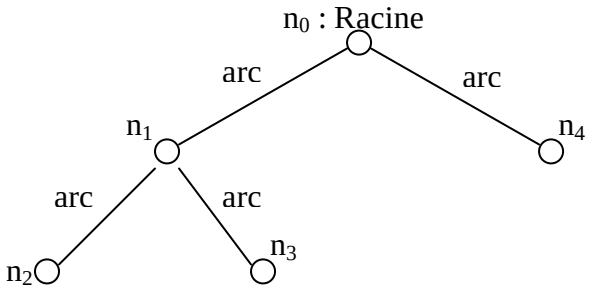
Chapitre 8 : Les Arbres

8.1 Définitions :

a) Arbre :

L'arbre est une structure de donnée récursive constituée :

- d'un ensemble de points appelés ***nœuds,***
- d'un nœud particulier appelé ***racine***,
- d'un ensemble de couples (n_1, n_2) reliant le nœud n_1 au nœud n_2 appelés ***arcs*** (ou arêtes). Le nœud n_1 est appelé ***père*** de n_2. Le nœud n_2 est appelé ***fils*** de n_1.



b) Feuilles :

Les nœuds qui n'ont aucun fils sont appelés feuilles ou nœuds terminaux (les nœuds n_2 , n_3 et n_4 sont des feuilles).

c) Chemin :

On appelle chemin la suite de nœuds n_0 n_1 … n_k telle que (n_{i-1} , n_i) est un arc pour tout i ϵ { 0,…,k}. L'entier k est appelé longueur du chemin n_0 n_1 … n_k .
k c'est aussi le nombre d'arcs.

Le nombre d'arcs d'un arbre = nombre de nœuds - 1.

d) Sous-arbre :

Les autres nœuds (sauf la racine n_0) sont constitués de nœuds fils, qui sont eux même des arbres. Ces arbres sont appelés sous-arbres de la racine.
Exemple : Les nœuds n_1 n_2 et n_3 constituent un sous-arbre.

e) Hauteur :

La hauteur d'un nœud est la longueur du plus long chemin allant de ce nœud jusqu'à une feuille.
La hauteur d'un arbre est la hauteur de la racine (nombre de nœuds).

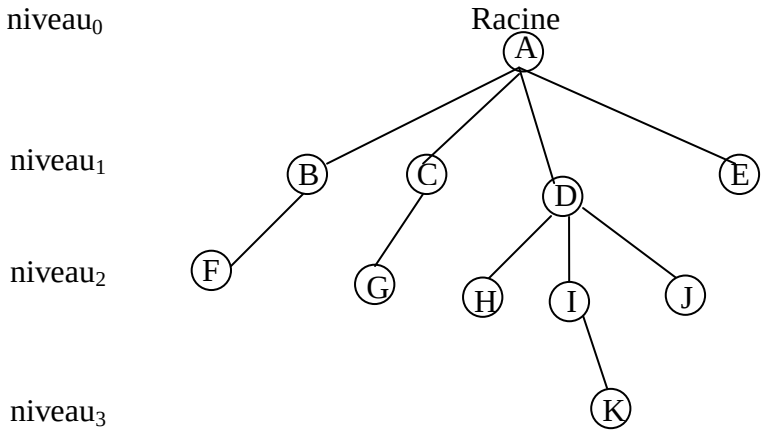
f) Niveau ou profondeur :

La profondeur d'un nœud est la longueur du chemin allant de la racine jusqu'à ce nœud. Tous les nœuds d'un arbre de même profondeur sont au même niveau.
Exemple : Les nœuds n_1 et n_4 ont la même profondeur et sont donc au même niveau.

g) Ascendance et Descendance :

Soit un nœud ***a*** et un nœud ***b*** s'il existe un chemin du nœud ***a*** au nœud ***b*** on dit que ***a*** est un ascendant de ***b*** ou que ***b*** est un descendant de ***a***.

Exemple récapitulatif:

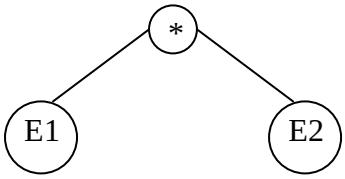


La racine c'est : A
Les nœuds fils de A sont : B C D E
Le nombre de sous-arbres = 4
Le père de F c'est B
B est un ascendant de F
F est un descendant de B
Les feuilles de l'arbre sont : F G H K J E
La hauteur de l'arbre = 4 (nombre de nœuds)
La longueur du chemin A-F = 2 (nombre d'arcs)
La profondeur de l'arbre = 3
La profondeur du nœud G = 2

Un arbre peut aussi être représenté sous forme parenthésée :
(A (B(F), C(G), D(H, I(K), J), E))

h) Arbre étiqueté:

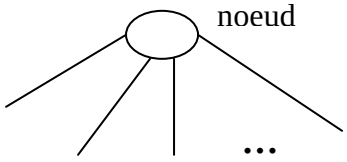
Un arbre étiqueté est un arbre dont chaque nœud possède une information ou étiquette. Cette étiquette peut être de nature très variée : entier, réel, caractère, chaine… ou une structure complexe.
Exemple : on peut représenter une expression par un arbre



E1* E2

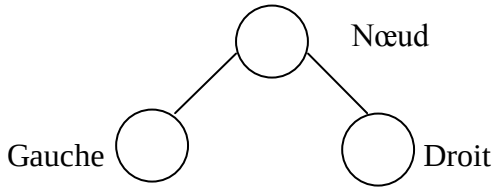
i) Arbre n-aire :

Un arbre n-aire est un arbre dont les nœuds ont au plus n successeurs.



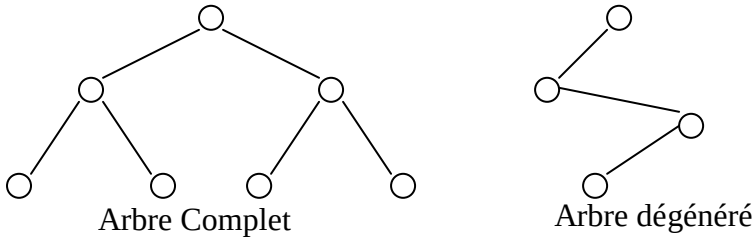
ii) Arbre binaire :

Un arbre binaire est dit binaire si tout nœud de l'arbre a 0, 1, ou 2 successeurs. Ces successeurs sont alors appelés respectivement successeur gauche et successeur droit.



Lorsque tous les nœuds d'un arbre binaire ont deux ou zéro successeurs ont dit que l'arbre est ***homogène*** ou ***complet***

Un arbre binaire est dit ***dégénéré*** si tous ses nœuds n'ont qu'un seul descendant.



Un arbre complet de hauteur ***h*** a un nombre de nœud = 2^h -1 et le nombre de feuilles est $2^{(h-1)}$.

Exemple : *h*=3 nombre de nœuds = 2^3-1=7 nombre de feuilles = $2^{(3-1)}$ = 4

iii) Arbre binaire équilibré: C'est un arbre binaire tel que les hauteurs des deux sous arbres SAG, SAD (sous arbre gauche, sous arbre droit) de tout nœud de l'arbre diffèrent de 1 au plus. Ou encore le nombre de nœuds de SAG et le nombre de nœuds du SAD diffèrent au maximum de 1.

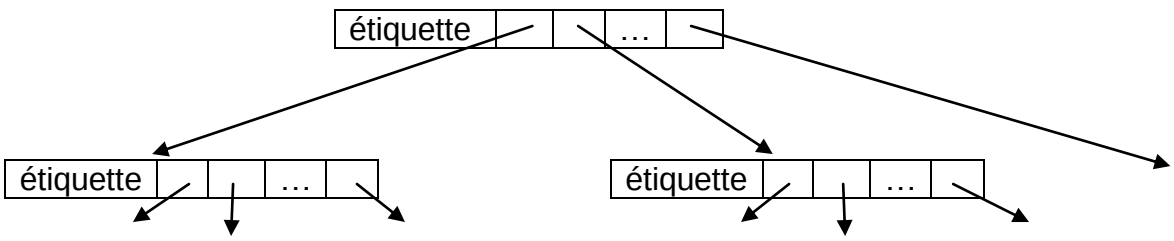
8.2 Représentation des arbres

8.2.1 Arbre n-aire :

a) Une manière de représenter un arbre est d'associer à chaque nœud un enregistrement contenant un ou plusieurs champs pour coder l'étiquette et d'un tableau de pointeurs vers les nœuds fils. La taille du tableau est donnée par le nombre maximum de fils des nœuds de l'arbre.

étiquette	P1	P2	…	P3

Déclaration : typedef struct no *arbre ;
typedef struct no { arbre tab[max_fils] ; /* tableau de pointeurs sur des arbres*/
typelem étiquette ; } nœud ;

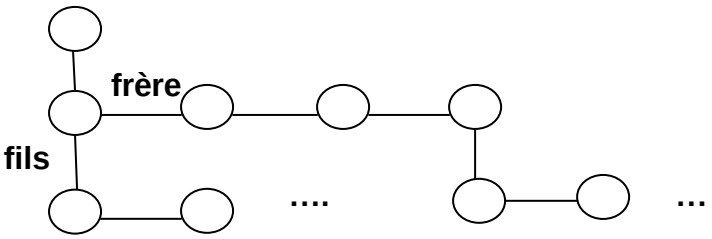


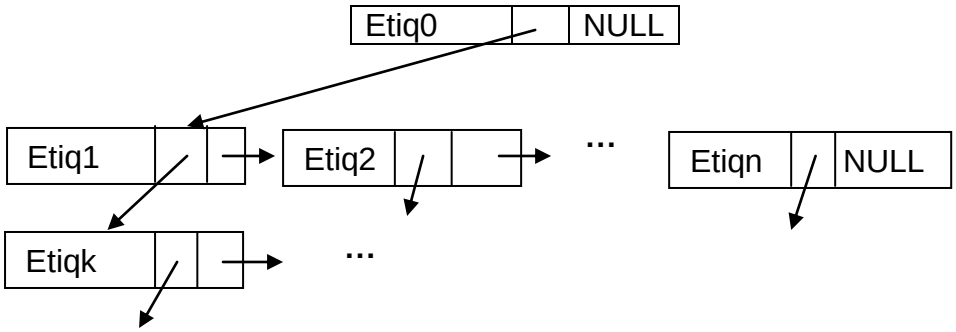
Inconvénients :

- l'arbre contient un petit nombre de nœuds ayant beaucoup de fils. (tableaux de grandes tailles)
- L'arbre contient beaucoup de nœuds ayant peu de fils. (plusieurs tableaux) Ceci conduit à consommer beaucoup d'espace mémoire.

b) Avec deux pointeurs fils et frère.
Afin de contourner l'inconvénient du tableau, on utilise un pointeur vers fils ainée et chaque fils possède un lien vers son frère le plus proche.

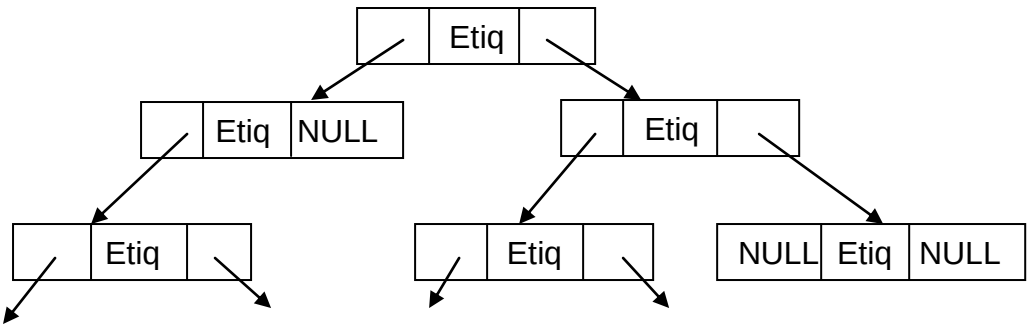
Déclaration : typedef struct no *arbre ;
typedef struct no { arbre fils, frère;
typelem étiquette ; } nœud ;





8.2.2 Arbre binaire :

Déclaration : typedef struct no *arbre ;
typedef struct no { arbre gauche, droit ;
typelem étiquette ; } nœud ;



8.3 Parcours d'un arbre

8.3.1 Parcours d'un arbre n-aire

a) En préordre : à partir d'un nœud quelconque on effectue :

- quelque chose sur ce nœud
- l'ensemble des opérations sur le fils ainé
- « « « « « suivant
- …
- l'ensemble des opérations sur le dernier fils.

b) En postordre : à partir d'un nœud quelconque on effectue :

- l'ensemble des opérations sur le fils ainé
- « « « « « suivant
- …
- l'ensemble des opérations sur le dernier fils.
- quelque chose sur ce nœud

8.3.2 Parcours d'un arbre binaire

a) ***En préordre*** (préfixe) : Racine - Fils gauche - Fils droit (RAC - SAG - SAD[1] ou bien RAC - SAD - SAG).

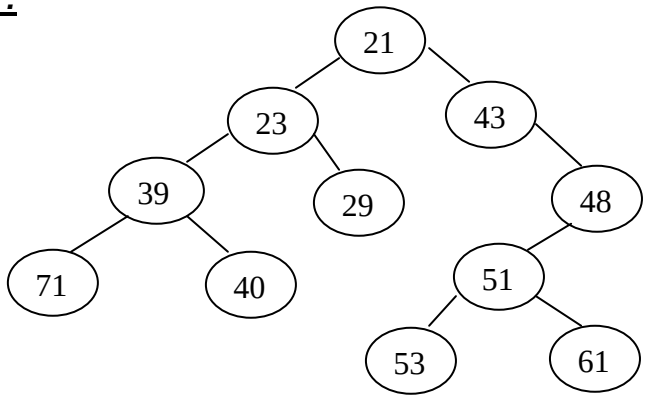
b) ***En postordre*** (Postfixe) : SAG – SAD - RAC ou bien SAD - SAG - RAC.

c) ***En ordre*** (infixe): SAG - RAC - SAD ou bien SAD - RAC - SAG (*arbre binaire uniquement)*

8.4 Arbre binaire ordonné

a) Verticalement : Un arbre binaire est ordonné verticalement, si la clé de tout nœud non feuille est inférieure (respectivement supérieure) à celle de ses fils (et donc par récurrence à celle de tous ses descendants).

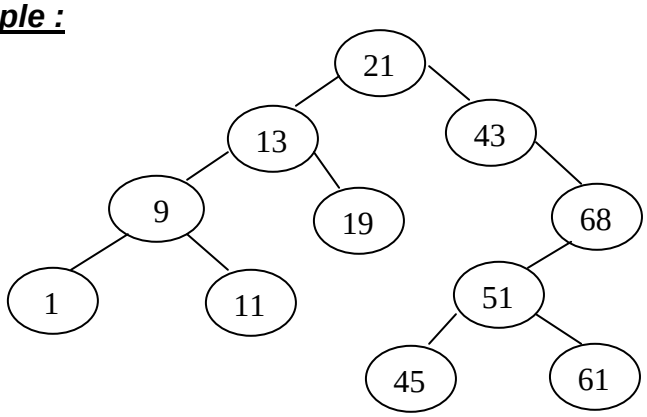
Exemple :



[1] RAC : Racine SAG : Sous Arbre Gauche SAD : Sous Arbre Droit

a) Horizontalement : Un arbre binaire est ordonné horizontalement (de gauche à droite) si la clé de tout nœud non feuille est supérieure ou égale à toutes celles de son sous arbre gauche et inférieure ou égale à toutes celles de son sous arbre droit. Ce type d'arbre est appelé aussi ***arbre binaire de recherche***.

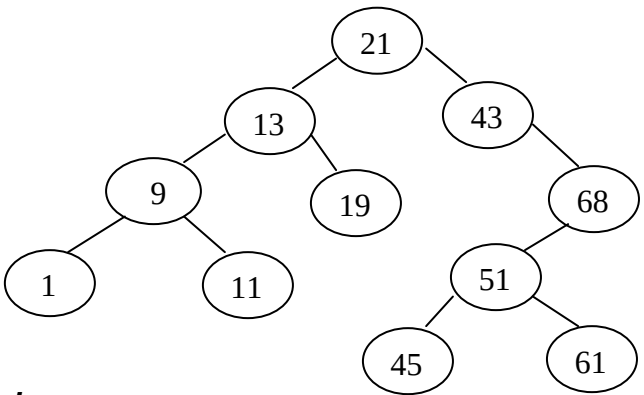
Exemple :



Inconvénient : Un arbre binaire est ordonné horizontalement on a affaire à une relation d'ordre total (on vérifie facilement que l'arbre est totalement ordonné dans un parcours en ordre, la notion d'ordre vertical est seulement une notion d'ordre partiel (il n'y a pas d'ordre à priori entre deux nœuds frères, (ou plus généralement entre deux nœuds d'un même niveau) c'est pourquoi la recherche dans un arbre binaire ordonné verticalement n'est pas dichotomique et peut conduire à un parcours exhaustif de l'arbre.

8.5 Arbre Binaire de Recherche (ABR)

8.5.1 Exemple de parcours d'un ABR:



a) En Préordre :

21 – 13 – 9 – 1 – 11 – 19 – 43 – 68 – 51 – 45 – 61

b) En Postordre :

1 – 11 – 9 – 19 – 13 – 45 – 61 – 51 – 68 – 43 – 21

c) En Ordre :

1 – 9 – 11 – 13 – 19 – 21 – 43 – 45 – 51 – 61 - 68

8.5.1.1 Fonctions récursives de parcours d'un ABR

a) Préordre (Préfixé)

```
void  parcours (arbre A)
{ if (A !=NULL)
    {  afficher(A→etiquette) ;
       parcours(A→gauche) ;
       parcours(A→ droit);
    }
}
```

b) En ordre (Infixé)

```
void  parcours (arbre A)
{ if (A !=NULL)
    {  parcours(A→gauche) ;
       afficher(A→etiquette) ;
       parcours(A→ droit);
    }
}
```

c) En postordre (postfixé)

```
void  parcours (arbre A)
{ if (A !=NULL)
    {  parcours(A→gauche) ;
       parcours(A→ droit);
       afficher(A→etiquette) ;
    }
}
```

8.5.1.2 Fonctions itératives de parcours d'un arbre de recherche

a) Préordre (Préfixé)

```
void  parcours (arbre A)
{ pile s=initpile() ;
  empiler(&s,NULL) ;
  while (A !=NULL)
    {  afficher(A→etiquette) ;
       if (A→droit !=NULL) empiler(&s,A→droit);
       if (A→gauche!=NULL) A=A→gauche;
       else desempiler(&s,&A) ;
    }
}
```

b) En ordre (Infixé)

```
void  parcours (arbre A)
{ pile s=initpile() ;
  while (A !=NULL) { empiler(&s,A); A=A→gauche; }
  while (!pilevide(s))
    {  desempiler(&s, &A) ;
       afficher(A→etiquette) ;
       if (A→droit !=NULL)
         { A=A→droit ;
           while( A !=NULL) { empiler(&s,A) ; A=A→gauche);  }
         }
    }
}
```

c) En postordre (postfixé)

```
void  parcours (arbre A)
{ pile pg=initpile(), pd=initpile() ;
  while (A !=NULL)
   {  empiler(&pg,A);
      if (A->droit !=NULL)
           { empiler(&pg,NULL); empiler(&pd, A->droit) ; }
      A=A->gauche;
   }
  while (!pilevide(pg))
   {  desempiler(&pg, &A) ;
      if (A!=NULL) printf("%d  ",A->etiquette) ;
      else { if(!pilevide(pd))
                { desempiler(&pd,&A);
                     if ( A->gauche ==NULL && A->droit ==NULL)
                                printf("%d  ", A->etiquette);
                     else  while( A !=NULL)
                                { empiler(&pg,A) ;
                                  if (A->droit !=NULL)
                                       { empiler(&pg,NULL);empiler(&pd,A->droit) ;}
                                  A=A->gauche;
                                }
                }
             }
   }
}
```

8.5.2 Recherche dans un ABR

a) Fonction recherche itérative :

```
int recherche(arbre A, typelem val)
{  int trouv=0 ;
   while(trouv==0 && A !=NULL)
    if (val==A→etiquette) trouv=1;
    else if (val <A→etiquette) A= A→gauche ;
           else A=A→droite ;
  return (trouv) ;
}
```

b) Fonction recherche récursive

```
int recherche(arbre A, typelem val)
{  If (A ==NULL) return 0 ;
   else if (val==A→etiquette) return 1;
        else if (val <A→etiquette) return recherche(A→gauche,val);
             else return recherche(A→droite,val) ;
}
```

8.5.3 Insertion d'un élément dans un ABR

Remarques : - Les étiquettes dans un arbre binaire ordonné horizontalement sont toujours uniques (elles ne sont pas dupliquées), donc l'élément à insérer est toujours une feuille.
- On utilise une autre fonction de recherche qui retourne (en paramètre) l'adresse de l'élément précédent.

```
int Recherche(arbre A, arbre *prd, typelem Val)
{  if (A==NULL) return(0);
   else { if (A->etiquette==Val) return(1);
          else { if (A->etiquette>Val) return(Recherche(A->gauche, &A ,Val));
                 else return (Recherche(A->droit, &A ,Val));
               }
        }
}
```

```
void Insert (arbre *racine,typelem Val)
{ arbre prd=NULL, A;
    If (Recherche(*racine,&prd,Val)==1) printf("Existe deja\n");
    else { A=(arbre)malloc(sizeof(noeud));
           A->etiquette=Val;   A->gauche=NULL;  A->droit=NULL;
           if (prd!=NULL)
                 if (Val<prd->etiquette) prd->gauche=A;
                 else prd->droit=A;
           else *racine=A; /* Quand l'arbre est vide  */
         }
}
```

8.5.4 Suppression d'un élément dans un ABR

Trois types de suppressions se présentent à nous :

a) Si l'élément est une feuille, alors on le supprime simplement. On a deux cas :

a.1) supprimer une feuille qui se trouve à gauche d'un nœud.
Exemple : prd->gauche=NULL ; free(A) ;

a.2) supprimer une feuille qui se trouve à droite d'un nœud.
Exemple : prd->droit=NULL ; free(A) ;

b) Si l'élément n' a qu'un seul descendant, alors on le remplace par ce descendant. On a quatre cas :

b.1) Exemple : prd->droit=A->droit ; free(A) ;

b.2) Exemple : prd->droit=A->gauche ; free(A) ;

b.3) Exemple : prd->gauche=A->droit ;free(A) ;

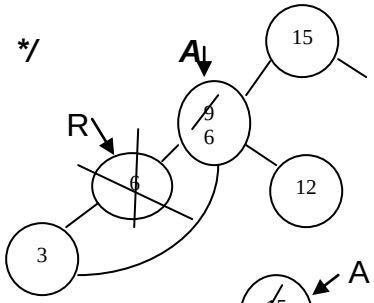
b.4) Exemple : prd->gauche=A->gauche ; free(A) ;

c) Si l'élément a deux descendants, on le remplace au choix soit par :
- L'élément le plus à droite du SAG (la valeur max)
- L'élément le plus à gauche du SAD (la valeur min)

On a deux cas :

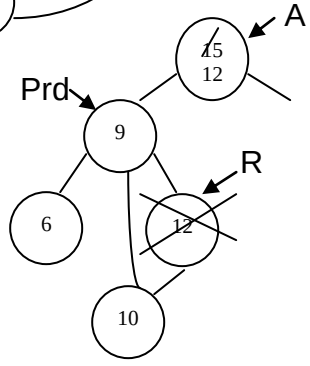
c.1) Exemple : ***/* R l'élément le plus à gauche */***

A->etiquette = R->etiquette ;
A->gauche=R->gauche ;
free(R) ;



c.2) Exemple :

A->etiquette = R->etiquette ;
prd->droit=R->gauche ;
free(R) ;



```
void suppFeuille(arbre prd, arbre A)
{   if (Val<prd->etiquette) prd->gauche=NULL;
    else prd->droit=NULL;
    free(A);
 }

void supp1Fils(arbre prd, arbre A)
{ if (A->gauche==NULL)
          if (Val<prd->etiquette) prd->gauche=A->droit;
          else prd->droit=A->droit;

  else  if (Val<prd->etiquette)  prd->gauche=A->gauche;
         else prd->droit=A->gauche;

  free(*A) ; *A=NULL;
 }

void remplace(arbre R, arbre A, arbre prd)                    /* 2 descendants */
{ if (R->droit!=NULL)  { prd=R;  remplace(R->droit, A, prd); }
  else { A->etiquette=R->etiquette;
           if (prd!=NULL) prd->droit=R->gauche;
           else A->gauche=R->gauche;
           free(R);
         }
}

void Supprim(arbre *racine, typelem Val)
{ arbre A, prd=NULL;

  if (Recherche(*racine, &prd, Val)==0)
                printf(" L'élément à supprimer n'existe pas\n");

  else {  if (val<prd->etiquette)    A=prd->gauche;   else A=prd->droit;

             /* Racine */
           if (prd==NULL && A->gauche==NULL && A->droit==NULL)
                { free(A);  printf("Arbre Vide\n");  *racine=NULL;  }

           else   /*  Feuille  */
                  if (A->gauche==NULL && A->droit==NULL)  suppFeuille(prd, A) ;

                       /* 1 seul descendant */
                  else if (A->gauche==NULL)  || (A->droit==NULL) supp1Fils(prd, A) ;

                            /* 2 descendants */
                       else remplace(A->gauche, A, NULL);
                                                  ↖
         }                                          Sous arbre gauche
}
```

8.5.5 Construction d'un arbre binaire de recherche

```
void constarbre(tab t, int i, int n, arbre *racine)
{
    if (i<n) { Insert(racine, t[i]);
               constarbre(t, i+1, n, racine);
             }
}
```

/ Autre fonction supprime */*

```
void Supprim(arbre *racine, typelem Val)          L'adresse de VAL l'élément
{ arbre A, prd=NULL;                                  à supprimer
  if (Recherche(*racine, &prd, Val, &A)==0)
              printf(" L'élément à supprimer n'existe pas\n");
  else { if (prd==NULL && A->gauche==NULL && A->droit==NULL) /* Racine */
              { free(A);  printf("Arbre Vide\n");  *racine=NULL;}
         else {   /*  Feuille  */
                  if (A->gauche==NULL && A->droit==NULL)
                    { if (Val<prd->etiquette) prd->gauche=NULL;
                      else prd->droit=NULL;
                      free(A);
                    }
                  else if (A->gauche==NULL)  /* 1 seul descendant */
                         { if (Val<prd->etiquette) prd->gauche=A->droit;
                           else prd->droit=A->droit;
                           free(A);
                         }
                       else if (A->droit==NULL)  /* 1 seul descendant  */
                              { if (Val<prd->etiquette)prd->gauche=A->gauche;
                                else prd->droit=A->gauche;
                                free(A);
                              }
                            else remplace(A->gauche, A, NULL);
              }                                          Sous arbre gauche
       }
}
```