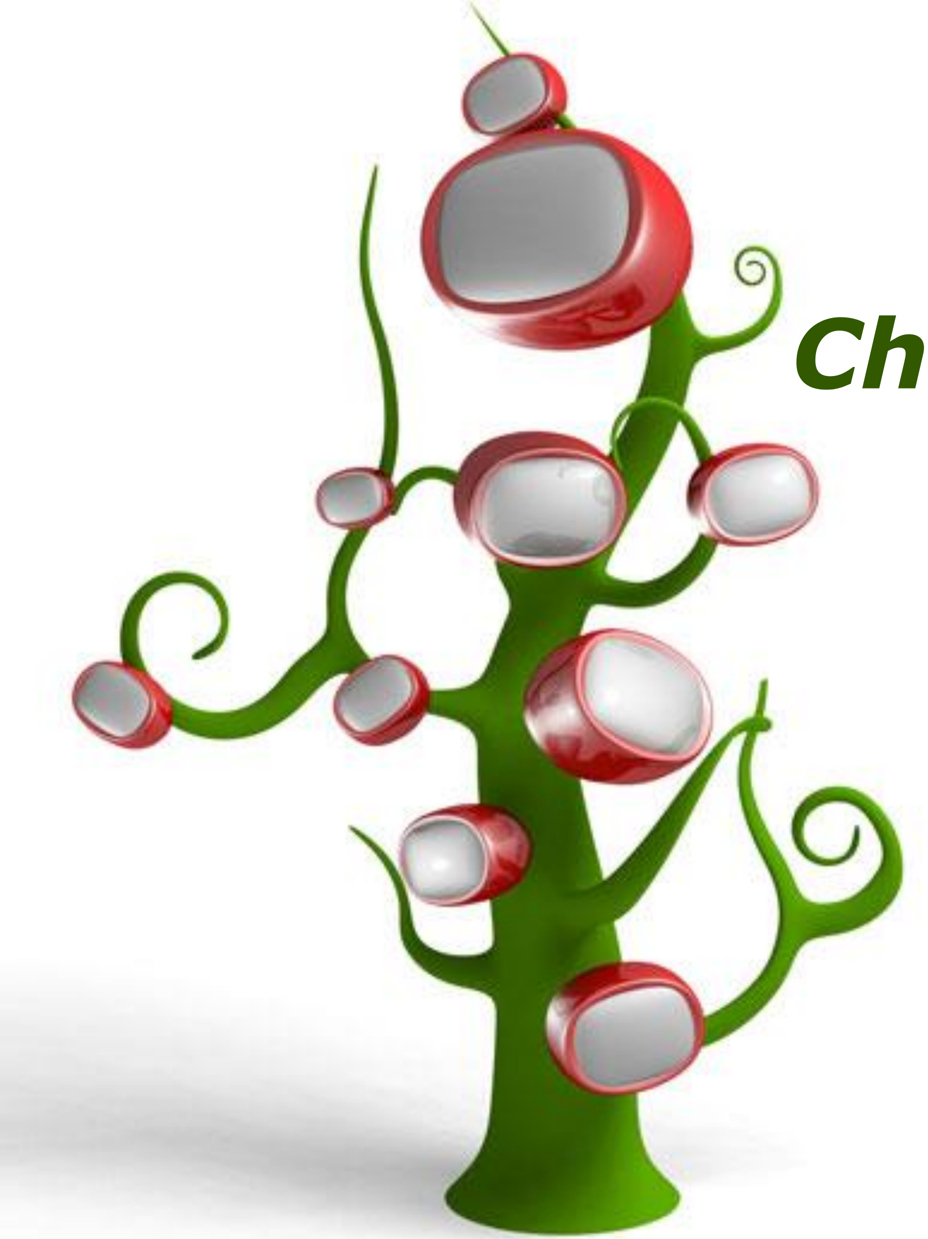


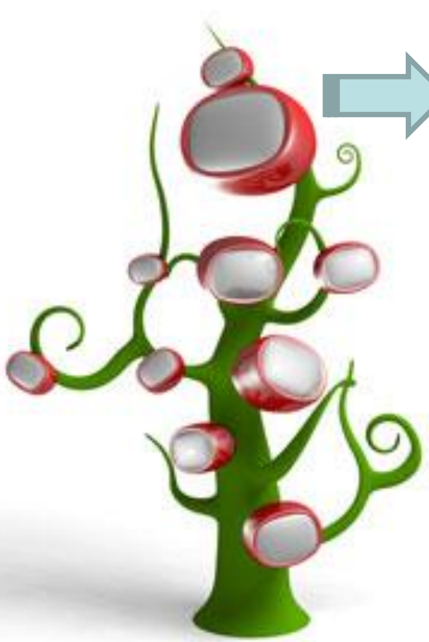
Chapitre 4 (Suite)

Les Arbres Binaires de Recherche

Un arbre binaire est une structure qui permet de regrouper un ensemble de données et de les structurer de manière hiérarchique.

Existe - t – il une manière efficace de localiser un élément parmi un grand nombre d’éléments dans un arbre binaire ?

Peut-on organiser les données d’un arbre binaire de manière analogue à un tableau trié ?



1 Propriété fondamentale:

❑ Définition:

Un Arbre Binaire de Recherche (ABR) est une structure fondamentale pour la représentation d'ensembles dotés d'un ordre. C'est un AB particulier où :

✓ Toutes les valeurs inférieures ou égales à celle de la racine sont stockées dans le descendant gauche de la racine.

✓ Toute les valeurs supérieures à celle de la racine sont stockées dans le descendant droit.

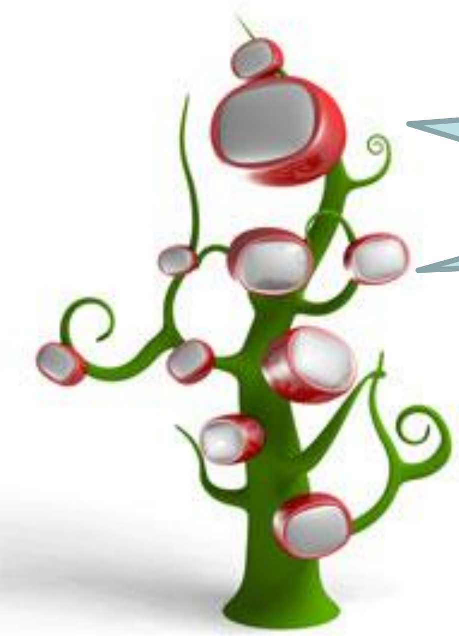
Ainsi, tout les nœuds à gauche d'un nœud valant x portent des valeurs inférieures ou égales à x et

tout ceux à droite portent des valeurs supérieures ou égales à x.

Autrement dit, pour tout noeud n (valant x) de l’arbre, on s’assure que :

max_val(n.sous_arbre_gauche) ≤ n.val < min_val(n.sous_arbre_droit)

Cette inéquation qui exprime la condition est appelée:

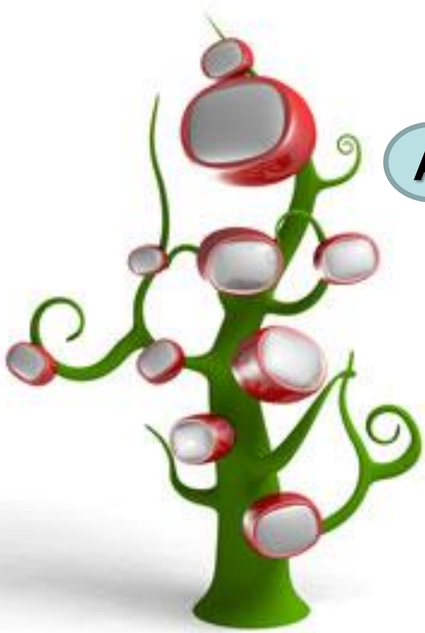


Propriété fondamentale des ABR.

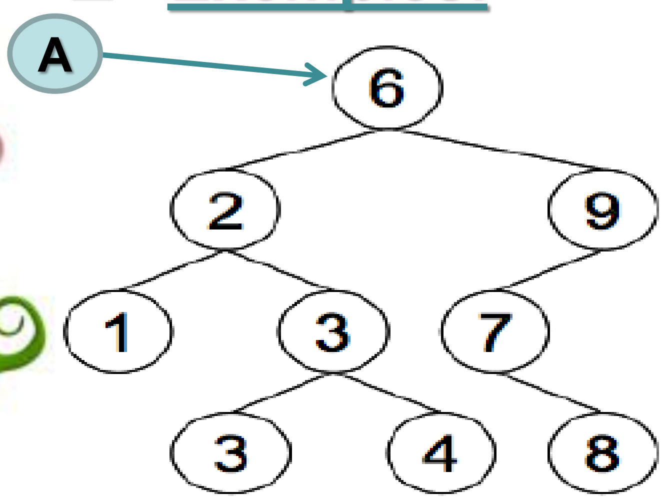
La définition d’un ABR est récursive, on dit que :

A est un ABR si :

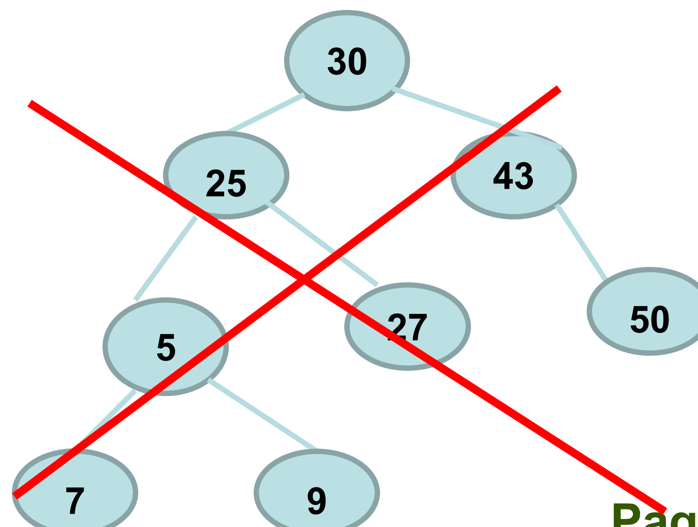
- ✓ A est vide ;
- ✓ A = < A_1, x , A_2 > tel que A_1 et A_2 sont des ABRs et tout nœud de A_1 est plus petit (ou égal) que x et tout nœud de A_2 est plus grand que x.



❑ Exemples:



Exemple d’un ABR



Remarques:

- ✓ Toute insertion ou suppression d'un nœud doit maintenir la propriété fondamentale
- ✓ Le parcours infixé de l'arbre donne une liste triée.

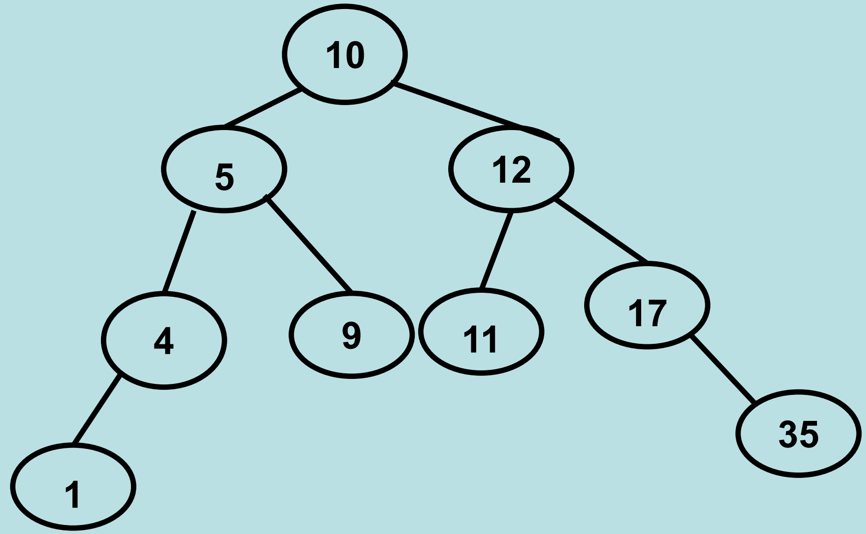
Application:

1. Soit T un ABR vide. Faire un schéma représentant T après l'insertion dans l'ordre des nombres suivants :

   10, 5, 12, 4, 11, 1, 17, 9, 35

2. Donner les listes des valeurs obtenues avec les parcours préfixé, postfixé et infixé.

1.



2.

✓Préfixé: 10, 5, 4, 1, 9, 12, 11, 17, 35

✓ Postfixé: 1, 4, 9, 5, 11, 35, 17, 12, 10

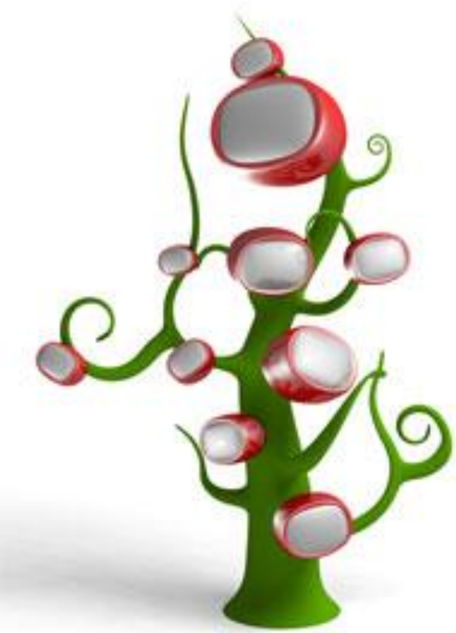
✓Infixé: 1, 4, 5, 9, 10, 11, 12, 17, 35

2 Recherche dans un ABR:

La propriété fondamentale rend le test d'appartenance à un ensemble représenté par un ABR particulièrement aisé.

Pour savoir si x est un élément de l'ensemble, on commence par comparer x à l'élément **r** situé à la racine de l'arbre :

- si x = **r**, la réponse à la question «*x est-il dans l'ensemble?* » est immédiate et est positive ;
- si x < **r**, (à supposer qu'il soit dans l'ensemble), x ne peut être qu'un descendant du fils gauche de la racine ;
- si x > **r**, x ne peut être qu'un descendant du fils droit de la racine.

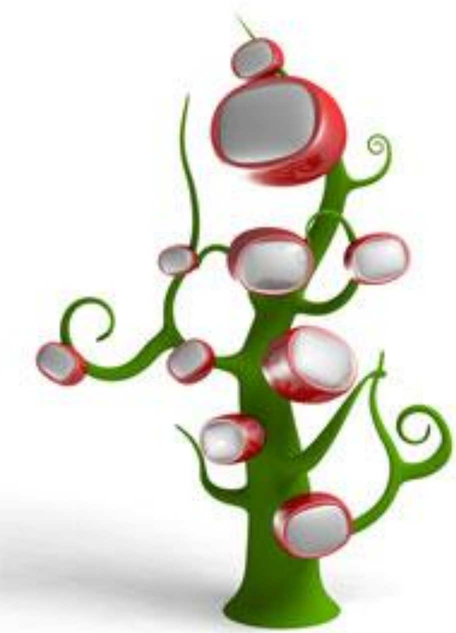


La fonction Recherche(A, x) permettant de tester l'appartenance de l'élément x dans l'ABR A sera donc de nature récursive.

❑ Application:

Enoncer la fonction récursive Recherche (x, A)

```
Fonction  Recherche (A: ABR, x: entier ): ^nœud

Début
    si  (A ≠  nil)  alors
              si  (x= A^.val ) alors  Retourner ( A)
              sinon
                 si  ( x < A^.val  ) alors
                      Retourner (Recherche (A^.fg, x))
                  sinon
                      Retourner (Recherche (A^.fd, x))
                  finsi
            finsi
     sinon
               Retourner ( Nil )
     finsi

Fin
```



3 Insertion d’un élément dans un ABR:

La procédure d’insertion permettant l’ajout de l’élément x à l’ensemble A peut être définie comme suit :

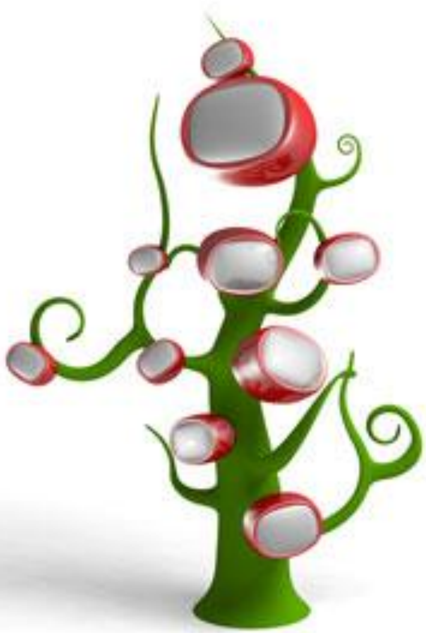
On commence par vérifier si A est vide :

✓ Si c’est le cas, on crée un nouveau nœud pour contenir x et on fait pointer A dessus ;

✓ Sinon, on cherche à positionner x de la même façon que dans la fonction Recherche().

❑ Application:

Enoncer la Procédure récursive Insérer (A, x)

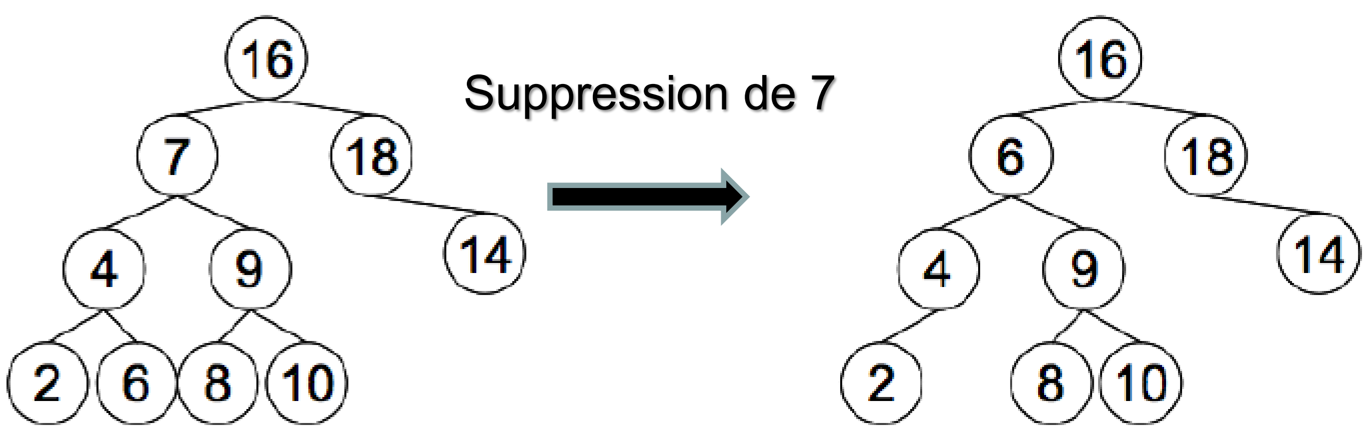


```
Procédure  Insérer ( var A: ABR, x: entier )

Début
     si  (A  =  nil)  alors
               Allouer (A)
               A^.val ← x
               A^.fg ← Nil
               A^.fd ← Nil
     sinon
               si  ( x ≤ A^.val ) alors
                    Insérer (A^.fg, x)
                  sinon
                    Insérer (A^.fd, x)
                  finsi
     finsi

Fin
```

4 Suppression d'un élément dans un ABR:

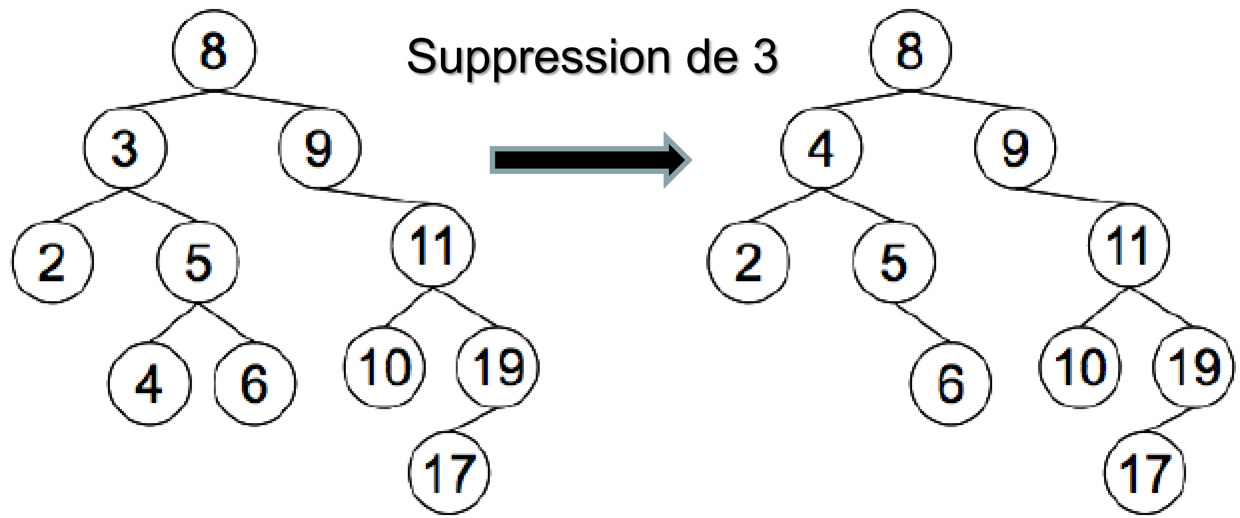
Le principe de la procédure permettant la suppression d'un élément x de l'ABR est le suivant :

- ✓ Si l'élément est une feuille, alors on supprime simplement ;
- ✓ Si l'élément n'a qu'un seul fils, alors on le remplace par celui-ci
- ✓Si l'élément a deux fils, on le remplace au choix :
  a) soit par l'élément le plus à droite du sous-arbre gauche ;



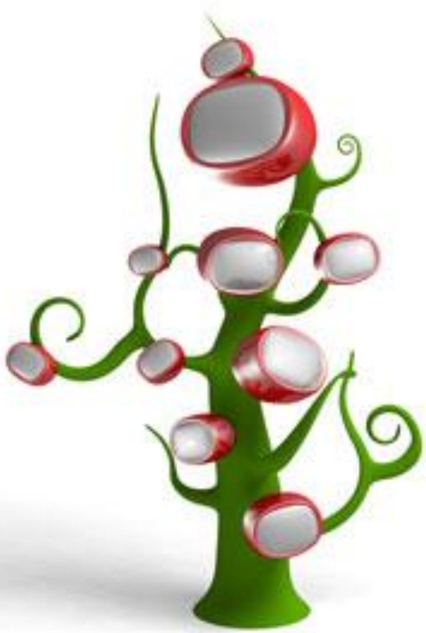
b) soit par l’élément le plus à gauche du sous-arbre droit;

afin de conserver la propriété d’ordre.;



❑ Application:

1. Ecrire une fonction récursive **supp_min(A)** qui supprime et retourne l’élément de plus faible valeur dans l’arbre A
2. En déduire la procédure récursive Supprimer (A, x) permettant de supprimer l’élément x de l’arbre A.



```
1. Fonction  Supp_min (var A: ABR): entier
   var      K: ^nœud
            Min: entier
   Début
    si  (A^.fg  =  nil)  alors
              Min ← A^.val
              K← A
              A ← A^.fd
              Libérer ( K)
               Retourner (Min)
     sinon
                Retourner (Supp_min (A^.fg))
     finsi

    Fin
```

```
2. Procédure Supprimer ( var A: ABR, x: entier )
 var     K: ^nœud
Début
   si (A ≠ nil) alors
          si ( A^.val= x) alors
             k ← A
             si ( A^.fg= nil) alors
               A ← A^.fd
               Libérér (K)
             sinon
               si ( A^.fd= nil) alors
                  A ← A^.fg
                  Libérér (K)
              sinon
             A^.val ← Supp-min ( A^.fd) // ou Supp-max(A^.fg)
             finsi
          finsi
```

```
  sinon
     si  ( x < A^.val ) alors
                    Supprimer (A^.fg, x)
     sinon
                    Supprimer (A^.fd, x)
     Finsi
  finsi
finsi

 Fin
```