Filière: MPI

Module: Programmation II

Plan du cours

Chapitre 2. Les structures arborescentes

1-Définition

Un arbre est un ensemble organisé de nœuds dans lequel chaque nœud a un père et un seul, sauf un nœud que l'on appelle la racine. On le représente généralement en mettant la racine en haut et les feuilles en bas (contrairement a un arbre réel).

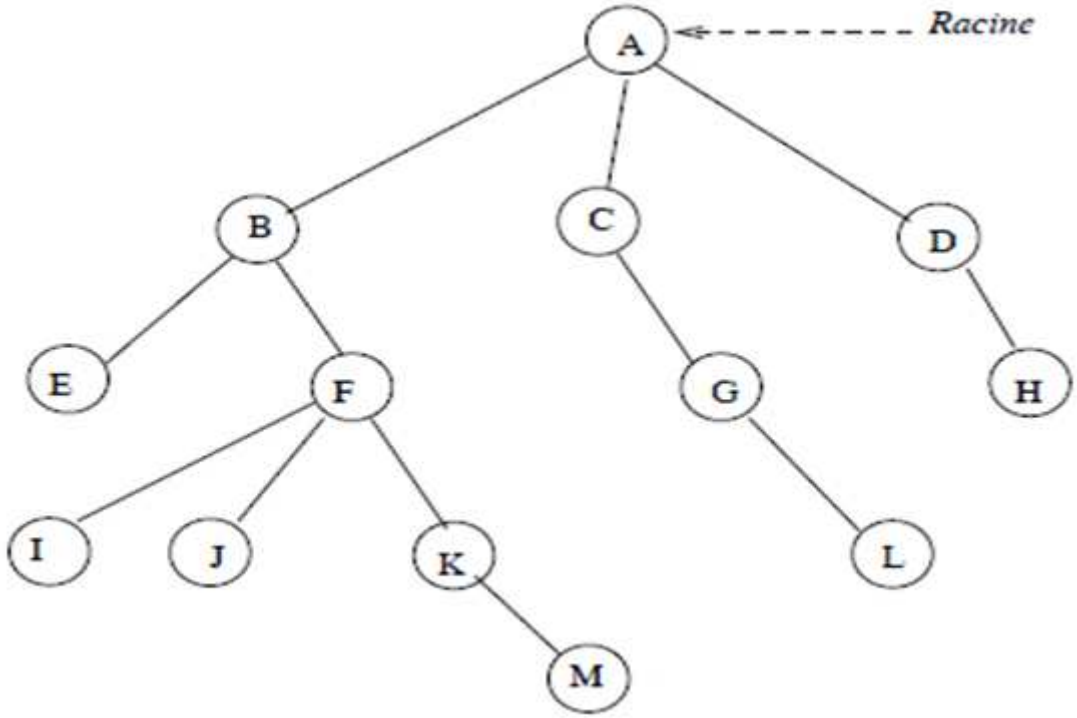
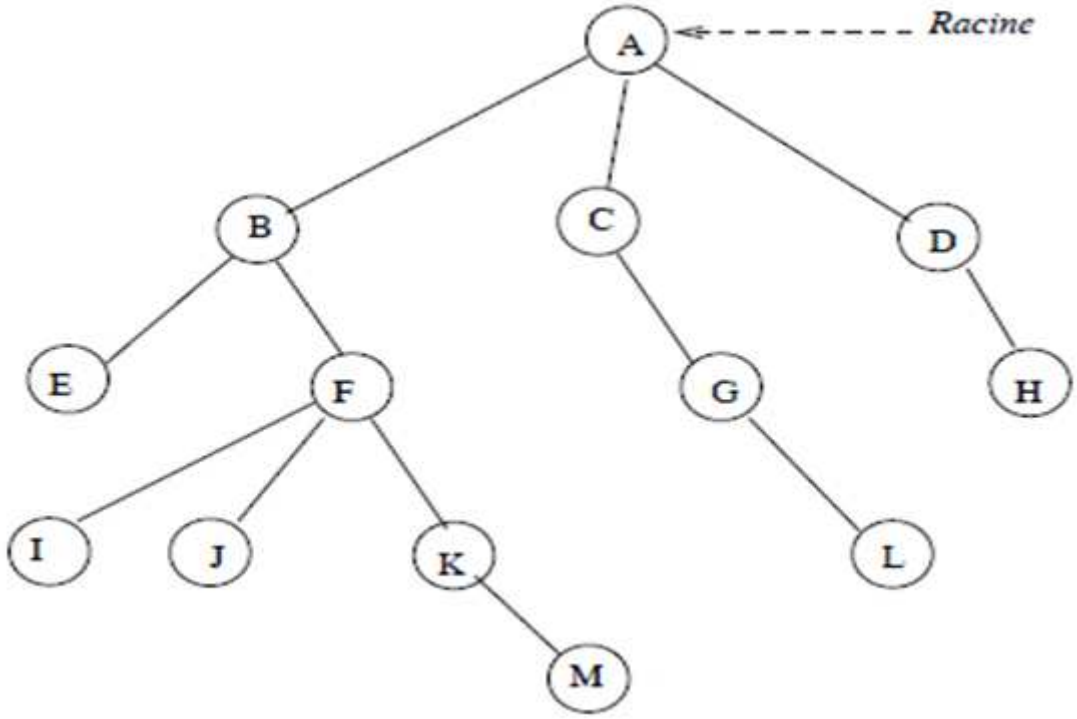


Figure 1: Exemple d'un arbre

Si le nœud B est le père du nœud F, nous dirons que F est un fils de B, et si un nœud n'a pas de fils nous dirons que c'est une feuille.

Le nœud A est la racine de l'arbre.
Les nœuds E, I, J, M, L et H sont des feuilles.
Les nœuds B, C, D, F, G et K sont des noeuds intermédiaires.



2- arbre binaire

• Un arbre binaire est un arbre tel que les nœuds ont au plus deux fils (gauche et droit).

• La hauteur d'un arbre est le nombre de niveaux de ses nœuds. C'est donc aussi le nombre de nœuds que contient la branche la plus longue.

• Un arbre binaire est complet si toutes ses branches ont la même longueur et tous ses nœuds qui ne sont pas des feuilles ont deux fils

Hauteur =5

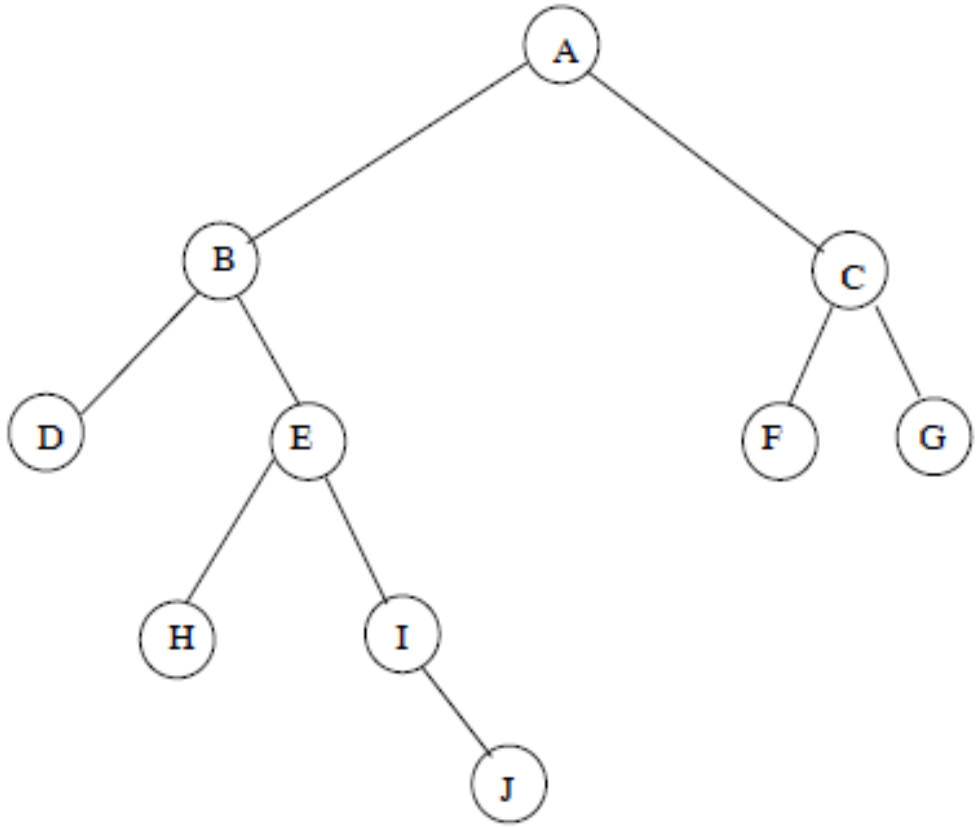


Figure 2: Exemple d'un arbre binaire

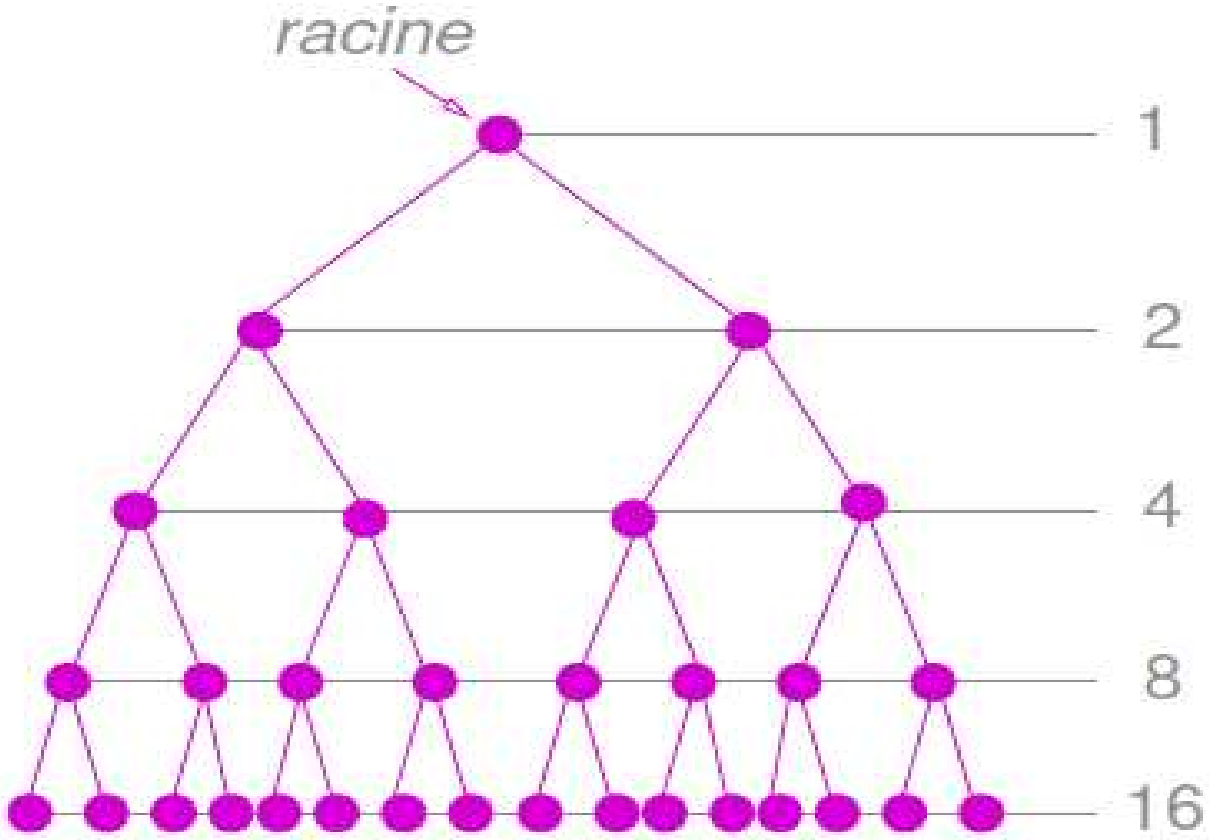


Figure 3: Exemple d'un arbre binaire complet

2.1- arbre binaire de recherche

Un arbre binaire de recherche est un arbre binaire qui possède la propriété fondamentale suivante:

• tous les nœuds du sous-arbre de gauche d'un nœud de l'arbre ont une valeur inferieure ou égale a la sienne.

• tous les nœuds du sous-arbre de droite d'un nœud de l'arbre ont une valeur supérieure ou égale a la sienne.

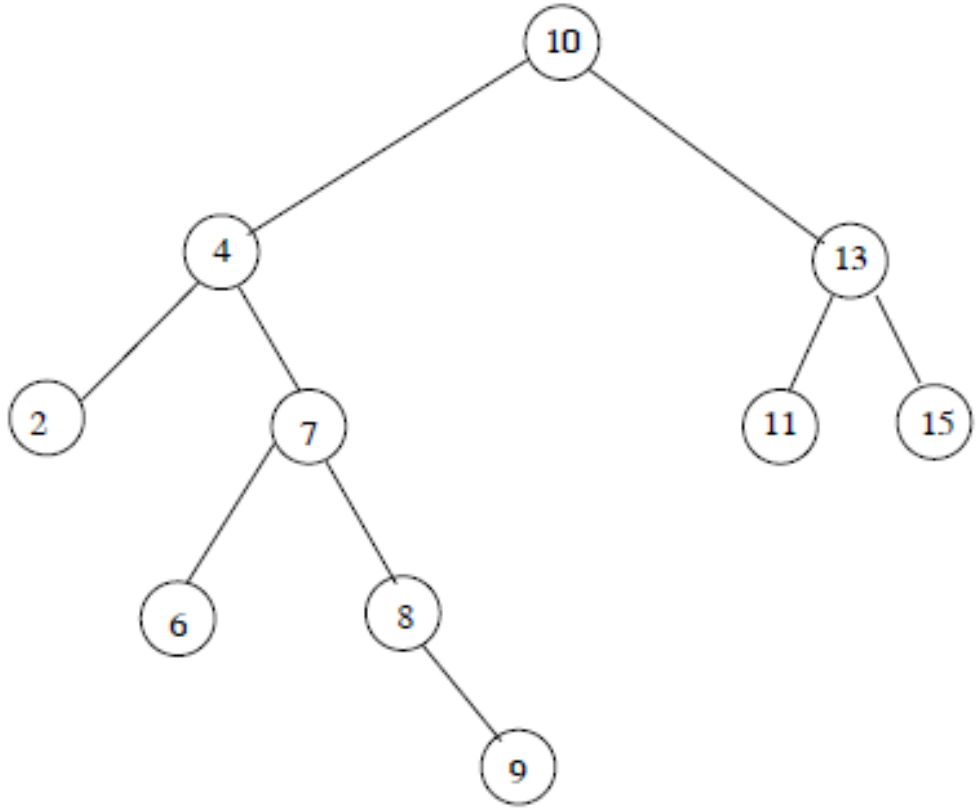


Figure 4: Exemple d'un arbre binaire de recherche

2.2-Définition en C

Définition en C de la structure Nœud d'un arbre binaire.

```
typedef struct arb
    { int val;
      struct arb *fg;
      struct arb *fd;
     } arb;
```

Définition en C d'un arbre binaire.

```
typedef arb* arbre;
```

Les opérations sur un arbre binaire

créer un arbre avec un seul noeud : *arbre creer_noeud (int el);*
Fusionner deux arbres un à gauche et un à droire:
arbre fusion (int el, arbre ag, arbre ad);
Taille d'un arbre*: int taille_abr (arbre ar)*
Profondeur d'un arbre: int profondeur_abr (arbre ar);
Rechercher un élément dans un arbre binaire de recherche :
int recherche_abr (int el, arbre ar);
Parcourir un arbre : *trois types de parcours*
Ajouter un élément à un arbre binaire de recherche:
arbre ajouter_abr(arbre ar, int el)
Supprimer un élément d'un arbre binaire de recherche:
arbre supprimer_abr(arbre ar, int el)

- programmes en C des opérations d'un arbre binaire

```
 arbre creer_noeud (int el)
{
      arbre nod;
 if (((nod)= (arbre ) malloc (sizeof(arb)))== NULL)

{ printf ("erreur allocation");
Exit(1);
}
 else
   {          nod→val =el;
              nod→ fg= NULL;
              nod→ fd=NULL;
              return(nod);
    }
}
```

```
arbre fusion (int el, arbre ag, arbre ad)
 {

      arbre nod;
if (((nod)=(arbre )malloc (sizeof(arb)))==NULL)
  { printf ("erreur allocation");
Exit(1);
}
else
   {        nod → val=el;
            nod → fg=ag;
            nod → fd=ad;
              return(nod);
    }
}
```

```
int taille_abr (arbre ar)
{ int p1, p2;
  if (ar==NULL)
            return 0;
  p1=taille_abr (ar→ fg);
  p2=taille_abr (ar→ fd);
 return  1+p1+p2;
}
```

```
int profondeur_abr (arbre ar)
 {
   int p1, p2;
  if (ar==NULL)
     return 0;
   p1=profondeur_abr (ar→ fg);
   p2= profondeur_abr (ar→ fd);
   if (p1 > p2)
      return 1+p1;
   else return 1+p2;
  }
```

```
int recherche_abr (int el, arbre ar)
{   if (ar==NULL)
                    return 0;
          if (el== ar → val)
                    return  1;
         else if (el <ar → val)
                        return recherche_abr(el, ar → fg);
                      else
                         return recherche_abr (el, ar → fd);
}
```

Parcours d'arbre

Trois types de parcours:

1- Parcours préfixé
2- Parcours infixé
3- Parcours postfixé

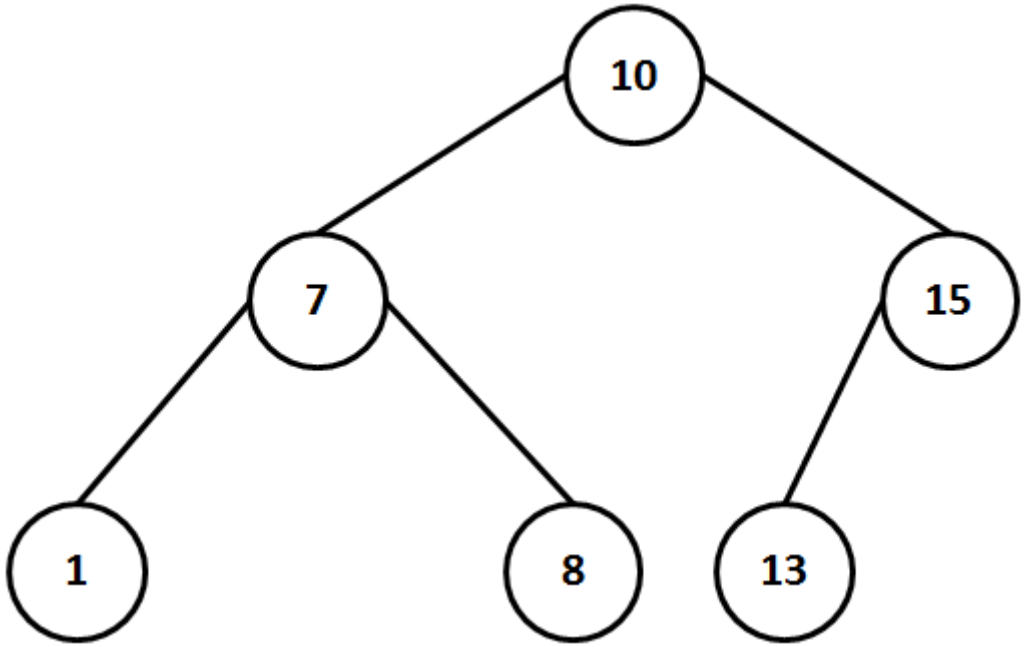
Parcours d'arbre

1- Parcours préfixé

Les nœuds sont ordonnés selon le première passage par un nœud.

```
void parcours_prefixe (arbre ar)
{
if (ar!= NULL)
        {
           printf("%d-",ar→val);
           parcours_prefixe (ar→fg);
           parcours_prefixe (ar→fd);

            }
 }
```



Parcours préfixé 10 - 7 - 1 - 8 - 15 - 13

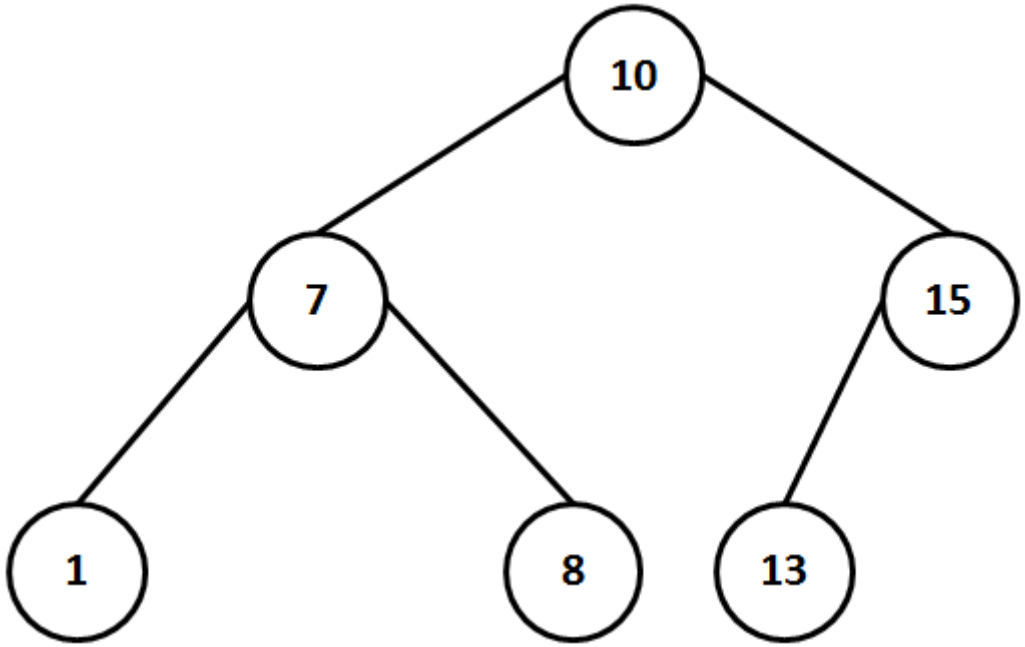
Parcours d'arbre

2- Parcours infixé

Les nœuds sont ordonnés selon le second passage par un nœud.

```
void parcours_infixe (arbre ar)
{
if (ar!= NULL)
        {
           parcours_infixe (ar→fg);
           printf("%d-",ar→val);
           parcours_infixe (ar→fd);

            }
 }
```



Parcours infixé **1 - 7 - 8 - 10 - 13 - 15**

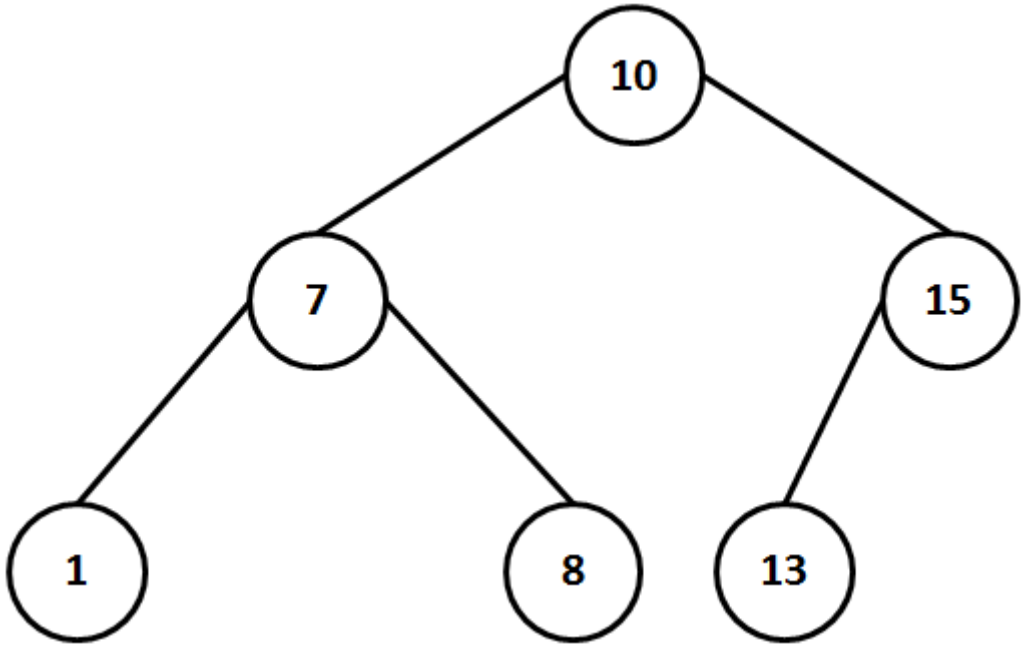
Parcours d'arbre

3- Parcours postfixé

Les nœuds sont ordonnés selon le troisième passage par un nœud.

```
void parcours_postfixe (arbre ar)
{
if (ar!= NULL)
        {
           parcours_postfixe (ar→fg);
           parcours_postfixe (ar→fd);
           printf("%d-",ar→val);

            }
 }
```



Parcours postfixé 1 - 8 - 7 - 13 - 15 - 10

<u>Ajout d'un élément à un arbre binaire de recherche</u> (dans le cas ou la valeur à insérer n'existe pas dans l'arbre)

```
arbre  ajouter_abr ( arbre ar , int valeur )
 {
if ( ar == NULL)
return creer_noeud ( valeur ) ;
else {
if ( valeur < ar→val )

ar→fg = ajouter_abr ( ar→fg , valeur ) ;
Else if(valeur >ar→val )
{
ar→fd = ajouter_abr ( ar→fd , valeur ) ;
}
return ar ;

}
}
```

Suppression d'un nœud dans un arbre binaire de recherche

La suppression d'un élément d'un arbre binaire de recherche consiste à supprimer une valeur de l'arbre tout en conservant les propriétés de l'arbre binaire de recherche. L'algorithme est le suivant (une fois trouvé le nœud contenant la valeur en question) :

- si le nœud à enlever ne possède aucun fils, on l'enlève,
- si le nœud à enlever n'a qu'un fils, on le remplace par ce fils,
- si le nœud à enlever a deux fils, on le remplace par le sommet de plus petite valeur dans le sous-arbre droit, ou bien par le sommet de plus grande valeur dans le sous arbre gauche puis on supprime ce sommet.