Algorithmique avancée

Arbres binaires de recherche

Frédéric Guyomarch

Université de Lille1
IUT-A de Lille

2016/2017 - Semestre 3

Introduction

Les arbres

On a vu des structures de base linéaires :

- Des tableaux
- Des listes chaînées

Les arbres permettent de *hiérarchiser* l'information.

Définition

Un arbre est un graphe non orienté, acyclique et connexe.

Les arbres

Exemple

- Organigramme
- Système de fichiers
- Expressions arithmétiques
- Tableau d'élimination direct en sport

Terminologie

- La terminologie est empruntée aux arbres
  - généalogiques : père, fils, descendant...
  - naturels : feuille, branche, racine...

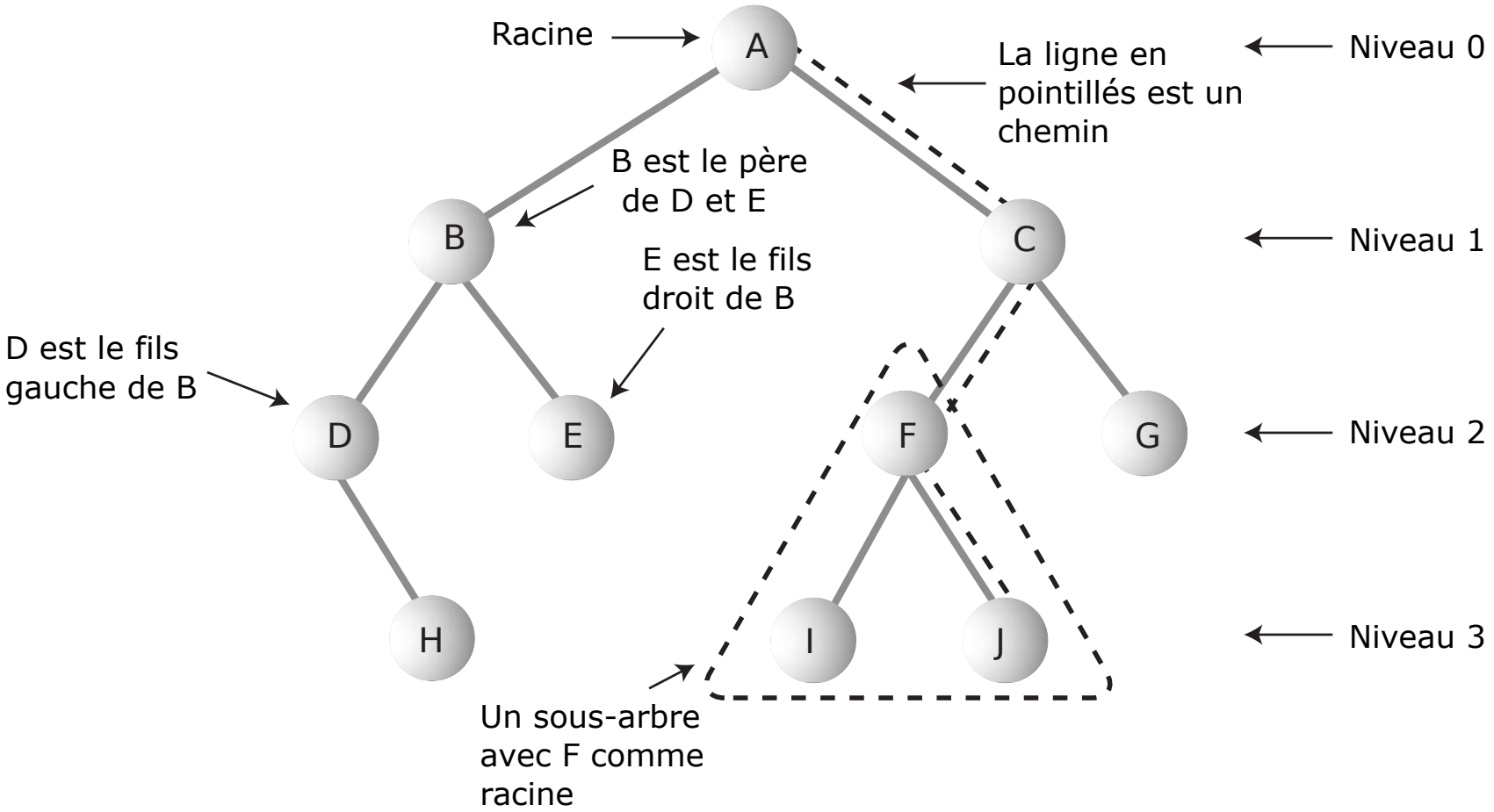
Un arbre est composé de nœuds reliés entre eux par des arêtes.

- La racine est l'unique nœud au sommet de l'arbre.
- Chaque nœud (hormis la racine) est le fils de son prédécesseur dans l'arbre qui est son père.
- Les nœuds qui n'ont pas de fils sont des feuilles.

Terminologie

- On dit qu'il y a un chemin entre deux nœuds si il est possible d'accéder de l'un à l'autre en passant par les arrêtes.
- Le niveau d'un nœud est le nombre de génération(s) qui le sépare de la racine.
- La hauteur est le nombre maximum de niveaux.
- Le degré ou l'arité d'un nœud est le nombre de ses fils.
- Un arbre d'arité n est un arbre dont le degré maximum de ses nœuds est n.

Terminologie

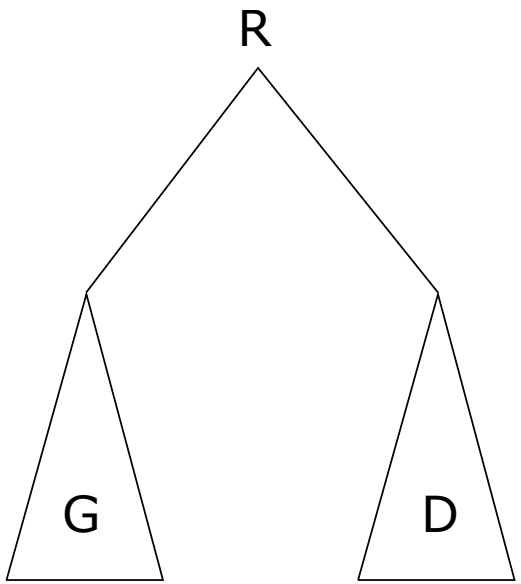


H, E, I, J et G sont des feuilles.

Récursivité

L'arbre est une structure récursive qui peut être vue :

- Soit comme vide
- Soit comme un arbre dont les fils sont des sous-arbres.



Arbre binaire

Définitions

Un arbre d'arité 2 est un arbre binaire. Il a au maximum deux fils, un fils gauche et un fils droit.
Un arbre binaire est dit pur si chacun des nœuds a soit exactement 2 fils, soit aucun.

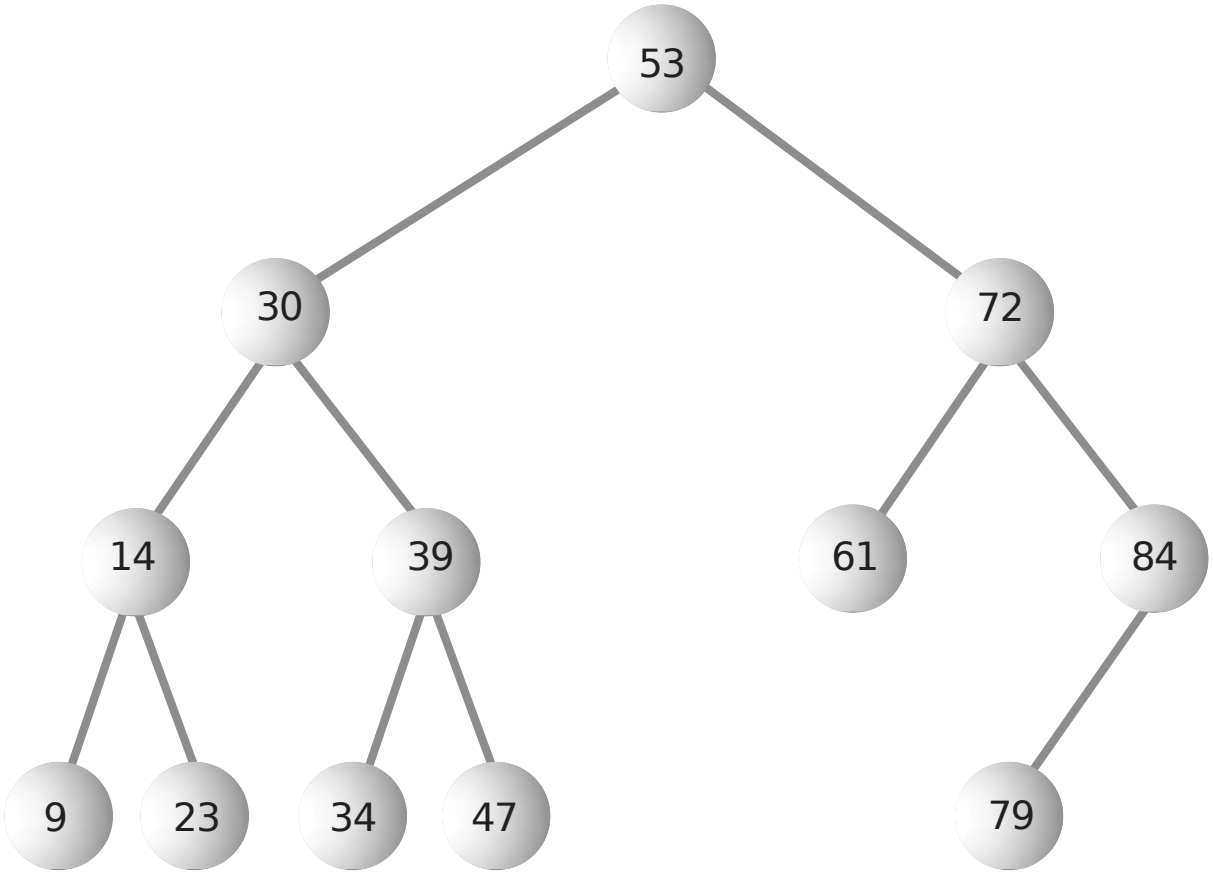
ABR

Un arbre binaire de recherche (ABR) est un type de données abstrait.

- Chaque élément a une clé unique, i.e une clé identifie un élément de façon unique.
- Les clés dans un sous arbre gauche non vide doivent être inférieures à la clé de la racine de ce sous arbre.
- Les clés dans un sous arbre droit non vide doivent être supérieures à la clé de la racine de ce sous arbre.
- Les sous arbres gauche et droit sont aussi des arbres binaires de recherche. (structure récursive).

ABR

Exemple



Un arbre binaire de recherche

ABR

On peut définir les opérations suivantes pour un ABR :

- `estVide()` : retourne vrai si l'arbre A est vide
- `estFeuille(K cle)` : retourne vrai si l'élément de clé *cle* est une feuille
- `estRacine(K cle)` : retourne vrai si l'élément de clé *cle* est la racine
- `recherche(K cle)` : retourne l'élément de clé *cle*
- `insertion(K cle, V val)` : insertion d'un élément de clé *cle* et de valeur *val*.
- `suppression(K cle)` : suppression d'un élément de clé *cle*.

ABR

Comparaisons

Quid de l'efficacité d'un ABR par rapport aux tables de hachage, aux tableaux ordonnés ou encore aux listes ?

ABR

Propriétés

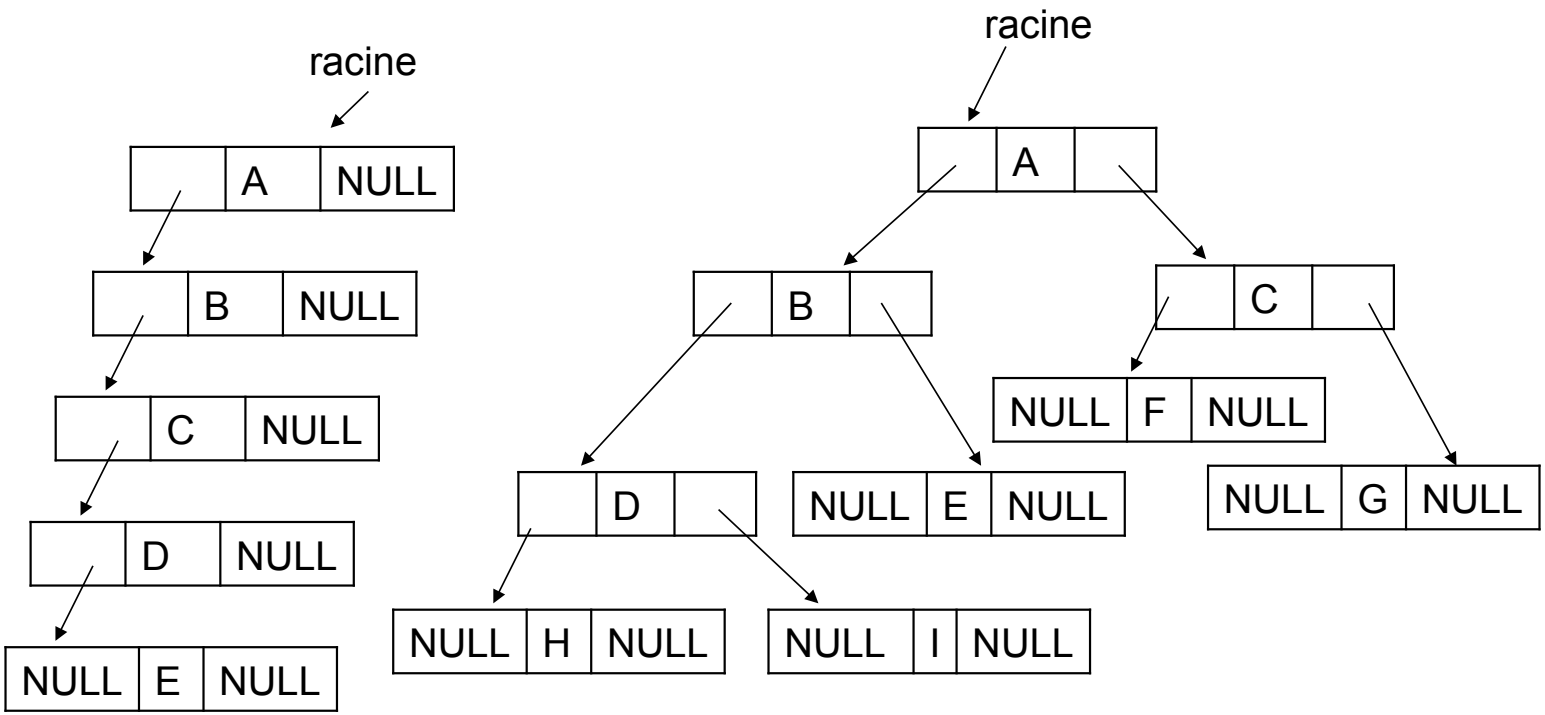
- Nombre maximal de nœuds :
  - Le nombre maximal de nœuds de niveau i dans un arbre binaire est $2^{i-1}, i >= 1$
  - Le nombre maximal de nœuds dans un arbre binaire de profondeur k est $2^k - 1, k >= 1$
  - Arbre binaire complet de profondeur k est un arbre binaire ayant $2^k - 1$ nœuds.
    - Tous les nœuds possèdent 0 ou 2 fils.
    - Le nombre maximal de nœuds est atteint donc toutes les feuilles sont au niveau k

Question : Dans un ABR complet de 20 nœuds, où l'on considère que la racine est au niveau 1, combien il y a de nœuds au niveau 5 ?

Représentation d'un arbre

Par chaînage

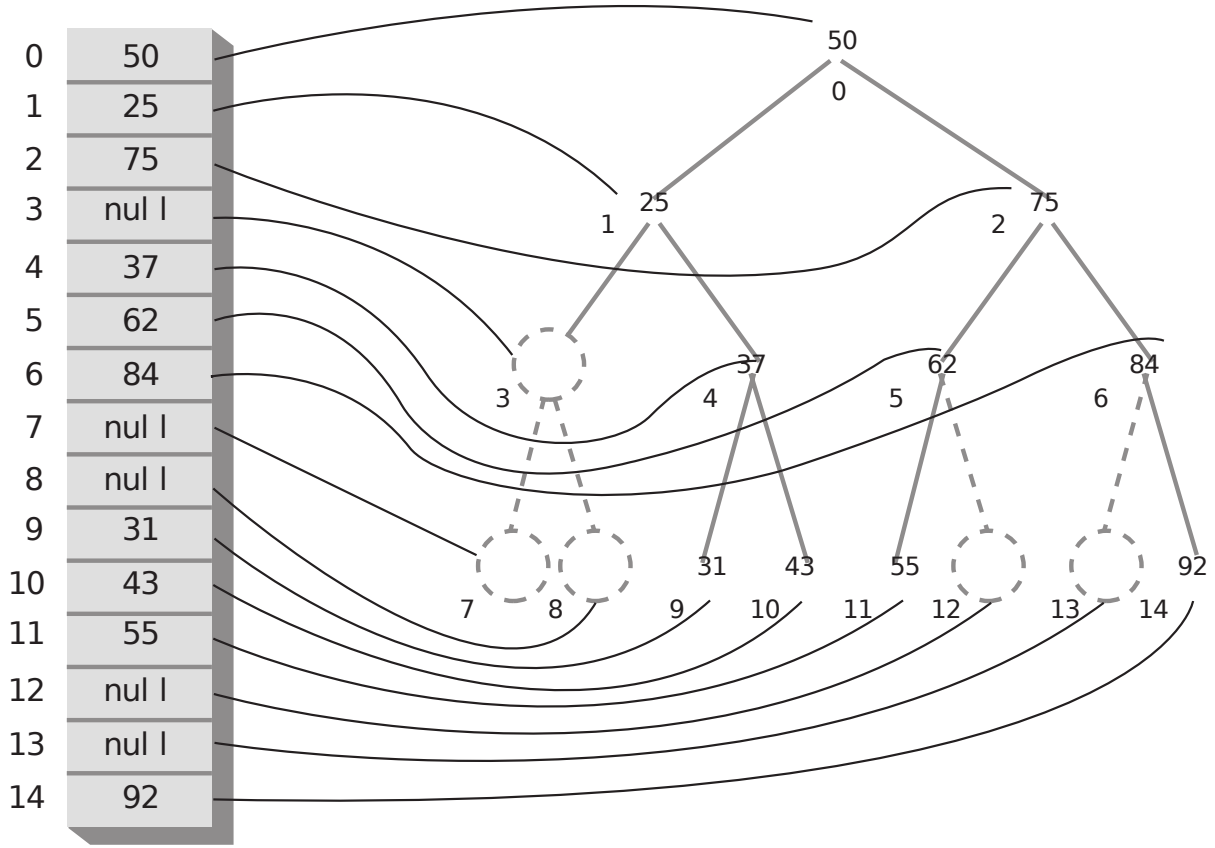
A la manière des listes, on peut utiliser une représentation chaînée.



Représentation

Par tableau

Il existe également une représentation par tableau :



Représentation

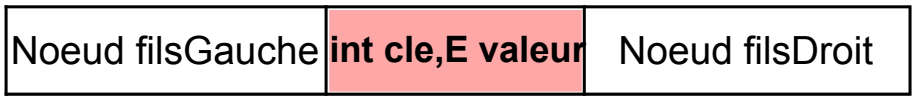
Par tableau

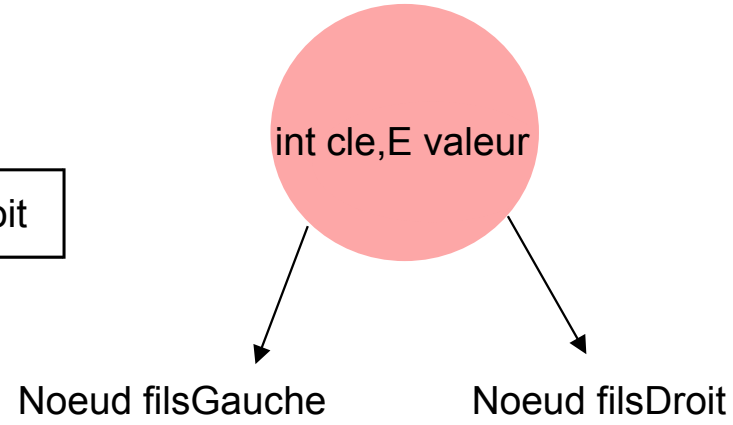
- Gaspillage probable de mémoire
- Si un nœud a pour indice `idx` :
  - Son fils droit a pour indice $2 * idx + 2$
  - Son fils gauche a pour indice $2 * idx + 1$
  - Son père a pour indice $(idx - 1)/2$

Nous préférerons le plus souvent la représentation chaînée.

Représentation d'un arbre

Par chaînage






```
class Arbre<E>{
  Noeud racine;
}

class Noeud{
  int cle;
  E valeur;
  Noeud filsGauche;
  Noeud filsDroit;
}
```

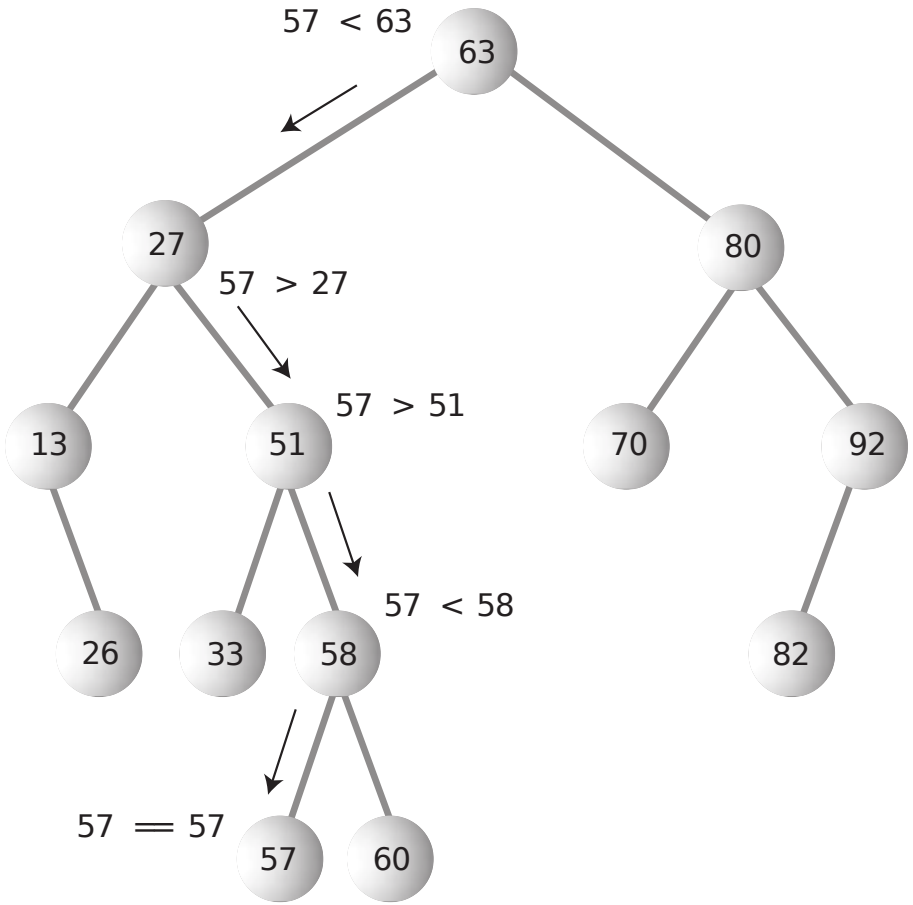
ABR

Recherche

1. Je compare la clé du nœud courant avec la clé recherchée
2. Si elle n'est pas égale alors :
   - Elle est supérieure à la clé recherchée, je regarde le fils gauche
   - Elle est inférieure à la clé recherchée, je regarde le fils droit
3. Tant que le nœud courant n'est pas *null* je réitère les étapes ci-dessus.

ABR

Recherche



Recherche de la valeur 57

ABR

Recherche itérative

```
public Noeud recherche(int k){
 Noeud courant = racine;

 while(courant.cle != k){
   if(k < courant.cle)
    courant = courant.filsGauche;
   else
    courant = courant.filsDroit;
   if(courant == null)
    return null;
 }
  return courant;
}
```

ABR

Recherche récursive

```
public Noeud recherche(Noeud noeud, int k){

  if(noeud == null)
   return null;

  if(noeud.cle == k)
   return noeud;

  if(noeud.cle <= k)
   return recheche(noeud.filsDroit, k);

   return recherche(noeud.filsGauche, k);
}
```

ABR

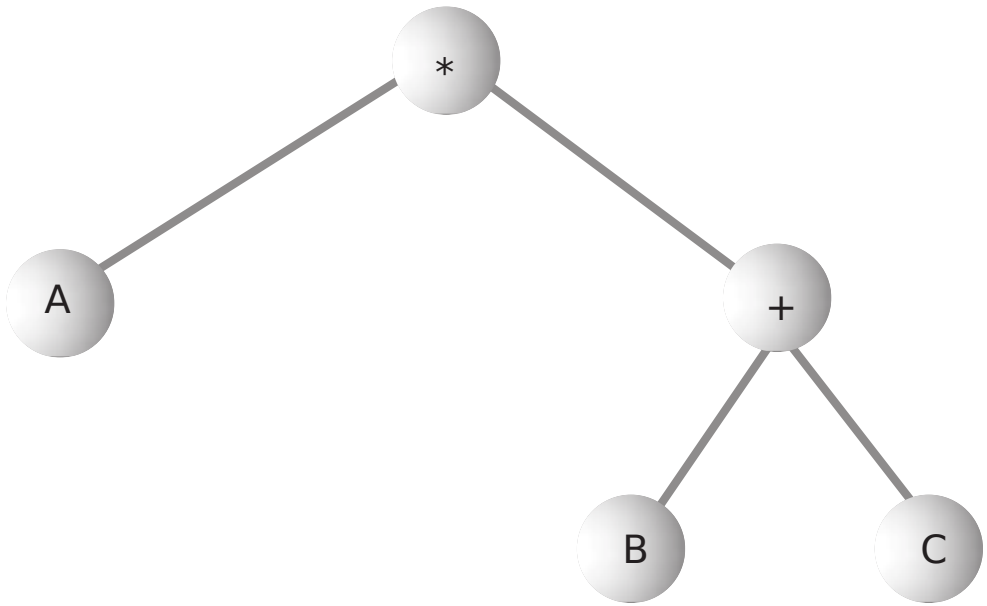
Parcours

- On veut visiter une seule fois chacun des nœuds de l'arbre
- On adopte une stratégie de parcours pour parcourir tout l'arbre
- Soit G,D les parcours des sous-arbres gauche et droit et V la visite le nœud courant (pour obtenir sa valeur par exemple). On peut dégager plusieurs stratégies de parcours, on retiendra :
  - GVD : parcours infixé
  - VGD : parcours préfixé
  - GDV : parcours postfixé

Le sens du parcours va dépendre de l'ordre des appels !

ABR

Parcours



Infixé : A*(B+C)
Préfixé : *A+BC
Postfixé: ABC+*

ABR

Parcours infixe

1. Appel récursif sur le fils gauche
2. Visite du nœud courant
3. Appel récursif sur le droit gauche

```
public void parcours(Noeud racine){

   parcours(racine.filsGauche);
   System.out.print(racine.valeur);
   parcours(racine.filsGauche);

}
```

ABR

Parcours préfixé

1. Appel récursif sur le fils gauche
2. Appel récursif sur le droit gauche
3. Visite du nœud courant

```
public void parcours(Noeud racine){

  parcours(racine.filsGauche);
  parcours(racine.filsDroit);
  System.out.print(racine.valeur);

}
```

ABR

Parcours postfixé

1. Visite du nœud courant
2. Appel récursif sur le fils gauche
3. Appel récursif sur le droit gauche

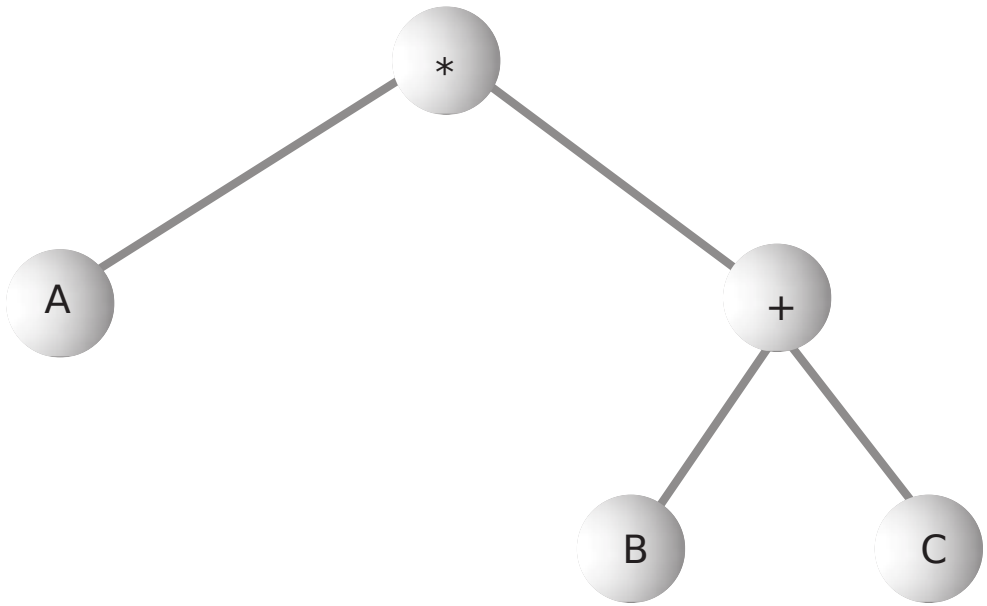
```
public void parcours(Noeud racine){

  System.out.print(racine.valeur);
  parcours(racine.filsGauche);
  parcours(racine.filsDroit);

}
```

ABR

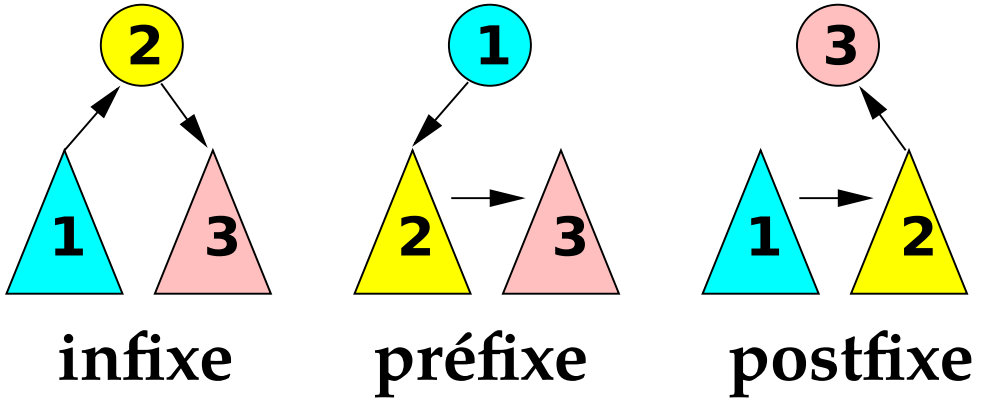
Parcours

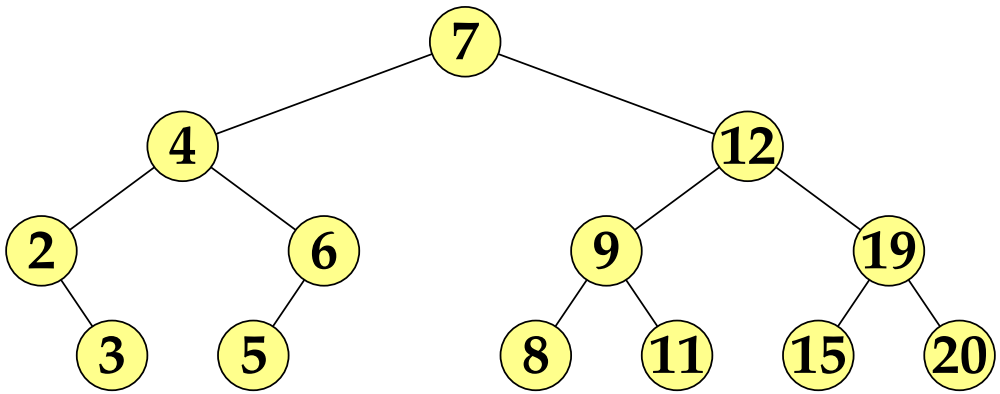


Infixé : A*(B+C)
Préfixé : *A+BC
Postfixé: ABC+*

ABR

Parcours

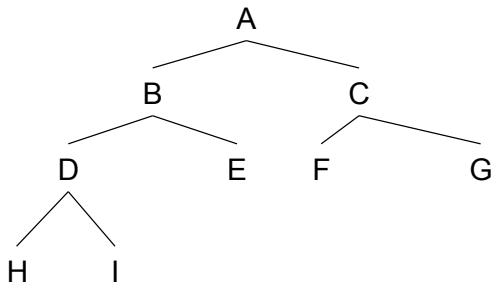




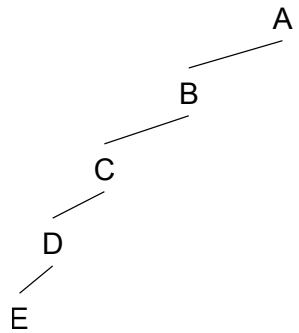
- Ordre infixe : 2, 3, 4, 5, 6, 7, 8 , 9, 11, 12, 15 ,19, 20.
- Ordre préfixe : 7, 4, 2, 3, 6, 5, 12, 9, 8, 11, 19, 15, 20.
- Ordre postfixe : 3, 2, 5, 6, 4, 8, 11, 9, 15, 20, 19, 12, 7.

Parcours

Efficacité

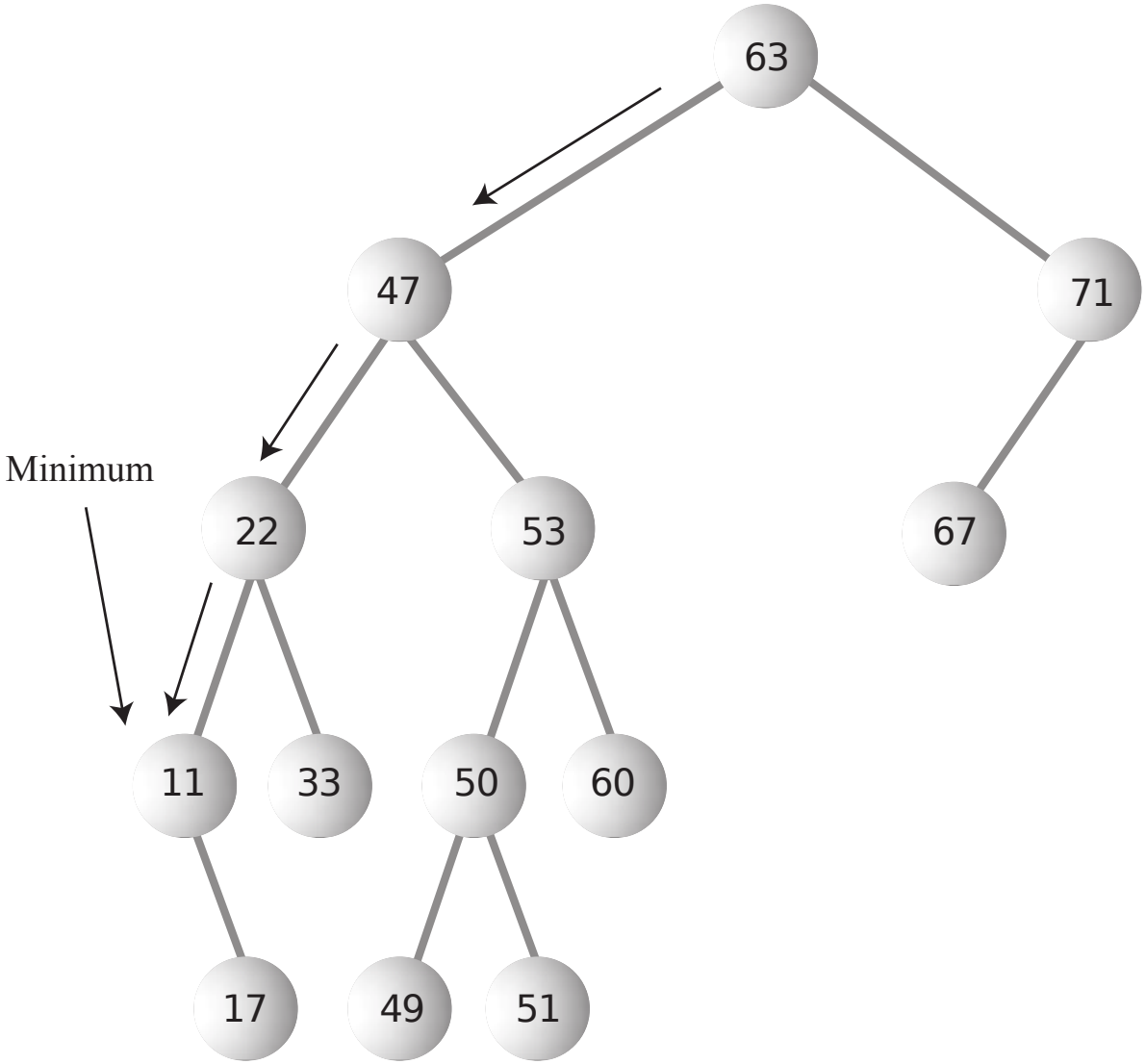


Arbre équilibré, nombre de niveaux minimum donc $log(n)$ étapes.



Arbre déséquilibré, n étapes.

Min/Max



Min/Max

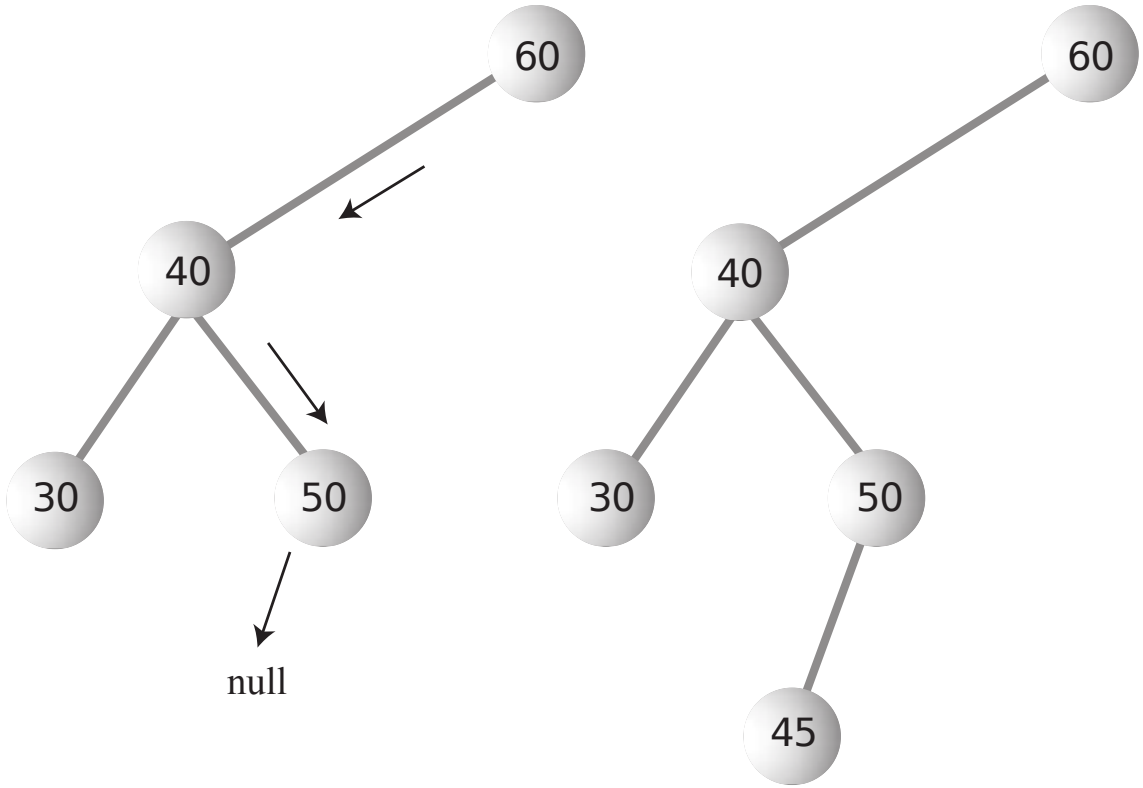
Code itératif

```
public Noeud maximum(){
 Noeud dernier, courant;

 courant = racine;

 while(courant!=null){
    dernier = courant;
    courant = courant.fildDroit;
    //courant = courant.filsGauche pour le min
 }
 return dernier;
}
```

Insertion



a)Avant insertion b)Après insertion de 45

Insertion

Version itérative

```
public void insertion(int cle,E valeur){
  Noeud newNode = new Noeud(cle,valeur);

  if(racine==null) // pas de noeud a la racine
    racine = newNode;
  else{
    Noeud courant = racine; // commence a la racine
    Noeud parent;
    boolean estAjoute = false;
```

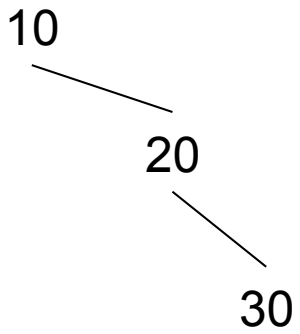
Insertion

Version itérative

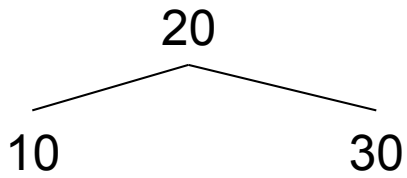
```
//Suite du code du slide precedent
 while(!estAjoute){
     parent = courant;
     if(cle < courant.cle){ // a gauche?
       courant = courant.filsGauche;
       if(courant == null){ // insertion a gauche
       parent.filsGauche = newNode;
       estAjoute = true;
      }
     }
     else { // a droite?
       courant = courant.filsDroit;
       if(courant == null){ // insertion a droite
       parent.filsDroit = newNode;
       estAjoute = true;
      }}}}}
```

Insertion

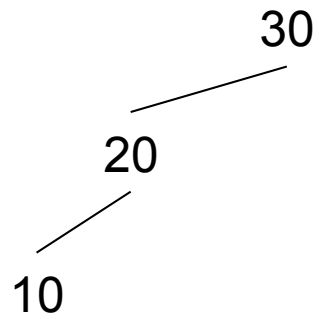
La structure de l'arbre diffère en fonction de l'ordre des insertions.



Ordre : 10, 20, 30



Ordre : 20, 10, 30

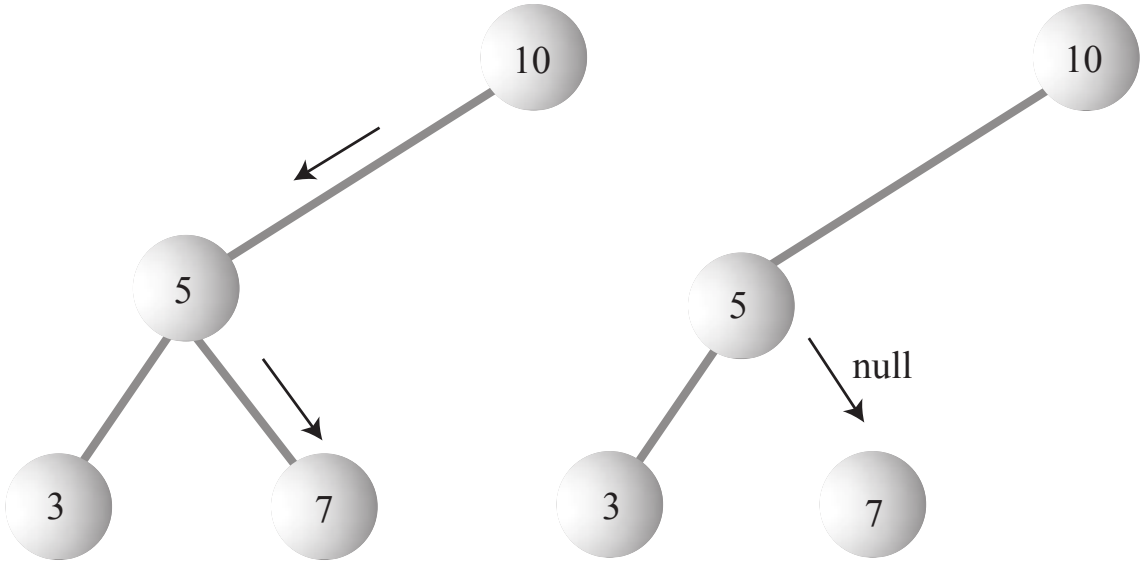


Ordre : 30, 20, 10

Suppression

Cas 1

Le nœud à supprimer est une feuille.



a)Avant suppression b)Après suppression de 7

Suppression

Cas 1 : Version itérative

```
public boolean suppression(int cle){
  Noeud courant = racine;
  Noeud parent = racine;
  boolean estFilsGauche = true;
  while(courant.cle != cle){
   parent = courant;
     if(cle < courant.cle){
      estFilsGauche = true;
      courant = courant.filsGauche;
   }else{
     estFilsGauche = false;
     courant = courant.filsDroit;
   }
   if(courant == null)
     return false;
  } // fin while, courant pointe le noeud a effacer
```

Suppression

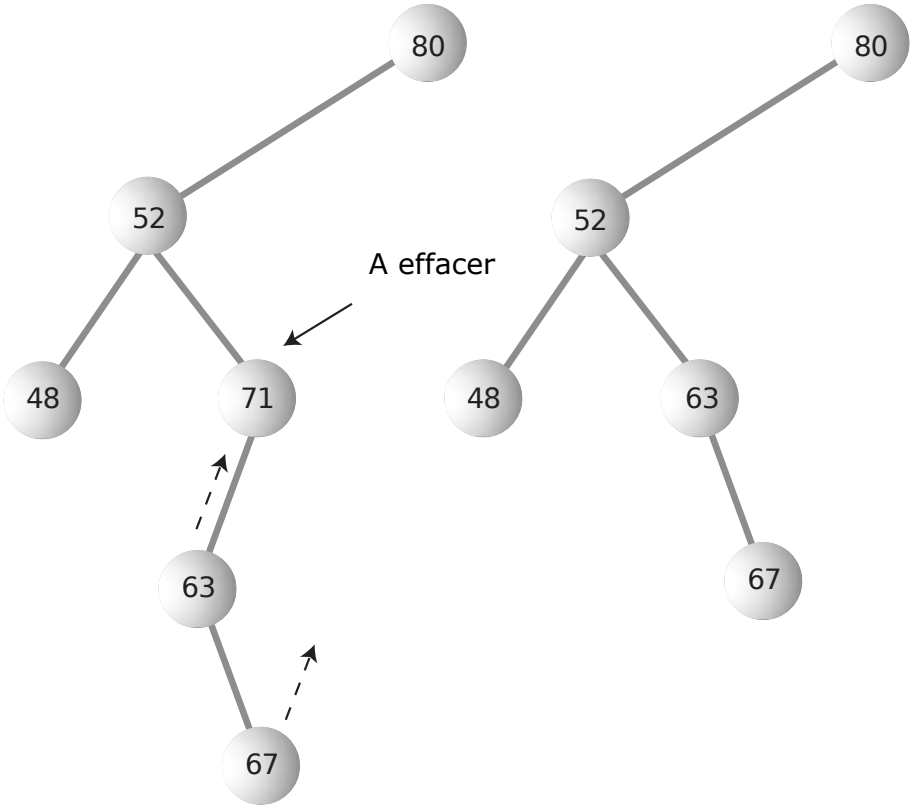
Cas 1 : Version itérative

```
// suite de la suppression cas 1
   if(courant.filsGauche==null &&
    courant.filsDroit==null){
     if(courant == racine)
      racine = null;
     else if(estFilsGauche)
      parent.filsGauche = null;
     else
      parent.filsDroit = null;
}
```

Suppression

Cas 2

Le nœud à supprimer a un unique fils.



a)Avant suppression b)Après suppression de 71

Suppression

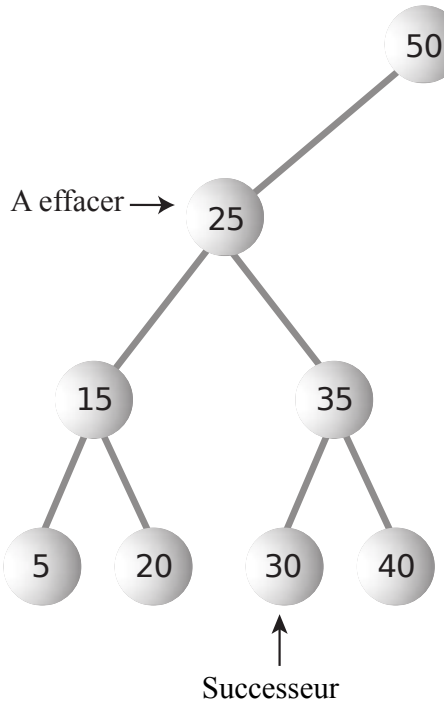
Cas 2 : Version itérative

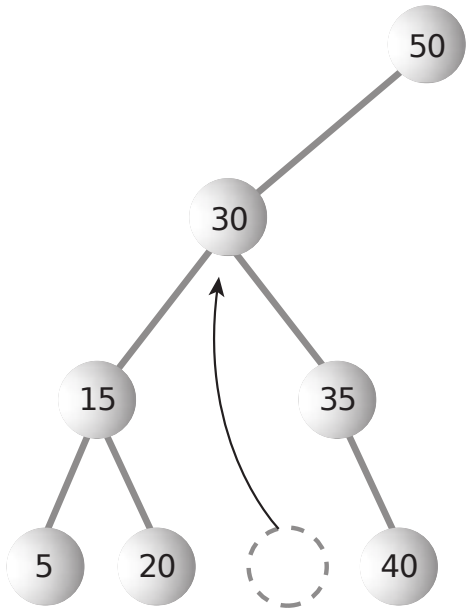
```
//Suite de traitement du 1er cas
//pas de fils droit on remplace avec le ss-arbre G
else if(courant.filsDroit==null)
   if(courant == racine)
    racine = courant.filsGauche;
   else if(estFilsGauche)
    parent.filsGauche = courant.filsGauche;
   else parent.filsDroit = courant.filsGauche;
//pas de fils gauche on remplace avec le ss-arbre D
else if(courant.filsGauche==null)
   if(courant == racine)
    racine = courant.filsDroit;
   else if(estFilsGauche)
    parent.filsGauche = courant.filsDroit;
   else parent.filsDroit = courant.filsDroit;
```

Suppression

Cas 3

Le nœud à supprimer a deux fils.





a)Avant suppression b)Après suppression de 25

Suppression

Cas 3

Recherche du successeur :

- On a besoin d'identifier le successeur du nœud à supprimer
- Si le nœud à supprimer a un fils droit
  - Il s'agit du nœud avec la clé minimum des successeurs
  - Il s'agit donc du nœud de clé minimum du sous-arbre droit du nœud à supprimer
- Sinon c'est le premier père droit en remontant la branche du nœud à supprimer.
- Respectivement, le prédécesseur : nœud de clé maximum dans le sous-arbre gauche du nœud à supprimer sinon le premier père gauche si pas de fils droit

Suppression

Recherche du successeur(si fils doit non null)

```
public Noeud successeur(Noeud noeud){
  return minimum(noeud.filsDroit);
}

public Noeud predecesseur(Noeud noeud){
  return maximum(noeud.filsGauche);
}
```

Si on veut mettre le successeur à la place d'un nœud à supprimer :

- il faut "détacher" le successeur
- et mettre à jour les références

Suppression

Recherche du successeur(si fils doit non null)

```
Noeud getSuccesseur(Noeud supNoeud){
  Noeud parentSuccesseur = supNoeud;
  Noeud successeur = supNoeud;
  Noeud courant = supNoeud.filsDroit;
   while(courant != null){
    parentSuccesseur = successeur;
    successeur = courant;
    courant = courant.filsGauche;
   }
   //Si succ. = fils droit de supNoeud rien a faire
   if(successeur != supNoeud.filsDroit){
    //MAJ des references
    parentSuccesseur.filsGauche = successeur.filsDroit;
    successeur.filsDroit = supNoeud.filsDroit;
   }
  return successeur;
}
```

Suppression

Cas 3 : Version itérative

```
//Suite de traitement du 1er et 2eme cas
// deux fils, on remplace par le successeur
else {
  //on recupere le successeur du noeud courant
  //mais on doit aussi le detacher
  Noeud successeur = getSuccesseur(courant);
  //connecte le pere du noeud courant au successeur
  if(courant == racine)
   racine = successeur;
  else if(estFilsGauche)
   parent.filsGauche = successeur;
  else parent.filsDroit = successeur;
  //connecte le successeur au fils gauche du courant
  successeur.filsGauche = courant.filsGauche;
 }return true; //Fin de la suppression du cas 3
}//(un successeur ne peut pas avoir de fils gauche)
```

Implémentation Java

- Pas d'implémentation directement manipulable d'ABR en Java
- Mais utilisés pour l'implémentation de Map (TreeMap) ou de Set (TreeSet) (à l'instar des tables de hachages).