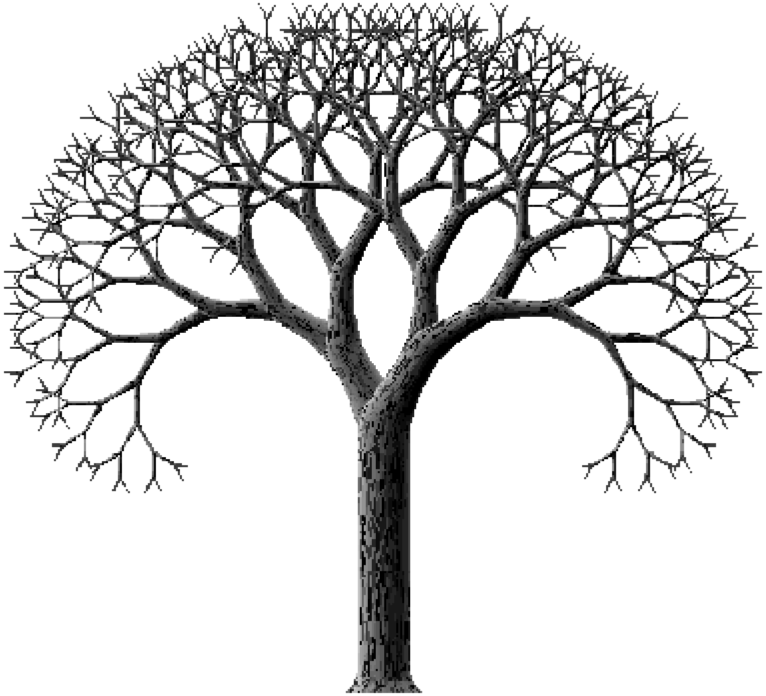
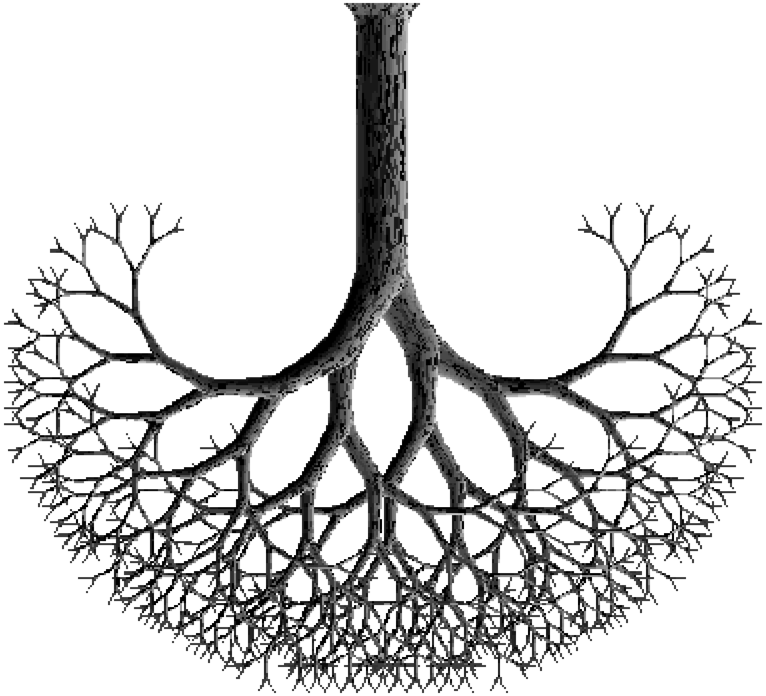


LES ARBRES

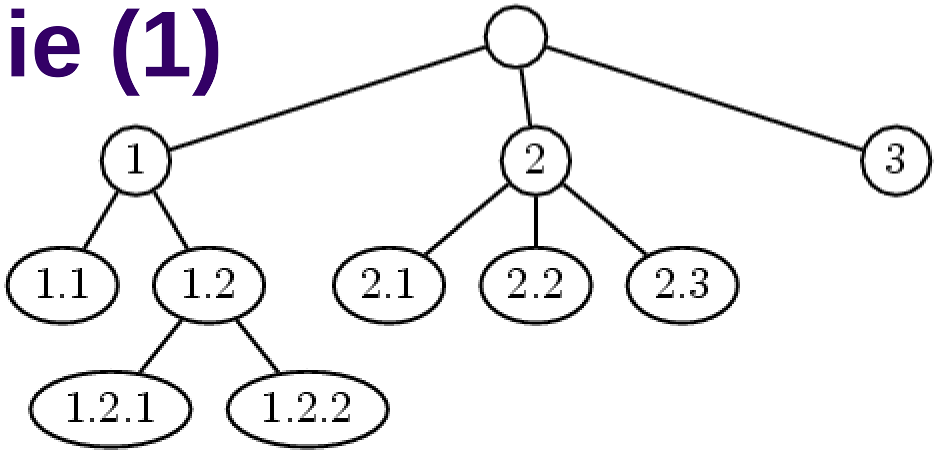


Définition

- Un arbre est un ensemble de nœuds organisés de façon hiérarchique à partir d'un nœud distingué : la racine

- C'est une structure fondamentale en informatique
  - répertoires des fichiers, compilation, expressions arithmétiques et logiques...
- Une propriété intrinsèque est la récursivité dans
  - les définitions,
  - la structure et
  - les algorithmes qui traitent les arbres

Terminologie (1)

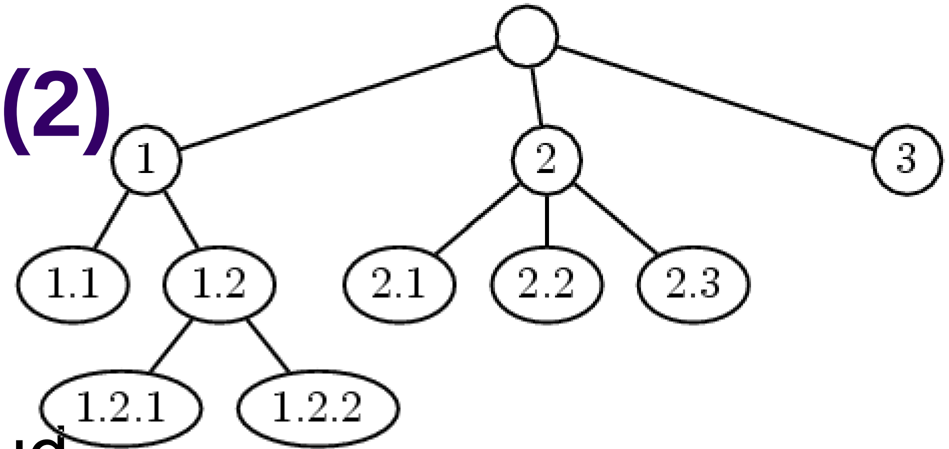


- Nœuds, racine, feuilles, arêtes
- Fils, père, frère, Sous-arbre, Branche, chemin

Retenons

- Un nœud n'a qu'un seul père
- Un nœud sans fils est une feuille ou un sommet pendant
- Une branche est le chemin qui lie un nœud à la racine

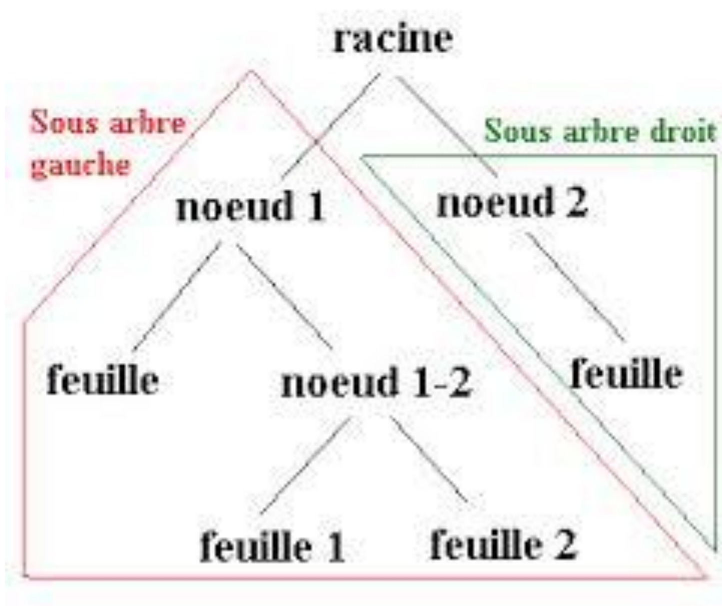
Terminologie (2)



- Niveau d’un nœud
  - nombre d’arêtes entre le nœud et la racine
- Hauteur d’un arbre
  - niveau maximum des feuilles de l’arbre
- Arbre ordonné (ordre entre père / fils)
- Degré d’un nœud
  - nombre de fils
- Arbre n-aire
  - les nœuds (hors feuilles) sont de degré n

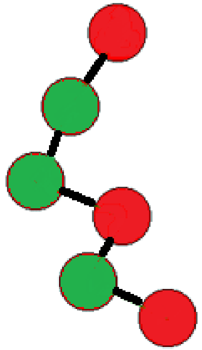
Arbre Binaire

- Un arbre binaire est soit vide ϕ, soit défini par :
  - Racine r ,
  - Sous-arbre gauche G, et
  - Sous-arbre droit D
- G et D sont eux-mêmes des arbres binaires disjoints

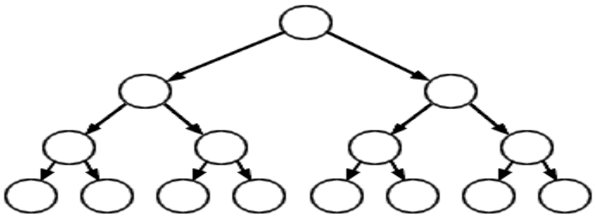


Arbre Binaire

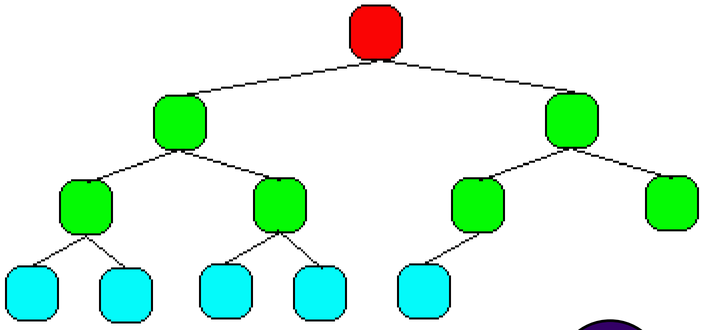
- Arbre filiforme ou dégénéré :
  - arbre binaire forme de nœuds qui n'ont qu'un seul fils



- Arbre complet :
  - arbre binaire dont chaque niveau est complètement rempli

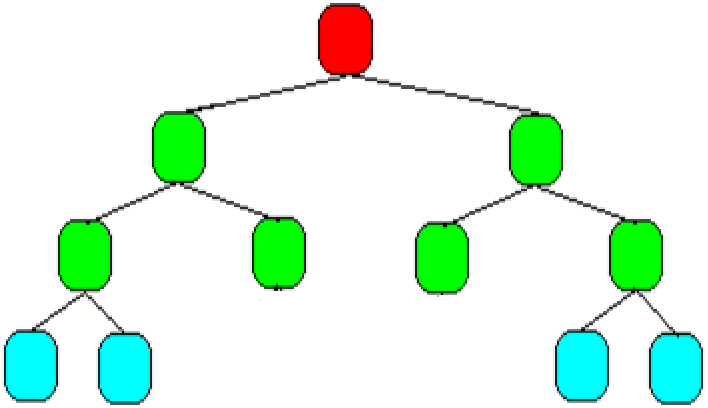


- Arbre parfait :
  - arbre binaire dont tous les niveaux sont complètement remplis sauf peut être le dernier, mais alors les feuilles du dernier niveau sont regroupées à gauche



Arbre Binaire

- Arbre localement complet :
  - arbre binaire non vide tel que tous les nœuds qui ne sont pas des feuilles ont 2 fils. Chaque nœud n'a pas de fils ou exactement 2 fils



- Peigne gauche :
  - arbre binaire localement complet tel que tout fils droit est une feuille

Arbre Binaire : Représentation physique (chaînée)

- Nœud = structure

  info : TypeInfo

  SAG : ↑ nœud

  SAD : ↑ nœud

  Fin structure

- Exemple : Arbre binaire d'entiers en C++
  - Struct nœud {

```
int info;
struct  nœud * SAG;
struct nœud * SAD;
} ;
```

Arbre Binaire : primitives (1)

- Création d'un arbre vide

```
Fonction créer_arbre () : ↑ nœud
   Début
      retourner Nil
   Fin Créer_arbre
```

- Test de vacuité d'un arbre

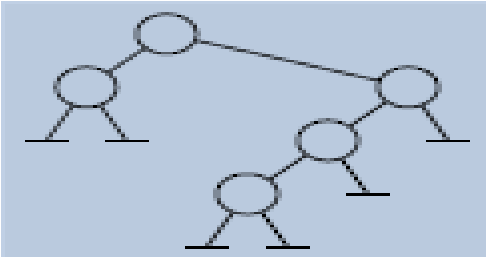
```
Fonction est_vide ( racine : ↑ nœud ) : logique
   Début
      retourner (racine = Nil)
   Fin est_vide
```

Arbre Binaire : primitives (2)

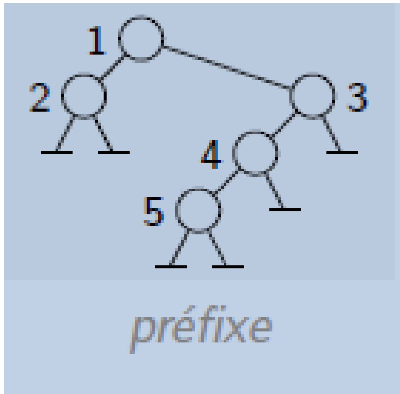
- Initialisation d'un arbre binaire

```
Fonction initialisation (val : typeInfo ) : ↑ nœud
   Var
       racine : ↑ nœud
   Début
     allouer(racine)
     racine ↑ .info ←val
     racine ↑.sag ← Nil
     racine : ↑.sad ← Nil
     retourner racine
   Fin initialisation
```

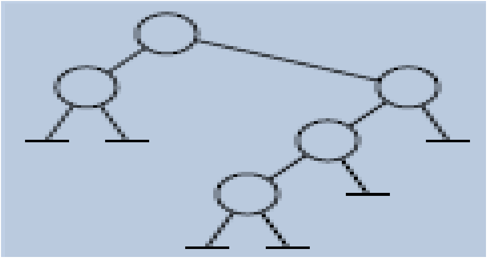
Arbre binaire : Parcours préfixe



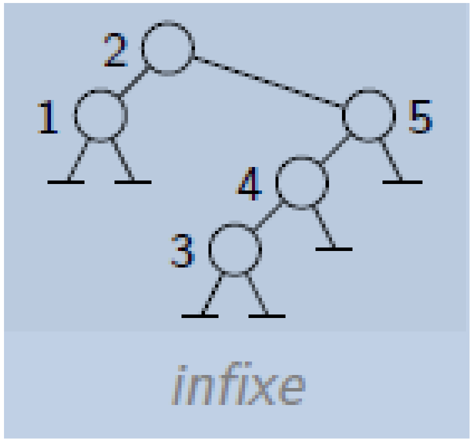
- Traiter la racine
- Parcours préfixe du sous-arbre gauche
- Parcours préfixe du sous-arbre droit



Arbre binaire : Parcours Infixe

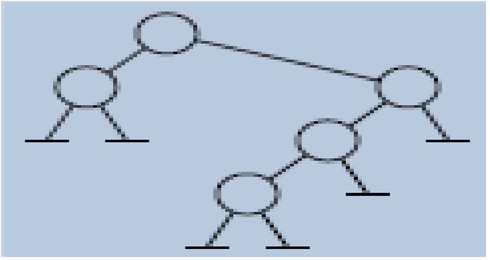


- Parcours infixe du sous-arbre gauche
- Traiter la racine
- Parcours infixe du sous-arbre droit

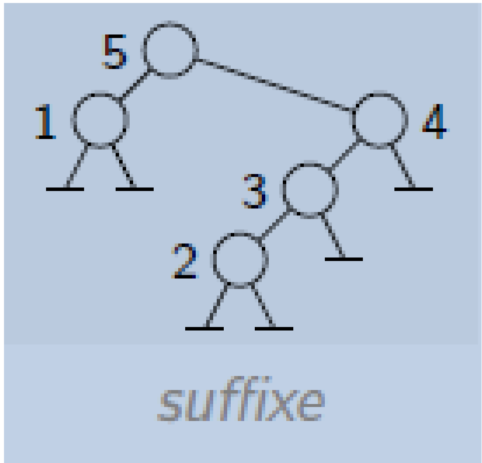


Arbre binaire : Parcours postfixé

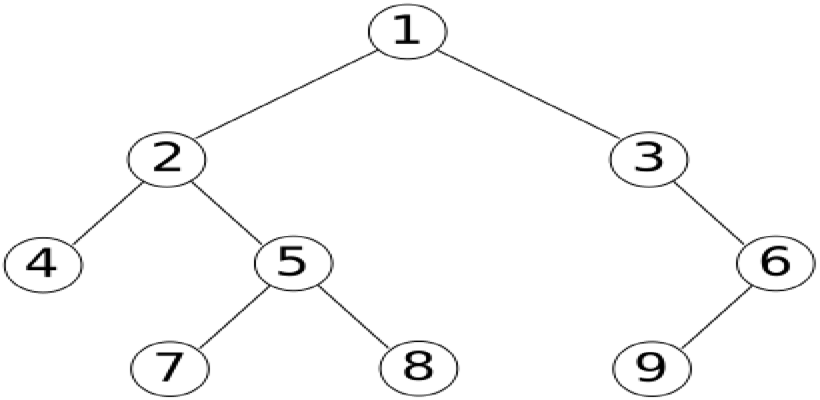
(suffixe)



- Parcours suffixe du sous-arbre gauche
- Parcours suffixe du sous-arbre droit
- Traiter la racine



Arbre binaire : Parcours (exemple)



- Rendu du parcours infixe :
  - 4, 2, 7, 5, 8, 1, 3, 9, 6
- Rendu du parcours postfixé :
  - 4, 7, 8, 5, 2, 9, 6, 3, 1
- Rendu du parcours préfixé :
  - 1, 2, 4, 5, 7, 8, 3, 6, 9

Arbre binaire : Opérations classiques

- Est_Feuille
  - Vérifie si un arbre est limité à une seule feuille

```
Fonction Est_feuille (racine : ↑nœud ) : logique
Début
   si est_vide(racine)
      alors retourner faux
      sinon retourner Est_vide(racine ↑.sag) et
                        Est_vide(racine ↑.sad)
  FinSi
Fin Est_feuille
```

Arbre binaire : Opérations classiques

- Nombre de nœuds dans arbre binaire

```
Fonction Nombre_Noeuds (racine : ↑nœud ) : entier
Début
   si est_vide(racine)
      alors retourner 0
      sinon retourner 1 + Nombre_Noeuds(racine ↑.sag)
            +  Nombre_Noeuds(racine ↑.sad)
  FinSi
Fin Nombre_Noeuds
```

Arbre binaire : Opérations classiques

- Hauteur d'un arbre binaire

```
Fonction Hauteur (racine : ↑nœud ) : entier
Début
   si est_vide(racine)
      alors retourner 0
      sinon retourner 1 + Max(Hauteur (racine ↑.sag),
                        Hauteur (racine ↑.sad))
  FinSi
Fin Hauteur
```

Arbre binaire : Opérations classiques

- Nombre de feuilles

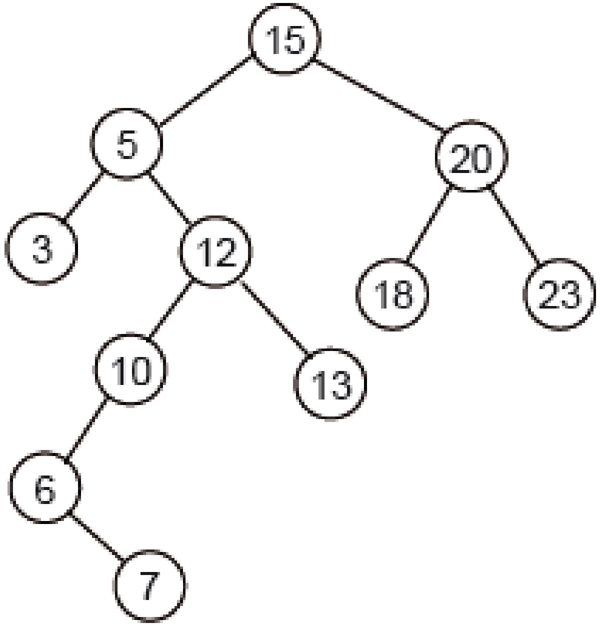
```
Fonction Nbre_feuilles(racine : ↑nœud ) : entier
Début
   si est_vide(racine)
     alors retourner 0
     sinon
      si est_feuille(racine )
         alors retourner 1
         sinon retourner Nbre_feuilles(racine ↑.sag)+
                        Nbre_feuilles (racine ↑.sad))
       FinSi
   FinSi
Fin Nbre_feuilles
```

Arbre binaire de Recherche

- Un arbre binaire de recherche (ABR) ; ou arbre binaire ordonné (ABO) est un arbre qui, s'il n'est pas vide, est tel que :
  - ses sous-arbres gauche et droit sont des ABR ;
  - les valeurs des nœuds du sous-arbre gauche sont strictement inférieures à la valeur du nœud -racine de l'arbre ;
  - les valeurs des nœuds du sous-arbre droit sont strictement supérieures à la valeur du nœud -racine de l'arbre.

Arbre binaire de Recherche

- Exemple



Arbre binaire de Recherche

- Recherche d'un élément (fonction récursive)

```
Fonction Chercher (racine : ↑Noeud , X : TypeInfo): logique
Début
Si ( racine = Nil) alors
   retourner 0
   Sinon
       Si( racine↑.info = X ) alors
             retourner 1
       sinon
       Si (X < racine↑.info ) alors
                   retourner chercher(racine↑.SAG, X)
                   sinon retourner chercher(racine↑.SAD, X)
       Finsi
   FinSi
FinSi
Fin chercher
```

Arbre binaire de Recherche

- Recherche d'un élément (fonction itérative)

```
Fonction Chercher (racine : ↑Noeud , X : TypeInfo): logique
Var
   n : ↑Noeud
Début
    n ← racine
    tantque (n <> Nil) et (n↑.info <> X) faire
        Si(X < n↑.info ) alors
                n ← racine↑.SAG
        sinon
                n ← racine↑.SAD
        FinSi
   Finfaire
  retourner( n <> nil)
Fin chercher
```

Arbre binaire de Recherche
Insertion d'un élément (itérative)

```
Procédure Insertion (Entrée X :
   TypeInfo, E/S racine : ↑Noeud )
Var
   n : ↑Noeud
Début
   si est_vide(racine)
        alors
        racine ← initialisation(X)
   Sinon
     // localisation du père
     tant que (racine <>nil ) faire
         père ← racine
         Si (X < racine↑.info )
        alors
            racine ← racine↑.SAG
        sinon
            racine ← racine↑.SAD
      finfaire
// création nouvel élément
   allouer(n)
   n↑.info ← X
   n↑.sag ← nil
   n↑.sad ← nil

// insertion de l'élément
Si (X < pere↑.info )  alors
        pere↑.SAG ←n
       sinon
        pere↑.SAD ←n
Fin Si

FinSi
Fin Insertion
```

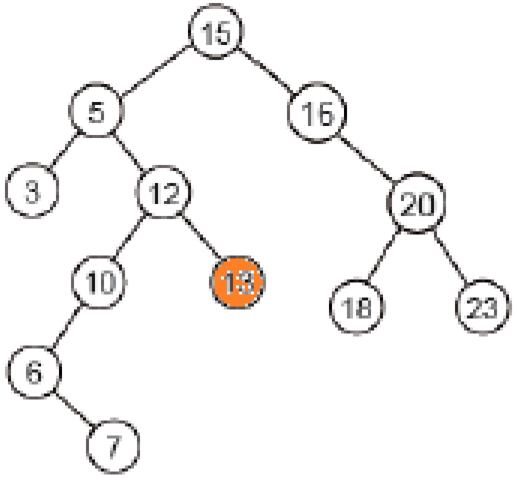
Arbre binaire de Recherche

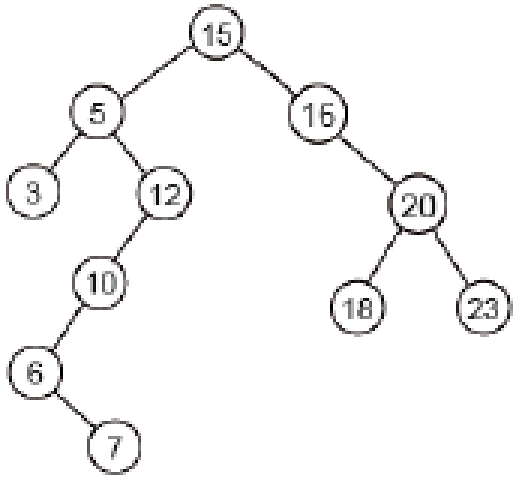
Insertion d’un élément (récursive)

```
Procédure Insertion (Entrée X : TypeInfo, E/S racine : ↑Noeud )
Début
   si est_vide(racine)
        alors
        racine ← initialisation(X)
   Sinon
        Si (X < racine↑.info ) alors
                    insertion(X,racine↑.SAG)
                    sinon insertion(X,racine↑.SAD)
        Finsi
   FinSi

Fin Insertion
```

Arbre binaire de Recherche
suppression d'un élément

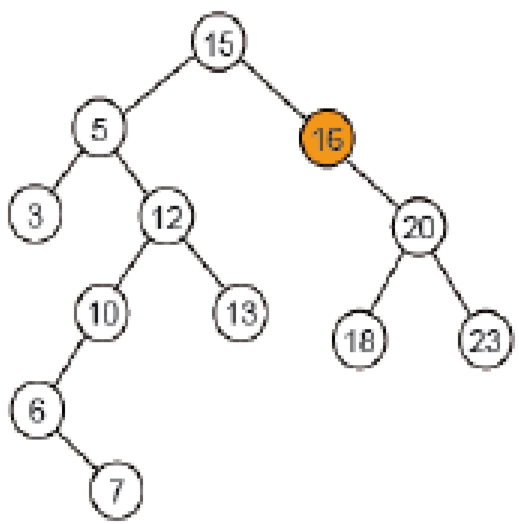
- 1^{er} Cas : l'élément à supprimer n'a pas de fils → il est terminal et il suffit de le supprimer

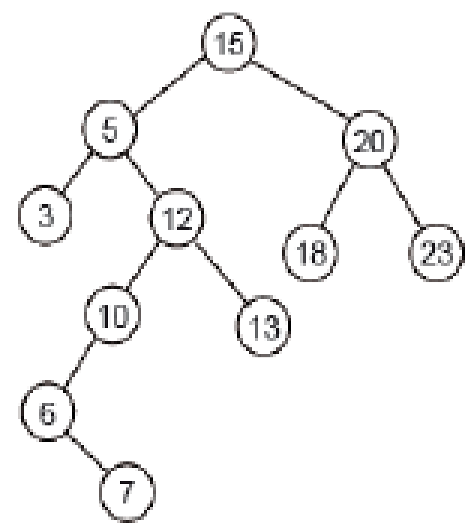




Arbre binaire de Recherche
suppression d’un élément

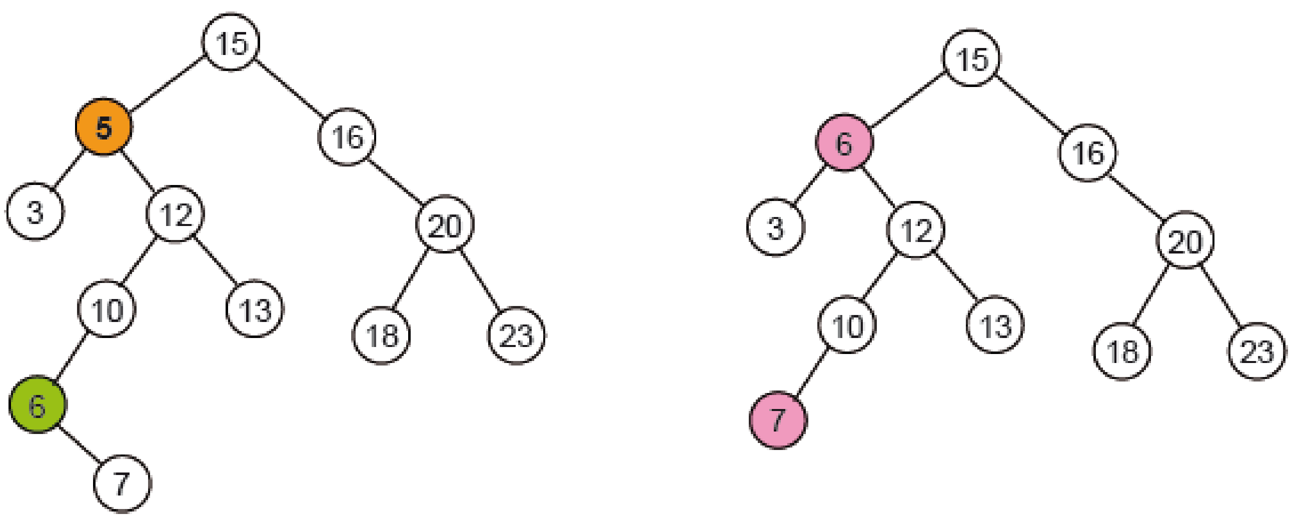
- 2ème Cas : l’élément a un fils unique
  → on supprime le nœud et on relie son fils à son père





Arbre binaire de Recherche
suppression d'un élément

- 3ème Cas : l'élément à supprimer a deux fils → on le remplace par son successeur qui est toujours le minimum de ses descendants droits.



Arbre binaire de Recherche
Suppression d’un élément (Récursive)

```
Procédure Suppression(Entrée X : TypeInfo, E/S racine : ↑Noeud )
Var
   n : ↑Noeud
Début
   si Non (est_vide(racine))
         alors
          Si (x > racine ↑.info )
             alors Suppression(x,racine ↑.SAD)
             sinon
                   Si (x < racine ↑.info )
                            alors Suppression(x,racine ↑.SAG)
                            sinon
                         // x trouvé
                            n ← racine
                            si (n ↑.SAG = nil) // 0 ou un seul fils
                              alors racine ← n ↑.SAD)
                              sinon
                                si (n ↑.SAD = nil) // 0 ou un seul fils
                                      alors racine ← n ↑.SAG)
                                      sinon  // deux fils
                                        SUPP ( n)
                                FinSi
                         Fin Si
                   FinSi
         FinSi
    Fin Si
Fin Suppression
```

Arbre binaire de Recherche

Suppression d’un élément (Récursive) suite

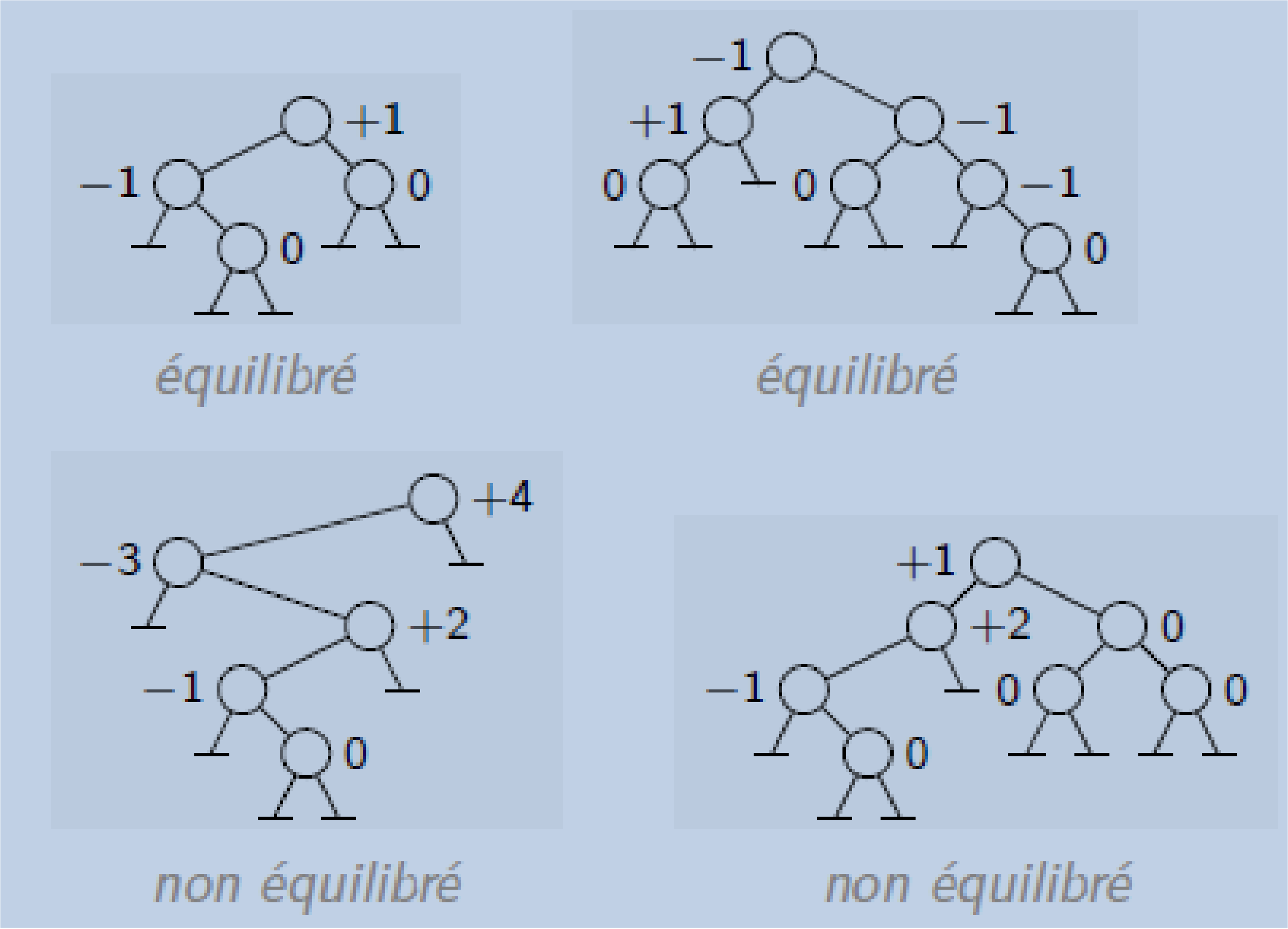
```
Procédure SUPP (E/S n : ↑Nœud  )
Var
   r,p : ↑Nœud
Début
    Min (n,r,p) // retourne l’@ r du plus petit droit et p son père
    n ↑.info ← r ↑.info
    p ↑.SAG ← r ↑.SAD
    liberer(r)

Fin SUPP




Procédure Min (Entrée n : ↑Nœud , Sortie r,p : ↑Nœud  )

Début
    p ← n ↑SAD
    r ← p ↑ . SAG
    tant que r ↑ .SAG  <> Nil faire
        p ← r
        r ← r ↑ . SAG
    Fin faire

Fin Min
```

Arbre binaire Equilibré : Définition

- L’équilibre d’un arbre binaire est un entier qui
  - vaut 0 si l’arbre est vide et
  - la différence des hauteurs des sous-arbres gauche et droit de l’arbre sinon.

- Un arbre binaire est équilibré lorsque l’équilibre de chacun de ses sous-arbres non vides n’excède pas 1 en valeur absolue.

Arbre binaire Equilibré : Exemple



Arbre binaire Equilibré : Exemple

```
Fonction Est_Equilibre(racine : ↑ nœud ) : logique
Var

Debut
  Si racine = nil

     alors retourner vrai

     sinon
       retourner (abs(hauteur (racine ↑.SAG) - hauteur (racine ↑.SAD) <= 1 ET
                 Est_equilibre(racine ↑.SAG) ET Est_equilibre(racine ↑.SAD)
Fin Est_Equilibre
```