Chapitre 8
Les arbres binaires de recherche (ABR)

Module: structures de données et programmation C

2éme LEESM

mlahby@gmail.com

30 mai 2021

Plan

Définition

Définition

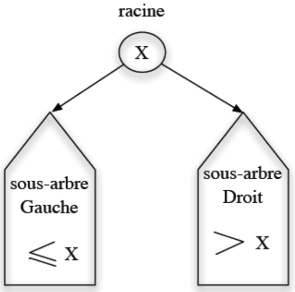
Un **arbre binaire de recherche** (ABR) est un **arbre binaire** danslequel, tous les noeuds du sous-arbre gauche d'un noeud ont une valeur inférieure (ou égale) à la sienne et tous les noeuds du sous-arbre droit ont une valeur supérieure à la valeur du noeud lui-même.

intérêt

- les arbres de recherche ont un intérêt quand il y a un grand nombre d'ajouts et de suppressions.
- On peut utiliser la structure ABR dans le but de réaliser des algorithmes de tris rapides, exemple tri par tas.

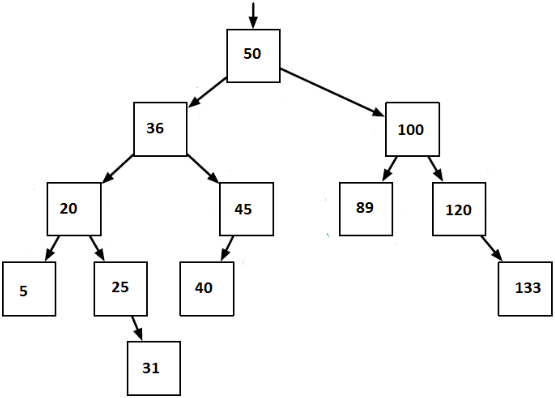
Définition récursive d'un ABR

- Soit vide
- Soit composé
  - un noeud racine contient un élément X
  - de 2 sous arbres binaires de recherche ABRG et ABRD disjoints
    - * dans ABRG $\longrightarrow$ noeuds $\leq$ X
    - * dans ABRD $\longrightarrow$ noeuds $>$ X



Représentation graphique d'un arbre binaire

⇒ **Exemple :**



Implantation d'un ABR avec deux pointeurs

Définition d'un arbre binaire de recherche

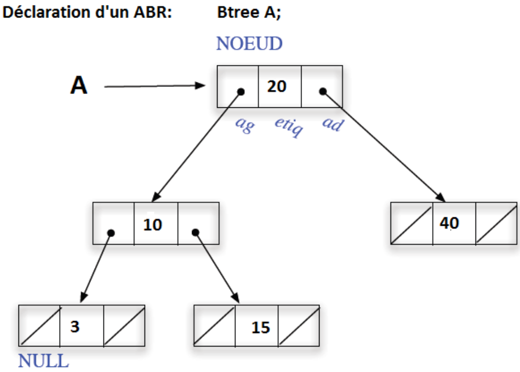
```
/* Définition du type Element */
typedef int Element;
/* Définition du type noeud d'un arbre binaire */
typedef struct noeud{
    Element etiq; /*le champ etiq peut avoir n'importe quel type*/
    struct noeud *fg; /*pointeur contenant l'adresse du ABG*/
    struct noeud *fd; /*pointeur contenant l'adresse du ABD*/
}Tnoeud;
/* Définition du type Btree */
typedef struct *Tnoeud Btree
```

Déclaration d'un arbre binaire de recherche

$$Btree \quad A$$

Implantation d'un ABR avec deux pointeurs

⇒ **Exemple :**



Rechercher un élément v dans un arbre ordonné a

Principe

Au lieu de parcourir tout l'arbre, on ne parcourt que la partie qui peut contenir l'élément.

3 cas

1. $(a->etiq) == v$ Alors trouvé
2. $(a->val) < v$ rechercher dans sous-arbre gauche.
3. $(a->val) > v$ rechercher dans sous-arbre droit.

Rechercher un élément v dans un arbre ordonné a

- La fonction **Rechercher()** teste si v est un élément de l'arbre binaire de recherche a,
- elle renvoie 1 si v dans a, et 0 sinon.

Définition de la fonction

```
int Rechercher(Btree a, Element v)
    {
    if (EstVide(a))
        return 0;
    if (a->etiq==v)
        return 1;
    if (a->etiq<v)
        Rechercher(a->fg,v)
    if (a->etiq>v)
        Rechercher(a->fd,v)
    }
```

Insertion d'un élément v dans un ABR a comme une feuille

Principe

- On compare v et a$->$etiq :
  - Cas 1 : égale ou inférieure :

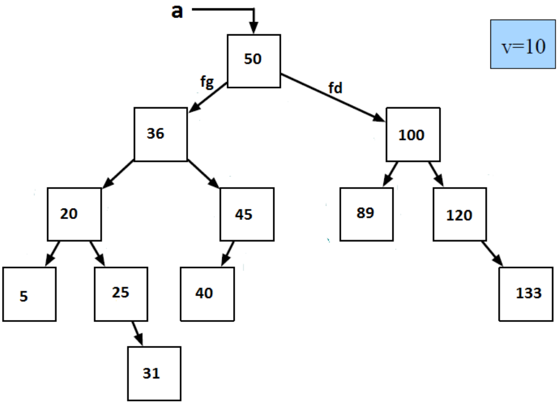
    $\rightarrow$**on va** à gauche
  - Cas 2 : supérieure :

    $\rightarrow$**on va** à droite
- «**on va**» : si pointeur != NULL, on considère SS Arbre
- si pointeur == NULL on crée une feuille

.

Insertion d'un élément v dans un ABR a comme une feuille

⇒ **Exemple :**



Insertion d'un élément v dans un ABR a comme une feuille

- La fonction **InsertFeuille** permet d'ajouter une nouvelle feuille dans une ABR.
- Elle fait appel à la fonction **CreerFeuille**.

Définition de la fonction Btree InsertFeuille(Btree a, Element v)

```
Btree InsertFeuille(Btree a, Element v)
    {if (EstVide(a))
        a=CreerFeuille(v);
    else
        if (a->etiq≥v)
            a->fg = InsertFeuille(a->fg,v );
        else
            a->fd = InsertFeuille(a->fd,v );
    return a;
    }
```

Insertion d'un élément v dans un ABR a comme une feuille

- La fonction **CreerFeuille** renvoie un arbre binaire dont la racine est e et les fils gauche et droit sont vides.

Définition de la fonction arbreBin CreerFeuille(Element e)

```
Btree CreerFeuille(Element e)
    {Btree n;
    n=(Btree)malloc(sizof(Tnoeud));
    n->etiq=e;
    n->fg=NULL;
    n->fd=NULL;
    return n;
    }
```

Insertion d'un élément v dans un ABR a comme une racine

Principe

- Pour ajouter X comme racine dans A on applique les deux étapes suivantes :

1. Couper A en 2 arbres binaires de recherche (ARBIN) G et D
   $\longrightarrow$ **G** contient tous les éléments de $A \leq X$
   $\longrightarrow$ **D** contient tous les éléments de $A > X$
2. Former un nouvel ARBIN A' dont la racine avec :
   - étiquette de A' $\longleftarrow$ X
   - ABG(A') $\longleftarrow$ G
   - ABD(A') $\longleftarrow$ D

.

Insertion d'un élément <u>v</u> dans un ABR <u>a</u> comme <u>une racine</u>

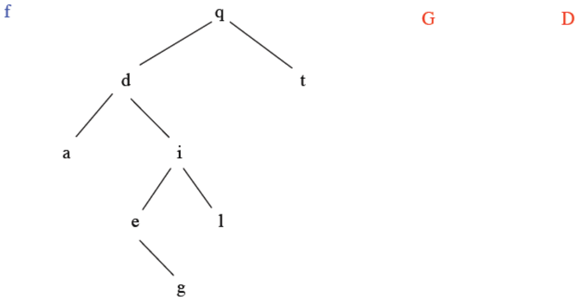
Principe de coupure

- Il n'est pas nécessaire de parcourir TOUS les noeuds de A $\longrightarrow$ seulement les noeuds N situés sur le chemin de recherche de X dans A
- si noeud N $\leq$ X : **G** $\leftarrow$ **N + ABG(N)** sur le bord droit de G
- si noeud N $>$ X : **D** $\leftarrow$ **N + ABD(N)** sur le bord gauche de D

.

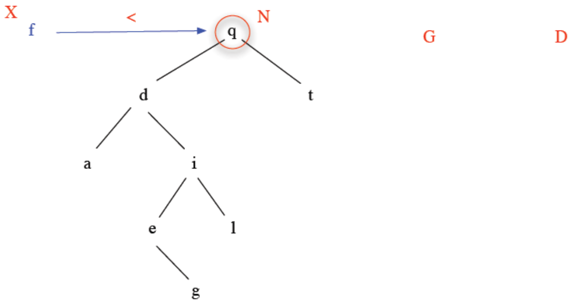
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



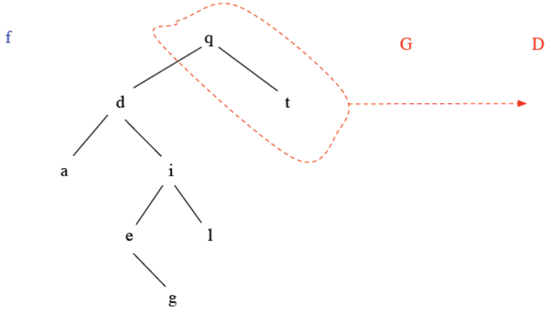
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



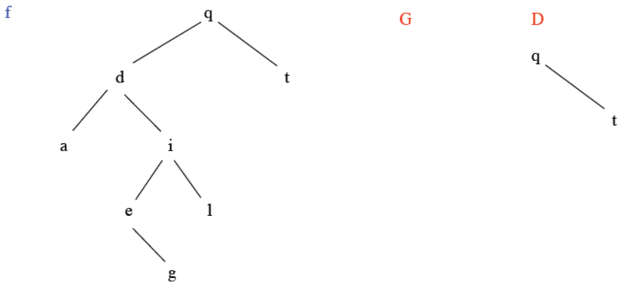
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



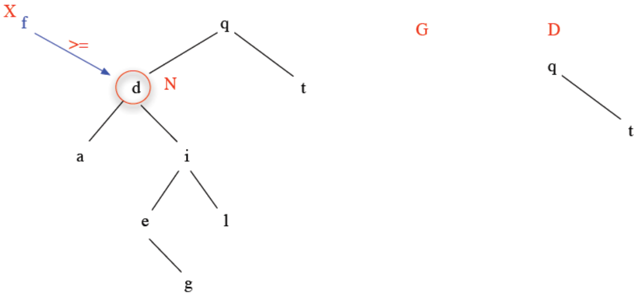
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



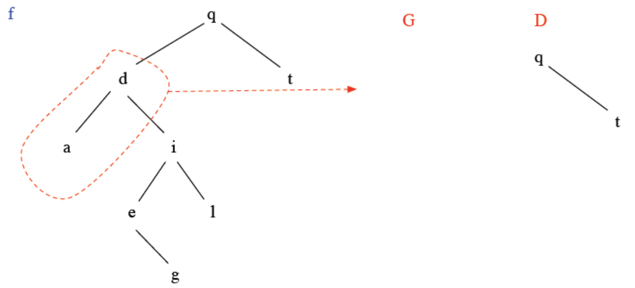
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



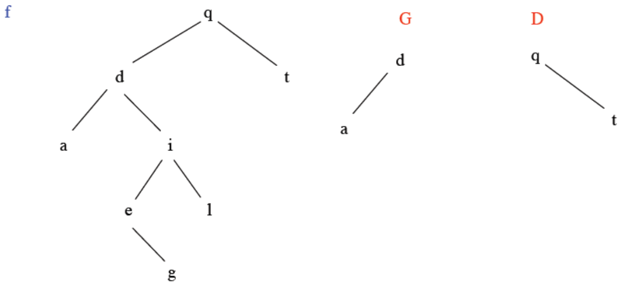
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



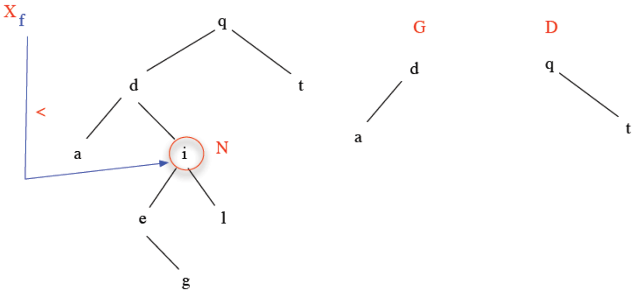
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



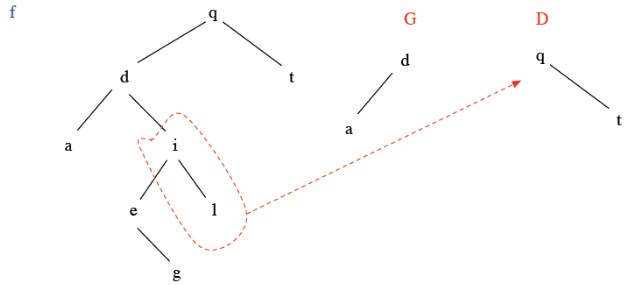
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



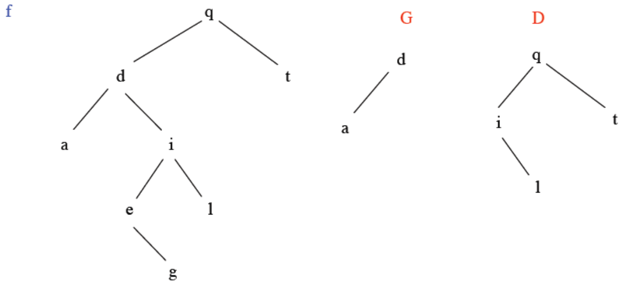
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



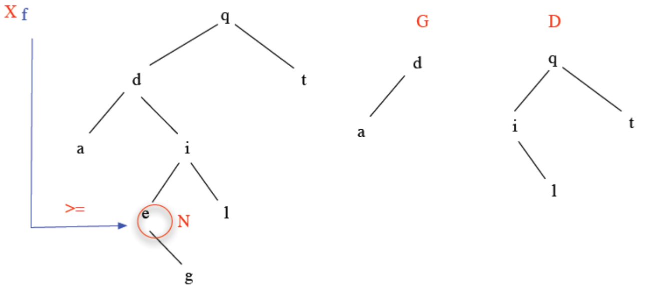
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



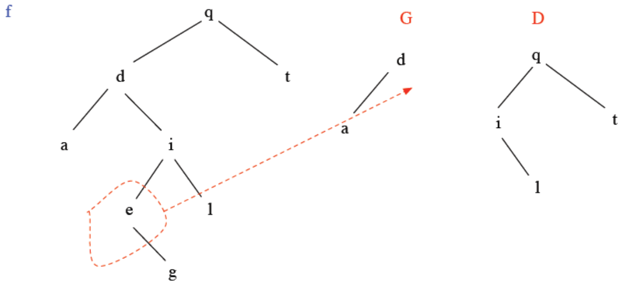
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



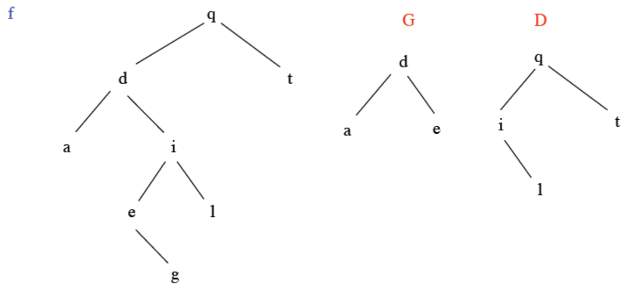
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



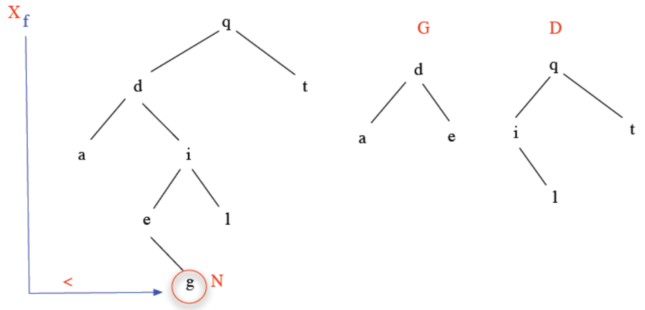
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



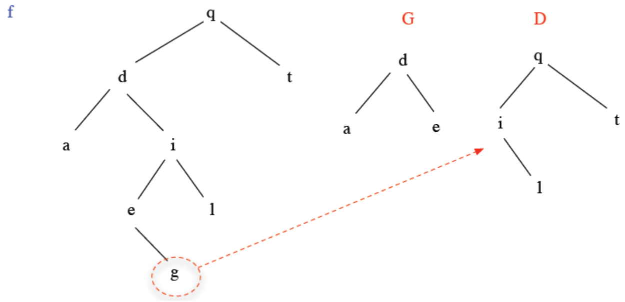
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



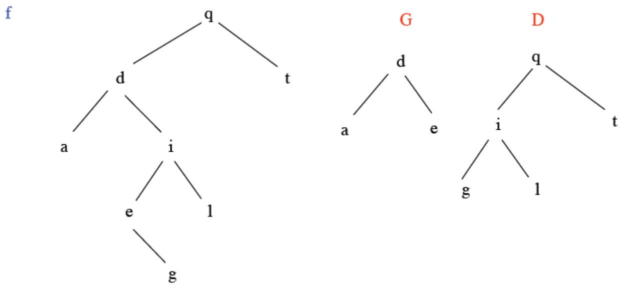
Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



Insertion d'un élément v dans un ABR a comme une racine

⇒ **Exemple de coupure**



Insertion d'un élément v dans un ABR a comme une racine

Algorithme : de coupure de A en 2 ARBIN G et D

Procedure Coupure(ELEMENT X, Btree A, G , D)
Début
  SI A est vide **ALORS**
    G $\leftarrow$ vide
    D $\leftarrow$ vide
  SINON
    SI X $\leq$ A **ALORS**
      D $\leftarrow$ A
      Coupure(X, ABG(A) , G, ABG(D))
    SINON
      G $\leftarrow$ A
      Coupure(X, ABD(A) , ABD(G) , D)
    FSI
  FSI

Fin

Insertion d'un élément v dans un ABR a comme une racine

Algorithme : Ajout d'un élément X à la racine de l'arbre A

Btree AjouterRacine(ELEMENT X, Btree A)
Variables Btree R;
Début

    étiquette de R ← X
    ABG(R) ← vide
    ABD(R) ← vide
    Coupure (X, A,ABG(R), ABD(R))
    A ← R
    retourner A;

Fin

Suppression d'un élément v dans un ABR A

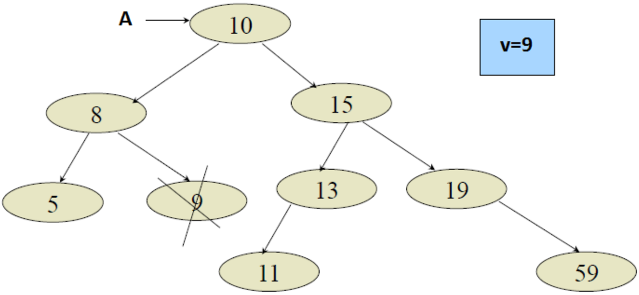
Principe

Pour supprimer un élément **v** dans **A** il faut :

1. déterminer la place de v dans A → noeud N : cela se fait d'une manière récursive (même principe que l'algorithme de recherche)
2. supprimer v avec réorganisation des éléments de A.
3. On distingue 3 cas :
    - **Le noeud N à 0 fils** : suppression immédiate
    - **Le noeud N à 1 fils** : on remplace v par ce fils
    - **Le noeud N à 2 fils** : 2 solutions
        - remplacer v par l'élément qui lui est immédiatement inférieur → Le MAX dans ABG(N)
        - remplacer v par l'élément qui lui est immédiatement supérieur → Le MIN dans ABD(N)

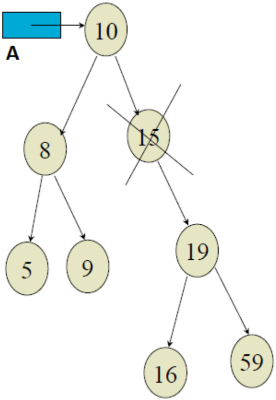
1^{er} Cas : Suppression d'un noeud à 0 fils dans un ABR A

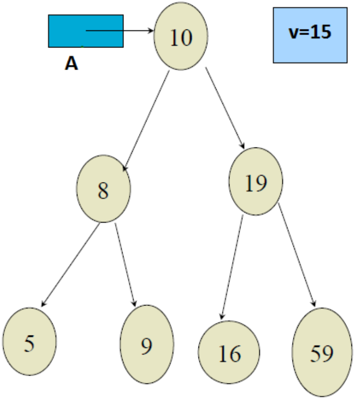
⇒ **Exemple :**



2^{eme} Cas : Suppression d'un noeud à 1 fils dans un ABR A

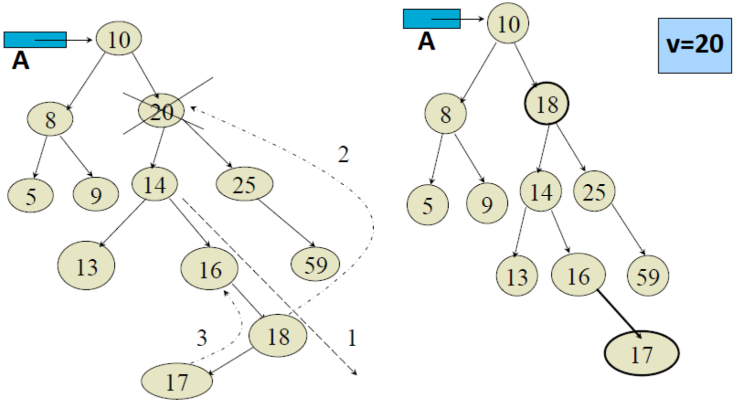
⇒ **Exemple :**





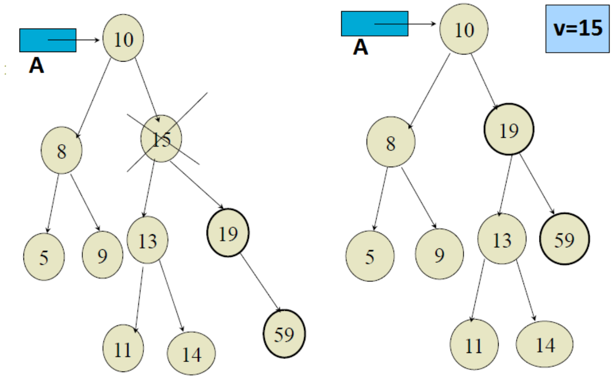
3^{eme} Cas : Suppression d'un noeud à 2 fils dans un ABR A

⇒ **Solution 1 :** le noeud est remplacé par le plus grand élément du SS arbre gauche



3^{eme} Cas : Suppression d'un noeud à 2 fils dans un ABR A

⇒ **Solution 2 :** le noeud est remplacé par le plus petit élément du SS arbre droit



Suppression du MAX

- La fonction **supprimerMax(Element *MAX, Btree A)** retourne l'élément le plus grand dans MAX et supprime cet élément de A
- L'élément le plus grand dans un arbre est l'élément le plus à droite.

Définition de la fonction

```
Btree supprimerMax(int *MAX, Btree A)
{
    if(A− >fd == NULL)
        {
            /*récupère la valeur dans MAX*/
            *MAX= rac− >etiq ;
            /*récupère l'adresse de fg*/
            A= A− >fg ;
        }
    else
            A− >fd = supprimerMax(MAX, A− > fd) ;
    return A ;
}
```

Suppression d'un élément v dans un ABR A

- La fonction **Supprimer()** supprime l'élément v dans A. Retour A si v $\notin$A sinon A-noeud(v)

Définition de la fonction Btree Supprimer(Element v,Btree A)

```
Btree Supprimer(int v,Btree A)
    { Btree temp;
       Element max;
       if ( A != NULL )
       if (v < A->etiq )
             A->fg = supprimer(x, A->fg)
       else
             if (v > A->etiq )
                   A->fd = supprimer(x, A->fd)
             else /* On a trouvé l'élément */
                   if( A ->fg == NULL ) /*0 ou 1 fils */
                        { tmp=A->fd;
                        free(A);
                        A=tmp;}
```

Suppression d'un élément v dans un ABR A

Suite du code

```
        else /*le fils gauche n'est pas vide */
            if( A - >fg == NULL ) /*1 fils */
                {
                tmp=A- >fd ;
                free(A) ;
                A=tmp ;
                }
            else /* il y a les deux fils 1ere solution */
                {
                A- >fg = supprimerMax(&max, A- >fg) ;
                A- >etiq = max ;
                }
return A ;
}
```