégénéré de taille 3 et de hauteur 3.



Arbre binaire complet

- un arbre qui est localement complet et dont toutes les feuilles ont la même profondeur.
- le nombre de nœuds **n** d'un arbre binaire complet peut se calculer en fonction de sa hauteur **h**:

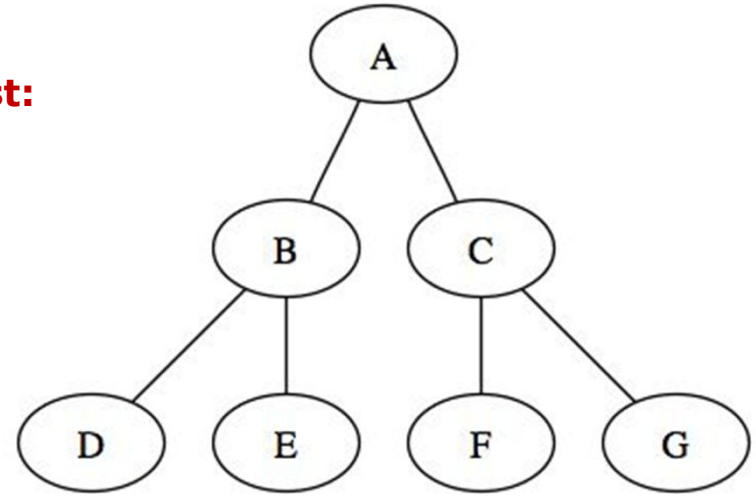
$$n = 2^{(h+1)} - 1$$

✓ **La hauteur de l'arbre de l'exemple est:**

h=2

✓ **Le nombre de nœud est:**

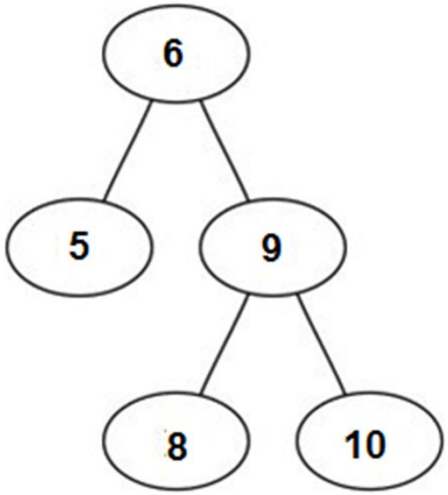
n=7=2^3-1



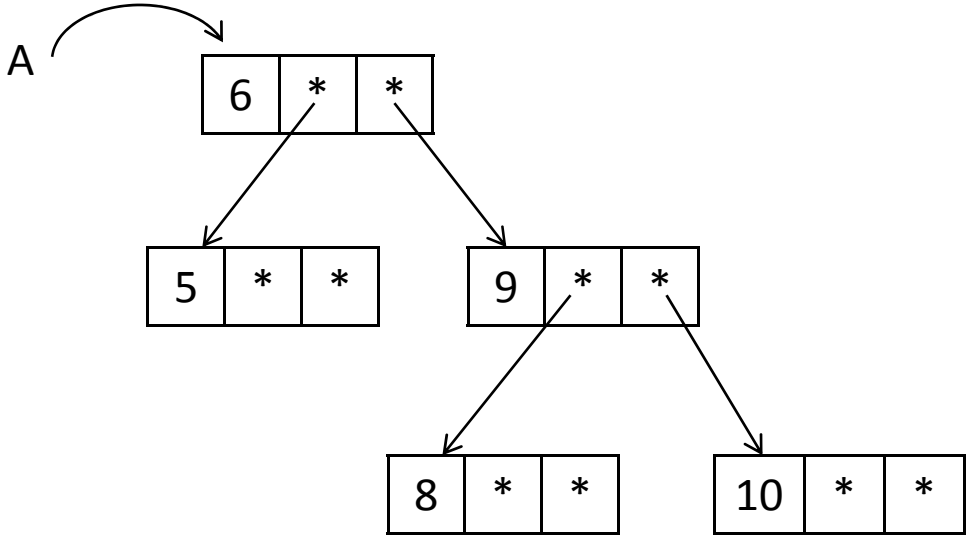
Représentation d'un arbre binaire en mémoire

Un nœud est une structure (ou un enregistrement) qui contient au minimum trois champs:

- un champ contenant valeur du nœud qui peut être ne n'importe quel type;
- deux autres champs sont le fils gauche et le fils droit du nœud: Ces deux fils on les appelle généralement les sous arbres gauches et les sous arbres droit du nœud.



Représentation symbolique d'un arbre A



Représentation en mémoire d'un arbre A

Implémentation d'un arbre binaire

```
/*définition du type nœud*/
typedef struct nœud
{
    int valeur;      //si valeurs stockées sont des entiers
    struct nœud* fdroit
    struct nœud* fgauche;
}noeud;

/*un arbre est un pointeur qui pointe vers le nœud racine*/
typedef noeud* arbre;

/*déclaration d'un arbre vide*/
arbre a=NULL;
```

Implémentation d'un arbre d'arité n>2 connue

- Utiliser un tableau pour stocker les fils.
- Pour un arbre d'arité n=5 nous aurons l'implémentation suivante :

```
typedef struct nœud
{
    int valeur;
    struct nœud* fils[5];
}noeud;

typedef nœud* arbre;

arbre a=NULL;
```

N.B: *Dans ce cas, les fils ne seront pas identifiés par leur position (droite ou gauche) mais par un numéro.*

Implémentation d'un arbre d'arité n>2 inconnue

- Utiliser une liste chainée pour stocker les fils.

```
struct nœud
{
      int valeur;
      liste fils;
};

typedef struct nœud* arbre;

struct element
{
      noeud* ar;
      element* suivant;
};
typedef struct element*  liste;

arbre a=NULL;
```

Parcours d'un arbre binaire

- Le parcours d'un arbre permet de visiter tous les nœuds d'un arbre et éventuellement appliquer des traitements sur ces nœuds.
- Nous distinguerons deux types de parcours :
  - Le parcours en profondeur permet d'explorer l'arbre en explorant jusqu'au bout une branche pour passer à la suivante.
  - Le parcours en largeur permet d'explorer l'arbre niveau par niveau: parcourir tous les nœuds du niveau 1 puis ceux du niveau deux et ainsi de suite jusqu'à l'exploration de tous les nœuds.

Parcours en profondeur

- On peut faire un parcours en profondeur à gauche ou à droite.
- Un parcours en profondeur à gauche signifie que l'on parcourt les branches de gauche avant les branches de droite.
- Le parcours en profondeur à gauche peut se réaliser de trois manières différentes:
  - Parcours infixe ou GRD
  - Parcours préfixe ou RGD
  - Parcours post fixe ou GDR

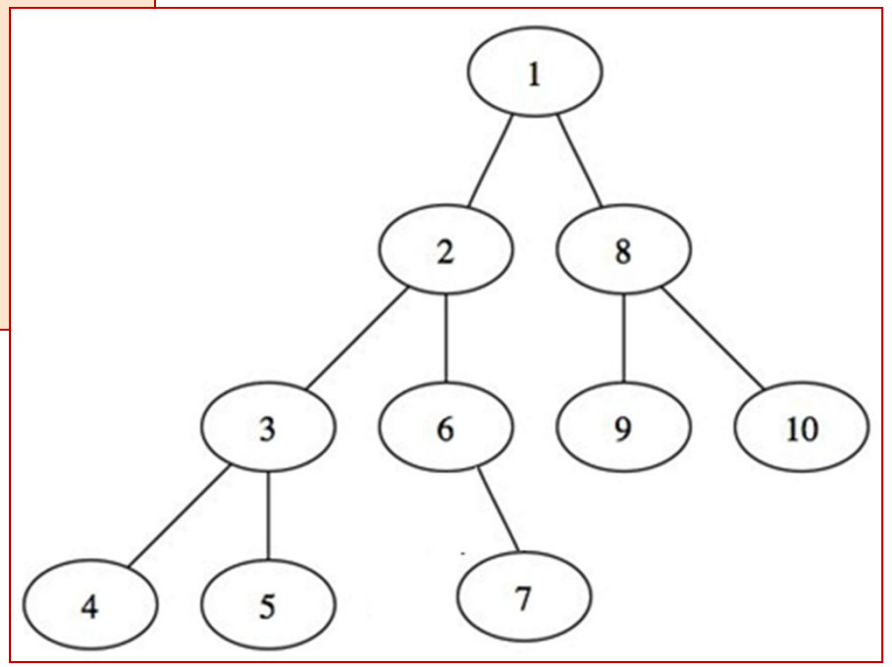
Parcours préfixe

- On traite la racine de l'arbre, on parcourt tout le sous arbre de gauche, une fois qu'il n'y a plus de sous arbre gauche on parcourt les éléments du sous arbre droit.
- Ce type de parcours peut être résumé en trois lettres : R G D (Racine Gauche Droit).

```
Void parcours_prefixe(arbre a)
{
      if (a!=NULL)
      {
           traiter(a);
           parcours_prefixe(a->fgauche);
           parcours_prefixe(a->fdroit);
      }
}
```

Exemple

```
Void parcours_prefixe(arbre a)
{
    if (a!=NULL)
    {
      printf("%d\t",a->valeur);
      parcours_prefixe(a->fgauche);
      parcours_prefixe(a->fdroit);
    }
}
```



L'exécution de cette fonction sur l'arbre ci-contre affichera:

1 2 3 4 5 6 7 8 9 10

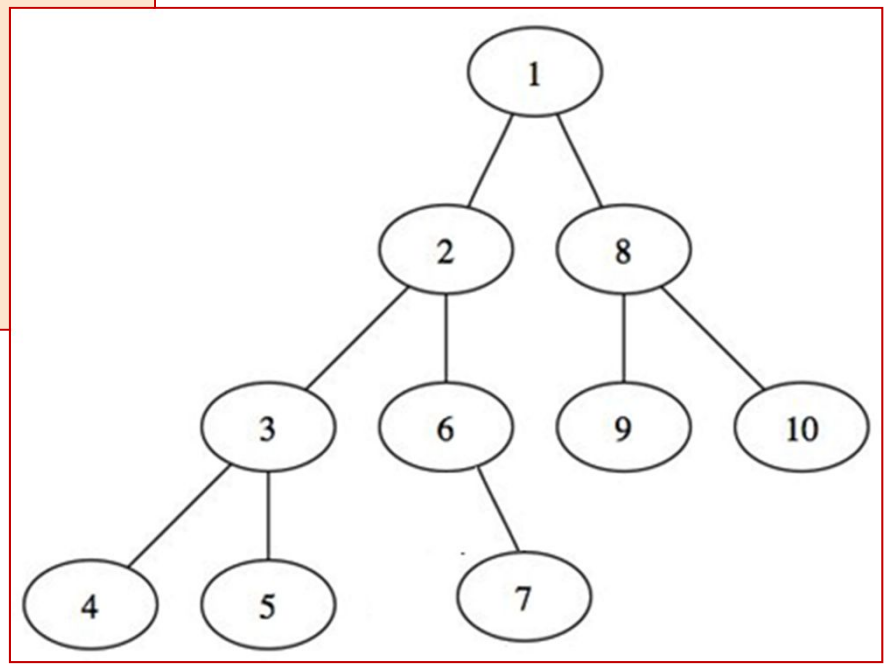
Parcours infixe

- On parcourt tout le sous arbre de gauche, une fois qu'il n'y a plus de sous arbre gauche, on traite la racine de l'arbre puis on parcourt les éléments du sous arbre droit.
- Ce type de parcours peut être résumé en trois lettres : G R D (Gauche Racine Droit).

```
Void parcours_infixe(arbre a)
{
      if (a!=NULL)
      {
          parcours_infixe(a->fgauche);
          traiter(a);
          parcours_infixe(a->fdroit);
      }
}
```

Exemple

```
Void parcours_infixe(arbre a)
{
    if (a!=NULL)
    {
      parcours_infixe(a->fgauche);
      printf("%d\t",a->valeur);
      parcours_infixe(a->fdroit);
    }
}
```



L'exécution de cette fonction sur l'arbre ci-contre affichera:

4 3 5 2 6 7 1 9 8 10

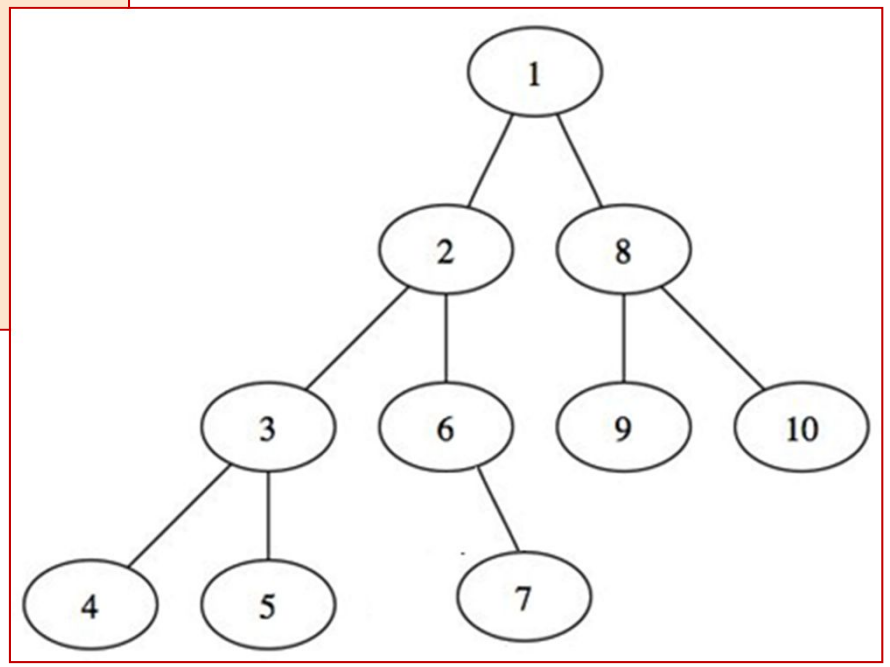
Parcours postfixe

- On parcourt tout le sous arbre de gauche, une fois qu'il n'y a plus de sous arbre gauche, on parcourt les éléments du sous arbre droit puis on traite la racine de l'arbre.
- Ce type de parcours peut être résumé en trois lettres : G D R (Gauche Droit Racine).

```
Void parcours_postfixe(arbre a)
{
     if (a!=NULL)
     {
          parcours_postfixe(a->fgauche);
          parcours_postfixe(a->fdroit);
          traiter(a);
     }
}
```

Exemple

```
Void parcours_postfixe(arbre a)
{
   if (a!=NULL)
   {
     parcours_postfixe(a->fgauche);
     parcours_postfixe(a->fdroit);
     printf("%d\t",a->valeur);
   }
}
```

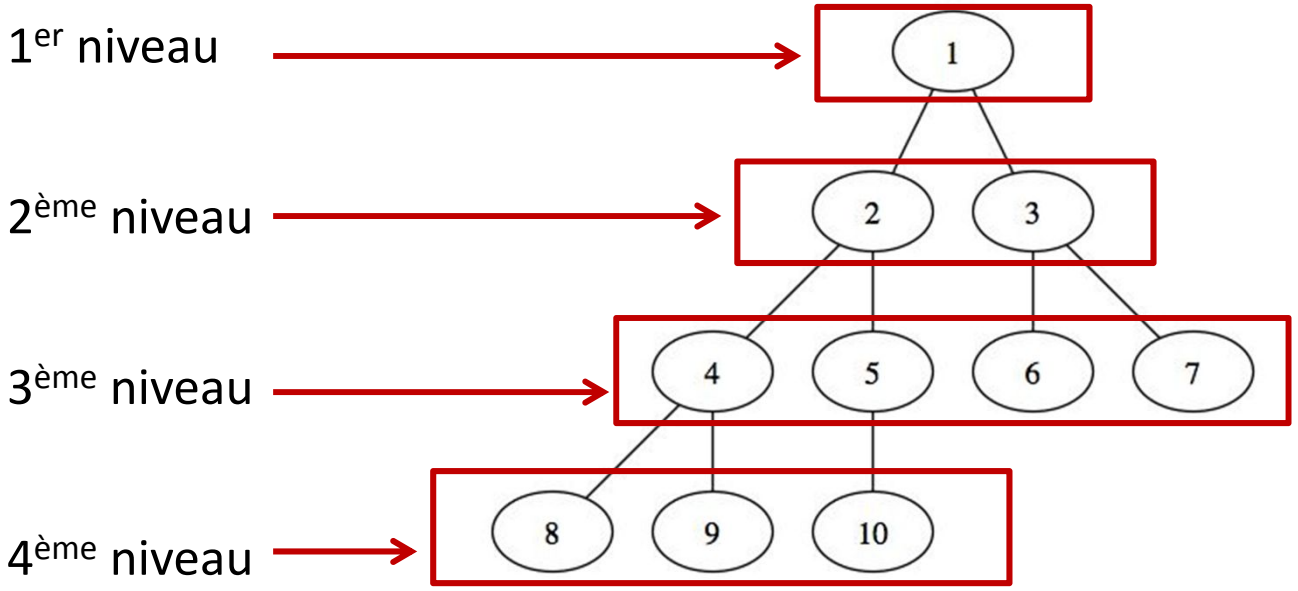


L'exécution de cette fonction sur l'arbre ci-contre affichera:

4 5 3 7 6 2 9 10 8 1

Parcours en largeur ou par niveau

- Il s'agit d'un parcours dans lequel, on traite les nœuds un par un sur un même niveau. On passe ensuite sur le niveau suivant, et ainsi de suite.
- L'affichage des données de l'arbre ci-dessous par utilisation du parcours en largeur donnera : 1 2 3 4 5 6 7 8 9 10.



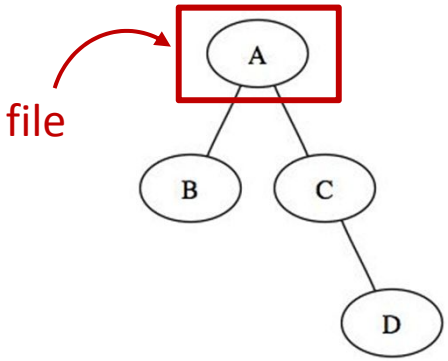
Réalisation du parcours en largeur

Utiliser une structure de données de type file d'attente pour de stocker les nœuds des niveaux successifs en vu de les traiter. le principe est le suivant:

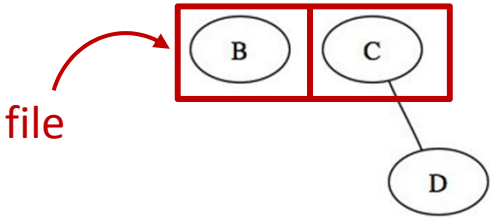
- Si l'arbre est non vide, on enfile le nœud racine dans la file d'attente
- Tant que la file n'est pas vide:
  - on défile un nœud, on le traite (par exemple on affiche sa valeur);
  - on enfile les fils gauche et droit non vides de ce nœud dans la file d'attente;

Réalisation sur un exemple

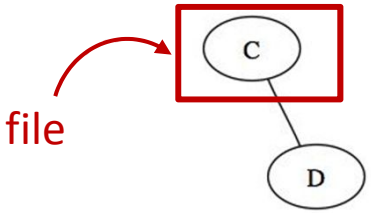
1- Au tout début de l'algorithme, la file ne contient rien, on y enfile le nœud racine de l'arbre, la file d'attente devient donc :



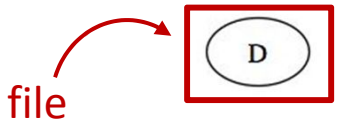
2- On défile la racine, on la traite puis on enfile le fils gauche B et droit C, la file devient donc :



3- On défile et on traite ensuite le prochain nœud B de la file d'attente. Ce nœud n'a pas de sous arbre, on n'ajoute donc rien à la file. Celle ci devient donc:



4- On défile et on traite ensuite le prochain élément C de la file d'attente. Celui ci a un fils droit, on l'enfile. La file d'attente devient:



5- On défile et on traite le nœud D. Celui ci n'ayant pas de fils, nous n'ajoutons donc rien à la file. La file devient vide et à ce stade l'algorithme se termine.

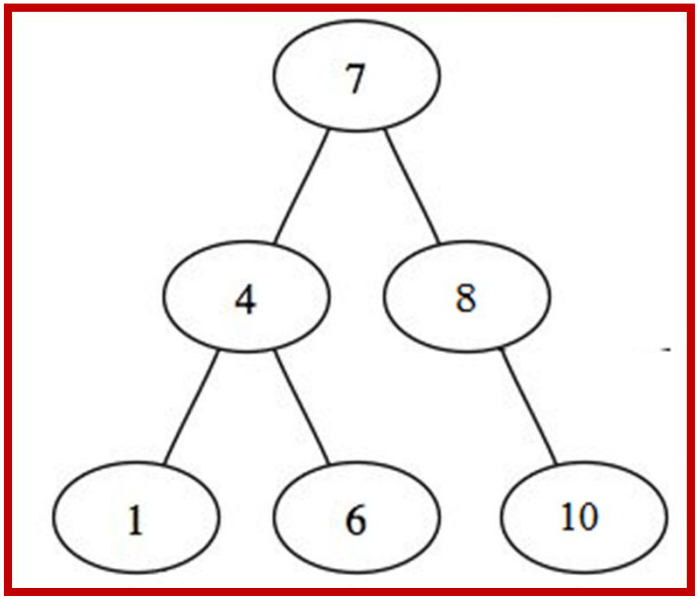
Algorithme de parcours en largeur

```
void parcours_largeur(arbre a)
{
   File f=NULL;   /* pour mémoriser les noeud d'un même niveau */
   noeud* x=NULL; /* pour pointeur le noeud à traiter */

   if(a!=NULL)
   {
     enfiler(&f,a); /* si l'arbre n'est pas vide, on enfile sa racine */

     while(f!=NULL)
     {
        /* on défile le nœud courant puis on le traite */
        x=defiler(&f);
        traiter(x);
        /* si le nœud défilé contient des fils, on les enfile à leur
        tour dans la file*/
        if(x->fgauche!=NULL)
           enfiler(&f,x->fgauche);
        if(x->fdroit!=NULL)
           enfiler(&f,x->fdroit);
     }
   }
}
```

Arbre binaire de recherche

C'est un arbre permettant un stockage ordonné et logique des données. Il a les propriétés suivantes:

- Tout nœud à gauche de la racine est inférieur à la valeur de la racine,
- Tout nœud à droite de la racine est supérieur à la valeur de la racine.



N.B: *Ces propriétés doivent être vérifiées récursivement à tous les niveaux pour que l'arbre binaire soit dit ordonné ou de recherche. C'est ce modèle d'arbre qu'on utilise le plus souvent car il permet de faire une recherche rapide et il est assez facile à actualiser (suppression, insertion).*

Conclusion

- Les arbres sont des structures de données appropriées à la manipulation d'objets entre lesquels il existe une relation d'hiérarchie:
  - Arbre généalogique
  - Structure d'un livre
  - Structure d'un système de fichiers ou d'une base de données
- Pour accéder au données stocké dans les nœuds d'un arbre et éventuellement appliquer des traitements sur ces nœuds, on peut utiliser soit:
  - un parcours en profondeur
  - Un parcours par niveau
- Pour manipuler les données au sein d'un arbre on utilise plutôt une structure ordonnée de celle-ci.
- Les autres algorithmes de manipulation des arbres ordonnées seront présentés en TD.