Algo 2 – séance 6
Arbres binaires de recherche (ABR (suite) : suppression et arbres équilibrés

---

Franck Hetroy et Denis Trystram

mars 2017

Grenoble INP ENSIMAG
ECOLE NATIONALE SUPERIEURE D'INFORMATIQUE ET DE MATHEMATIQUES APPLIQUEES

Retour sur la séance 5

```
procedure InsertABR(E : in Value ; A : in out Binary_Tree) is
begin
   if A = Empty_Tree then
      A := new Node'(Val => E, Children => (Empty_Tree, Empty_Tree)) ;
   else
      if E <= A.Val then (1) else (2) end if ;
   end if ;
end Insert ;
```

Corrigé

```
procedure InsertABR(E : in Value ; A : in out Binary_Tree) is
begin
   if A = Empty_Tree then
      A := new Node'(Val => E, Children => (Empty_Tree, Empty_Tree)) ;
   else
      if E <= A.Val then
      (1) InsertABR(E, A.Children(Left))
      else
      (2) InsertABR(E, A.Children(Right))
      end if ;
   end if ;
end Insert ;
```

Aujourd'hui

- Arbres binaires de recherche : suite et fin
- Arbres binaires de recherche équilibrés (AVL)

Opérations sur les ABR

- **Recherche**
  - Vu en cours 5
- **Insertion**
  - Vu en TD 5
- **Suppression**
  - Principe : Maintenant
  - Code en TD

Opérations sur les ABR

- **Recherche**
  - Vu en cours 5
- **Insertion**
  - Vu en TD 5
- **Suppression**
  - Principe : Maintenant
  - Code en TD

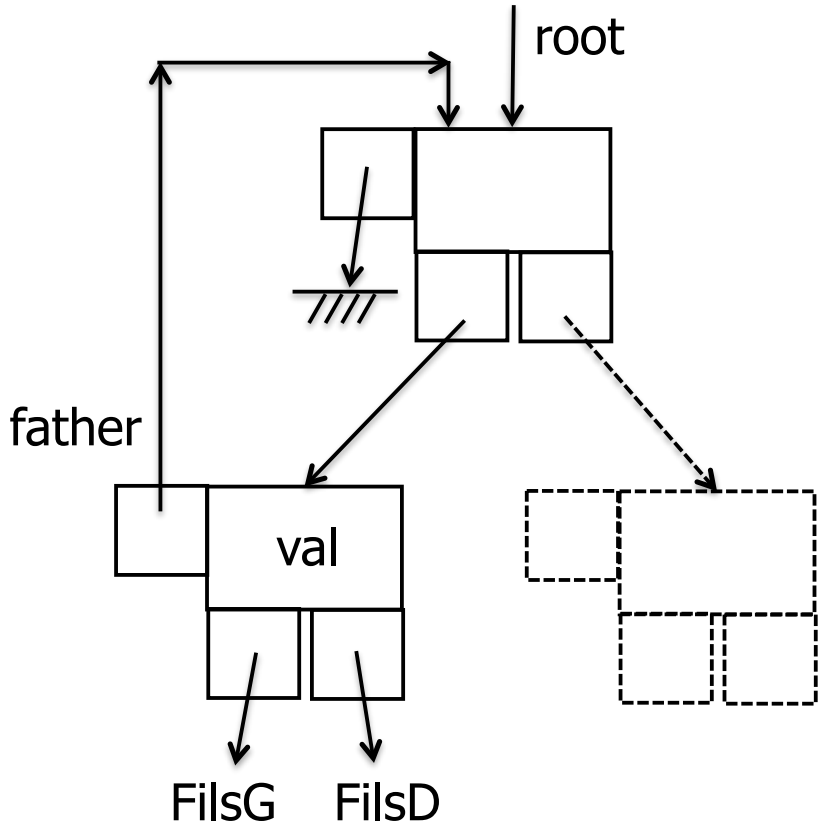
Suppression : contexte

- On suppose que l'on **pointe** sur l'élément à supprimer
  - Pas de recherche
- Pointeur vers le **père**
- Invariant : Conserver la **structure** d'ABR !

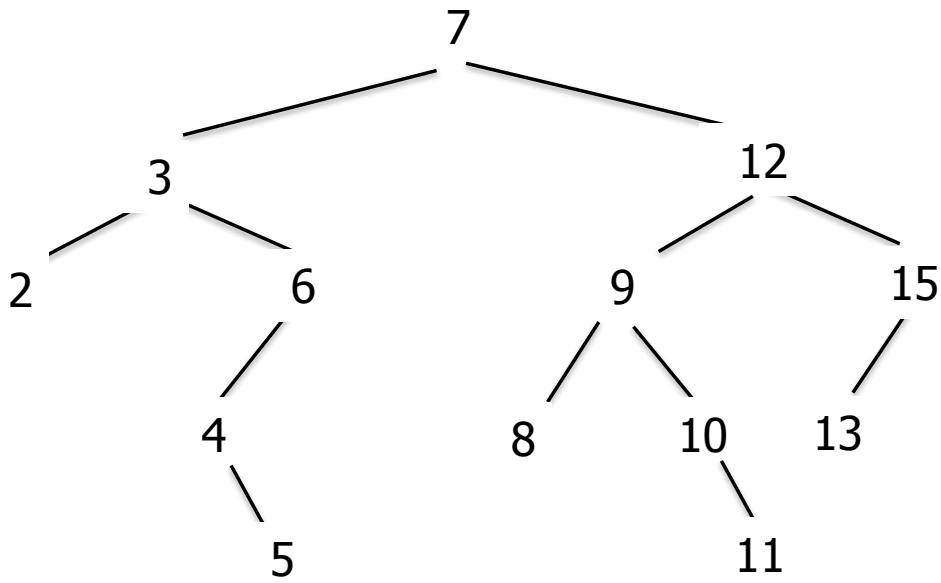
Suppression : contexte

- On suppose que l'on **pointe** sur l'élément à supprimer
  - Pas de recherche
- Pointeur vers le **père**
- Invariant : Conserver la **structure** d'ABR !

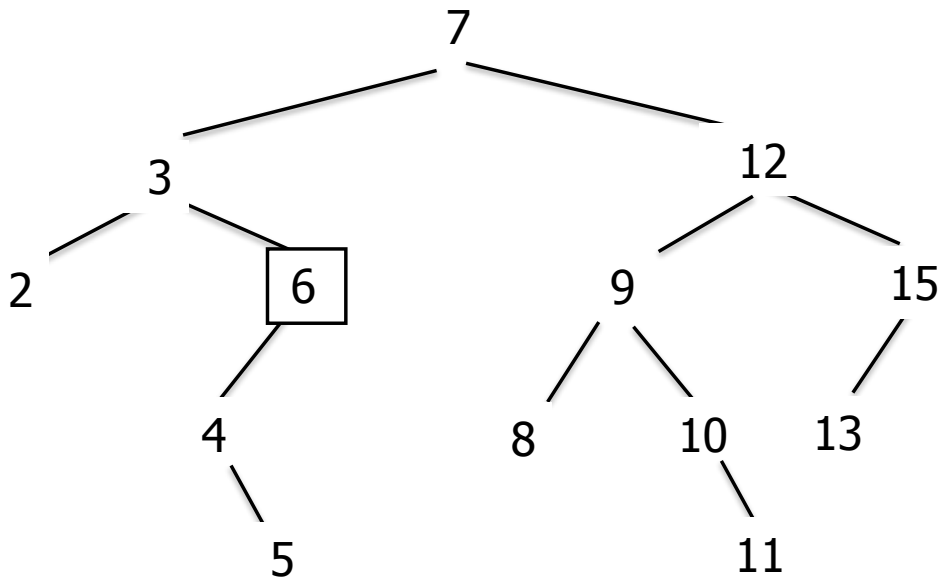
Exemple



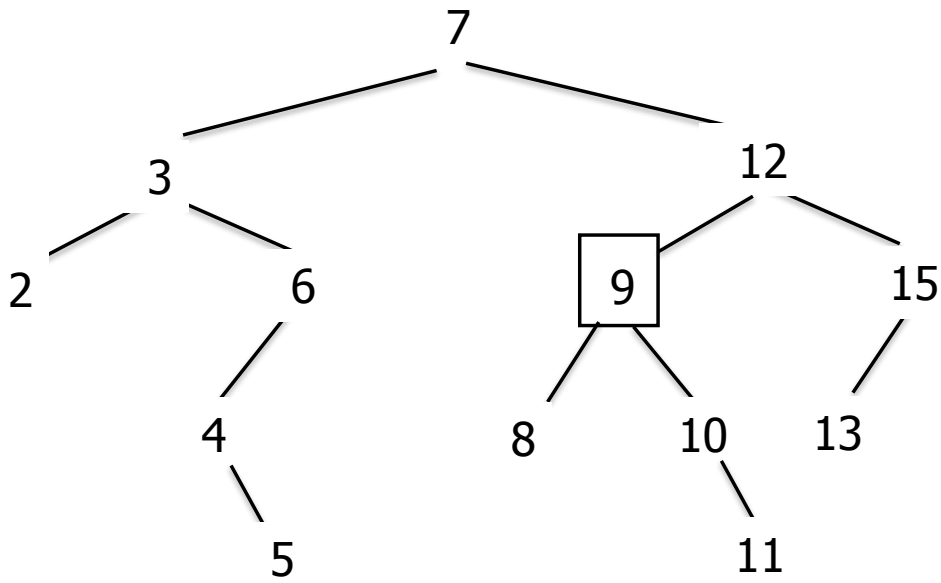
Exemple



Exemple



Exemple



Question générale

Par quoi remplacer le nœud à supprimer ?

Suppression : algorithme

- On remplace **la valeur** du nœud à supprimer par celle de son **successeur** (ou son **prédécesseur**)
  - Sauf cas particuliers (feuille, ...)
- Par définition, ce nœud a **au plus un** fils
- On peut donc lui-même le remplacer par son **unique sous-arbre**
  - Pas de fils $\Longrightarrow$ c'est une feuille $\Longrightarrow$ rien à faire

Suppression : algorithme

- On remplace **la valeur** du nœud à supprimer par celle de son **successeur** (ou son **prédécesseur**)
  - Sauf cas particuliers (feuille, . . .)
- Par définition, ce nœud a **au plus un** fils
- On peut donc lui-même le remplacer par son **unique sous-arbre**
  - Pas de fils $\Longrightarrow$ c'est une feuille $\Longrightarrow$ rien à faire

Suppression : algorithme

- On remplace **la valeur** du nœud à supprimer par celle de son **successeur** (ou son **prédécesseur**)
  - Sauf cas particuliers (feuille, . . .)
- Par définition, ce nœud a **au plus un** fils
- On peut donc lui-même le remplacer par son **unique sous-arbre**
  - Pas de fils $\Longrightarrow$ c'est une feuille $\Longrightarrow$ rien à faire

Question

Quelle est la complexité en pire cas de cet algorithme de suppression d'un élément dans un ABR de hauteur h ?

1. $O(1)$
2. $\Theta(h)$
3. $\Theta(h \log h)$
4. $\Theta(h^2)$
5. Elle ne dépend pas de h.

Corrigé

Quelle est la complexité en pire cas de cet algorithme de suppression d'un élément dans un ABR de hauteur h ?

1. $O(1)$
2. $\Theta(h)$
3. $\Theta(h \log h)$
4. $\Theta(h^2)$
5. Elle ne dépend pas de h.

- **On pointe déjà sur l'élément à supprimer**
- **Recherche du successeur : parcours en profondeur d'un des sous-arbres**

ABR : à retenir

- Insertion, suppression et recherche en $\Theta(h)$
- Importance d'avoir un arbre **équilibré**
  - h faible (au mieux $\log n$)
- Peut-on **garantir** l'équilibre d'un arbre ?

ABR : à retenir

- Insertion, suppression et recherche en $\Theta(h)$
- Importance d'avoir un arbre **équilibré**
  - h faible (au mieux $\log n$)
- Peut-on **garantir** l'équilibre d'un arbre ?

ABR : à retenir

- Insertion, suppression et recherche en $\Theta(h)$
- Importance d'avoir un arbre **équilibré**
  - h faible (au mieux $\log n$)
- Peut-on **garantir** l'équilibre d'un arbre ?

AVL

- Georgy **A**delson-**V**elsky et Evgenii **L**andis (1962)
- ABR **automatiquement équilibré** : $h = \Theta(\log_2 n)$
- Donc **recherche/insertion/suppression en** $\Theta(\log_2 n)$
- Inconvénient : place mémoire

AVL

- Georgy **A**delson-**V**elsky et Evgenii **L**andis (1962)
- ABR **automatiquement équilibré** : $h = \Theta(\log_2 n)$
- Donc **recherche/insertion/suppression en** $\Theta(\log_2 n)$
- Inconvénient : place mémoire

AVL

- Georgy **A**delson-**V**elsky et Evgenii **L**andis (1962)
- ABR **automatiquement équilibré** : $h = \Theta(\log_2 n)$
- Donc **recherche/insertion/suppression en** $\Theta(\log_2 n)$
- Inconvénient : place mémoire

Définitions

- **Déséquilibre** $d(A)$ d'un arbre binaire A : hauteur fils gauche - hauteur fils droit
  - $d(A) \in \mathbb{Z}$
- **Arbre équilibré** : A tel que, $\forall S$ **sous-arbre de** A, $d(S) \in \{-1, 0, 1\}$
- **AVL** : ABR équilibré

Définitions

- **Déséquilibre** $d(A)$ d'un arbre binaire A : hauteur fils gauche - hauteur fils droit
  - $d(A) \in \mathbb{Z}$
- **Arbre équilibré** : A tel que, $\forall S$ **sous-arbre de** A, $d(S) \in \{-1, 0, 1\}$
- **AVL** : ABR équilibré

Définitions

- **Déséquilibre** $d(A)$ d'un arbre binaire A : hauteur fils gauche - hauteur fils droit
  - $d(A) \in \mathbb{Z}$
- **Arbre équilibré** : A tel que, $\forall S$ **sous-arbre de** A, $d(S) \in \{-1, 0, 1\}$
- **AVL** : ABR équilibré

Propriété

- La **hauteur** d'un arbre équilibré (donc en particulier d'un AVL) est en $O(\log_2 n)$
- Démonstration par **récurrence** sur h
  - Soit $n(h)$ le nombre de nœuds d'un arbre équilibré de hauteur h, deux cas co-existent :
  - $n(h) = 2n(h-1)+1$ pour deux sous-arbres de même hauteur $h-1$
  - $n(h) = n(h-1)+n(h-2)+1$ dans le cas contraire
  - $h = \log_2(n+1)-1$ dans le premier cas
  - Résolution compliquée dans le second : $h = \log_\phi(\sqrt{5}(n+2))-2$ (où ϕ nombre d'or : Fibonacci)
  - Au final, encadrement : on reste en *log*.

Propriété

- La **hauteur** d'un arbre équilibré (donc en particulier d'un AVL) est en $O(\log_2 n)$
- Démonstration par **récurrence** sur h
  - Soit $n(h)$ le nombre de nœuds d'un arbre équilibré de hauteur h, deux cas co-existent :
  - $n(h) = 2n(h-1)+1$ pour deux sous-arbres de même hauteur $h-1$
  - $n(h) = n(h-1)+n(h-2)+1$ dans le cas contraire
  - $h = \log_2(n+1)-1$ dans le premier cas
  - Résolution compliquée dans le second : $h = \log_\phi(\sqrt{5}(n+2))-2$ (où ϕ nombre d'or : Fibonacci)
  - Au final, encadrement : on reste en *log*.

Propriété

- La **hauteur** d'un arbre équilibré (donc en particulier d'un AVL) est en $O(\log_2 n)$
- Démonstration par **récurrence** sur h
  - Soit $n(h)$ le nombre de nœuds d'un arbre équilibré de hauteur h, deux cas co-existent :
  - $n(h) = 2n(h-1)+1$ pour deux sous-arbres de même hauteur $h-1$
  - $n(h) = n(h-1)+n(h-2)+1$ dans le cas contraire
  - $h = \log_2(n+1)-1$ dans le premier cas
  - Résolution compliquée dans le second : $h = \log_\phi(\sqrt{5}(n+2))-2$ (où ϕ nombre d'or : Fibonacci)
  - Au final, encadrement : on reste en *log*.

Propriété

- La **hauteur** d'un arbre équilibré (donc en particulier d'un AVL) est en $O(\log_2 n)$
- Démonstration par **récurrence** sur h
  - Soit $n(h)$ le nombre de nœuds d'un arbre équilibré de hauteur h, deux cas co-existent :
  - $n(h) = 2n(h-1)+1$ pour deux sous-arbres de même hauteur $h-1$
  - $n(h) = n(h-1)+n(h-2)+1$ dans le cas contraire
  - $h = \log_2(n+1)-1$ dans le premier cas
  - Résolution compliquée dans le second : $h = \log_\phi(\sqrt{5}(n+2))-2$ (où ϕ nombre d'or : Fibonacci)
  - Au final, encadrement : on reste en *log*.

Propriété

- La **hauteur** d'un arbre équilibré (donc en particulier d'un AVL) est en $O(\log_2 n)$
- Démonstration par **récurrence** sur h
  - Soit $n(h)$ le nombre de nœuds d'un arbre équilibré de hauteur h, deux cas co-existent :
  - $n(h) = 2n(h-1)+1$ pour deux sous-arbres de même hauteur $h-1$
  - $n(h) = n(h-1)+n(h-2)+1$ dans le cas contraire
  - $h = \log_2(n+1)-1$ dans le premier cas
  - Résolution compliquée dans le second : $h = \log_\phi(\sqrt{5}(n+2))-2$ (où ϕ nombre d'or : Fibonacci)
  - Au final, encadrement : on reste en *log*.

AVL : algorithmes

- **Recherche** : cf. ABR
- **Insertion** et **suppression** : besoin de conserver l'équilibre
- $\Longrightarrow$ opérations de **rééquilibrage**

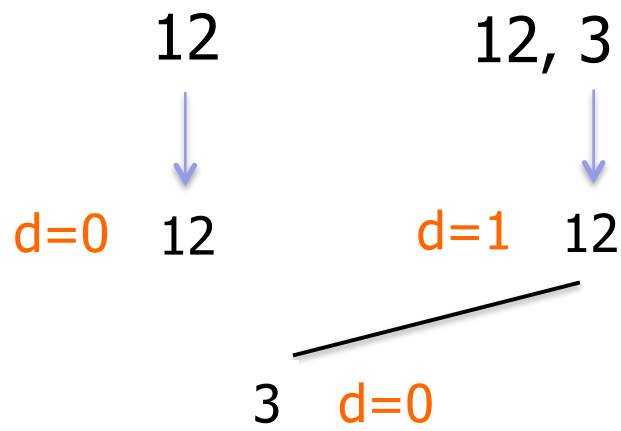
AVL : algorithmes

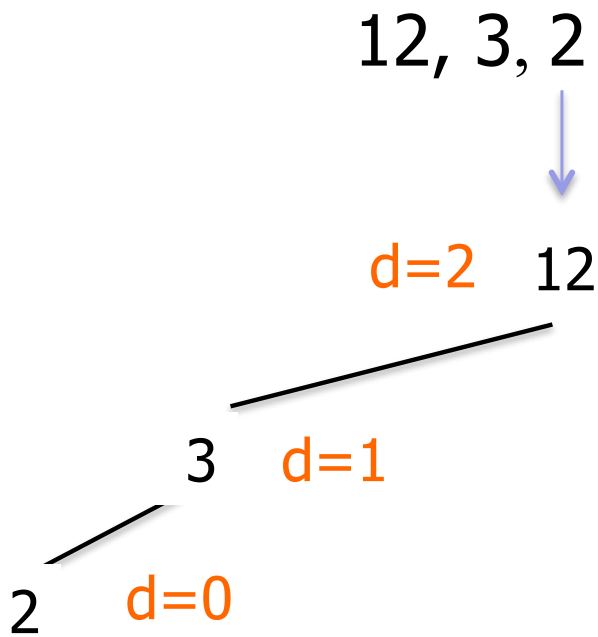
- **Recherche** : cf. ABR
- **Insertion** et **suppression** : besoin de conserver l'équilibre
- $\Longrightarrow$ opérations de **rééquilibrage**

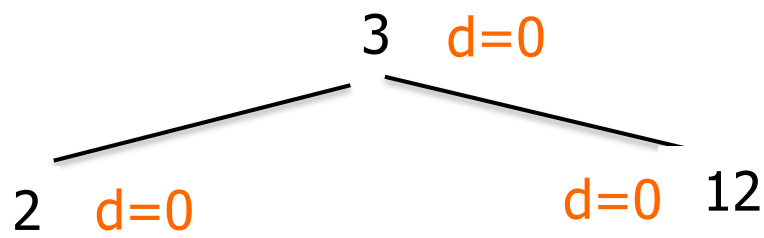
Insertion : exemples

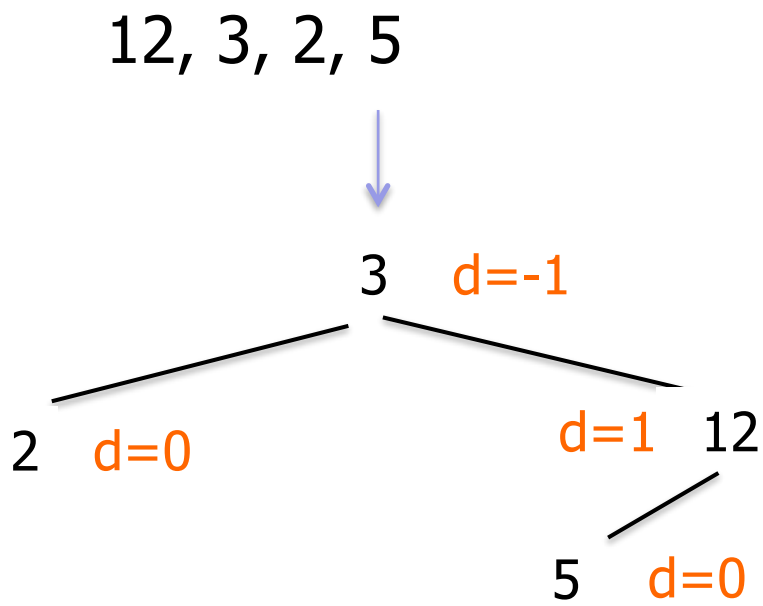
Exercice :

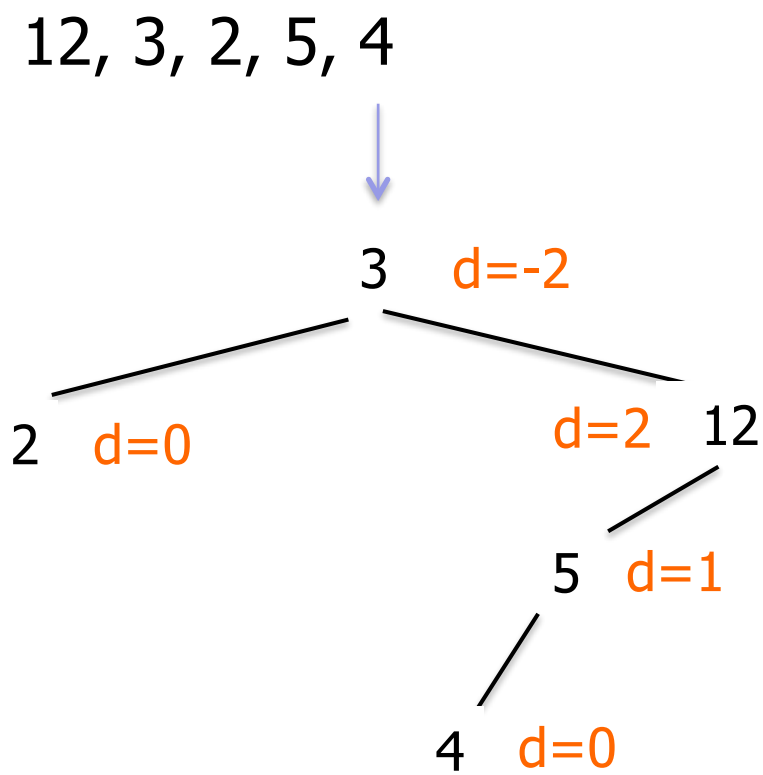
- Dessiner les étapes de la construction d'un AVL (initialement vide) correspondant aux insertions successives des éléments : 12, 3, 2, 5, 4, 7, 9, 11, 14, 10.
  - Noter les hauteurs et les déséquilibres en chaque nœud

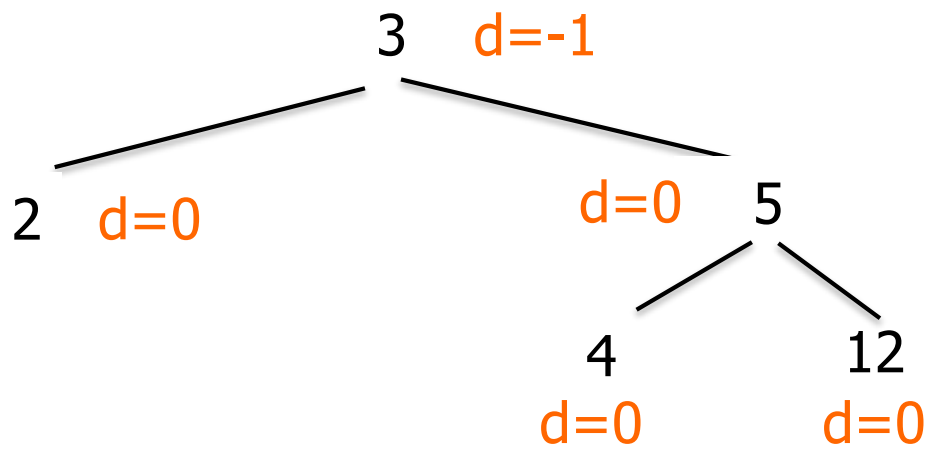

12
12, 3
d=0
12
d=1
12
3
d=0


12, 3, 2
d=2
12
3
d=1
2
d=0

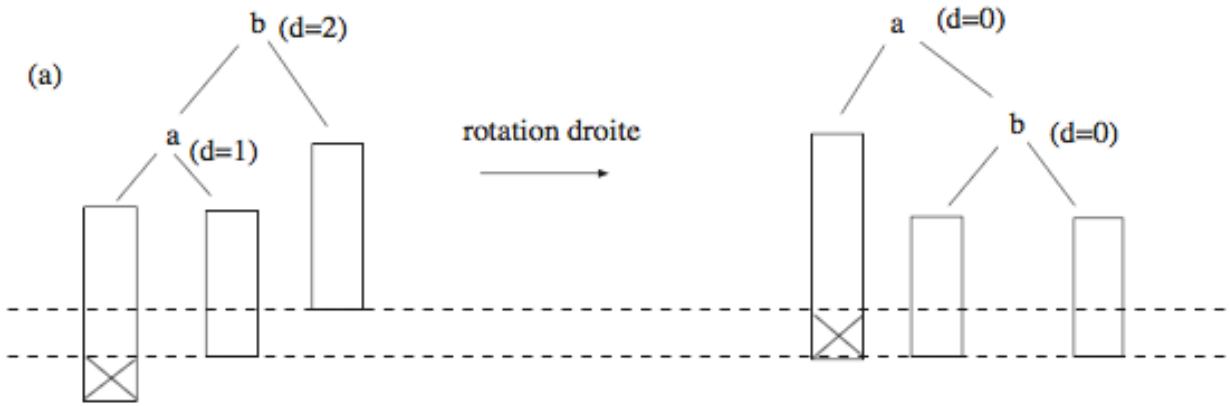

3
d=0
2
d=0
d=0
12


12, 3, 2, 5
3
d=-1
2
d=0
d=1
12
5
d=0

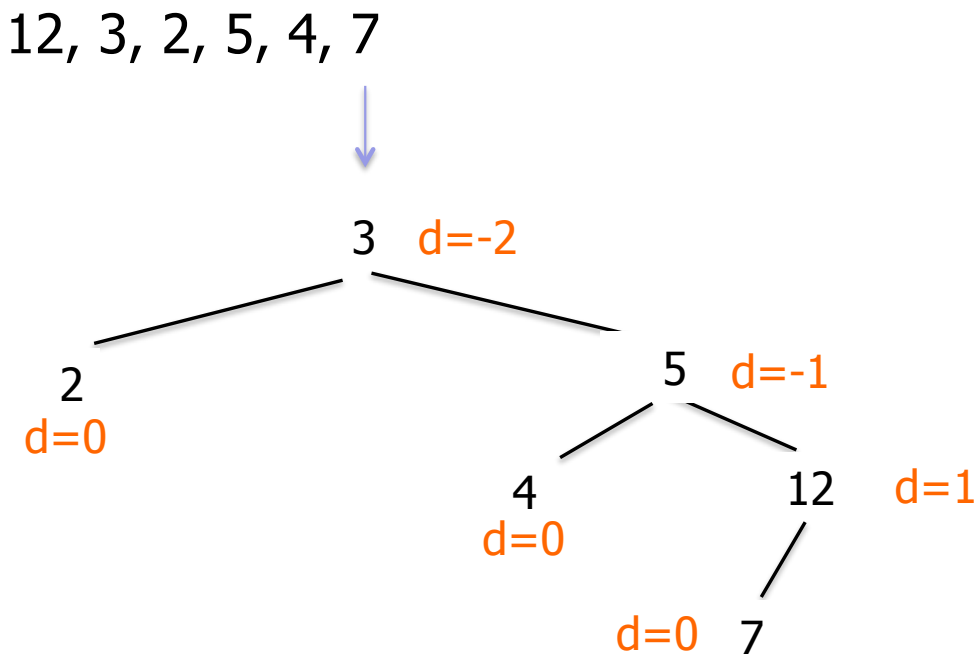

12, 3, 2, 5, 4
3
d=-2
2
d=0
d=2
12
5
d=1
4
d=0


3
d=-1
2
d=0
d=0
5
4
d=0
12
d=0

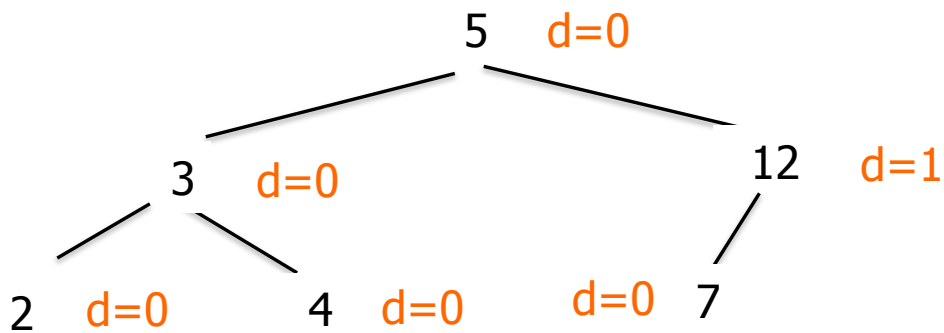
Principe 1

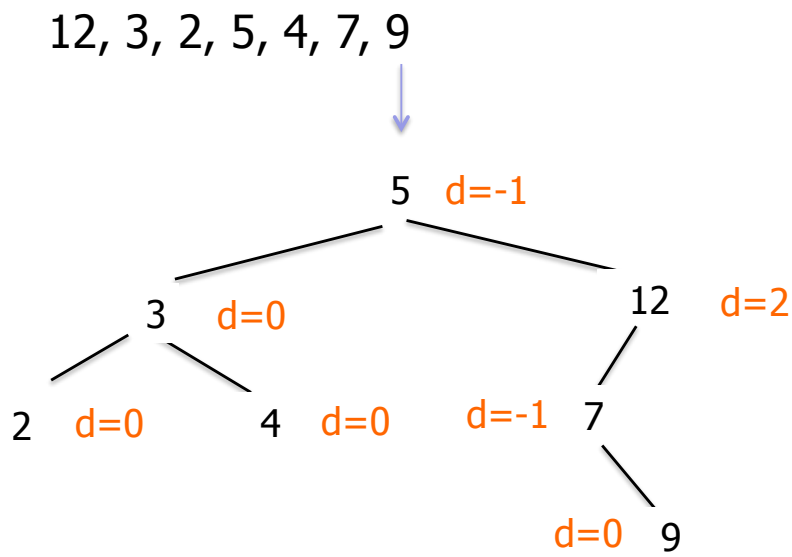


De même symétriquement pour une rotation gauche.

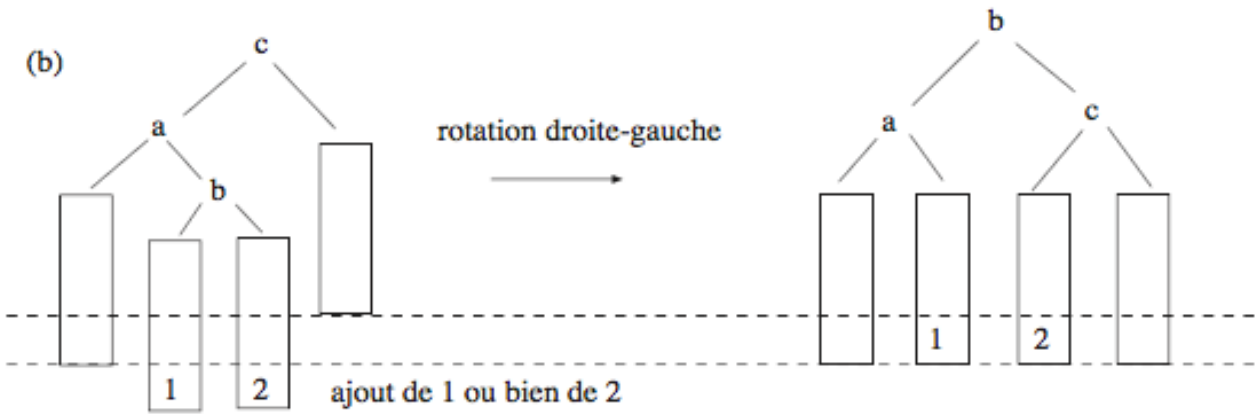

12, 3, 2, 5, 4, 7
3
d=-2
2
d=0
5
d=-1
4
d=0
12
d=1
d=0
7

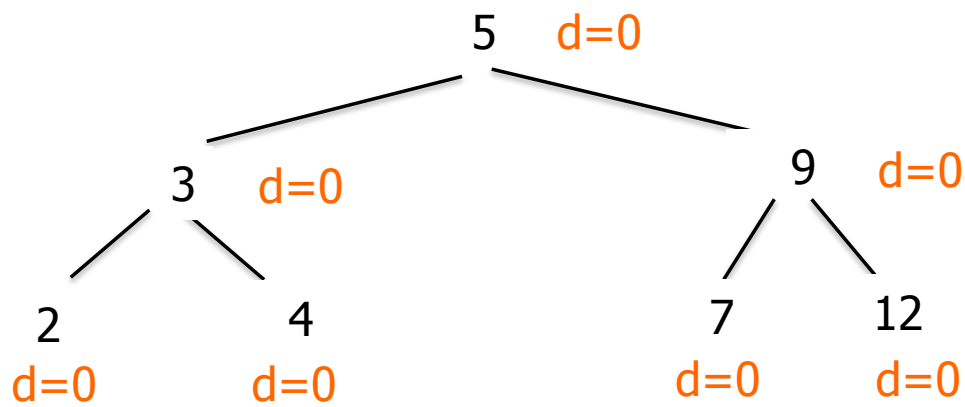
Rotation gauche

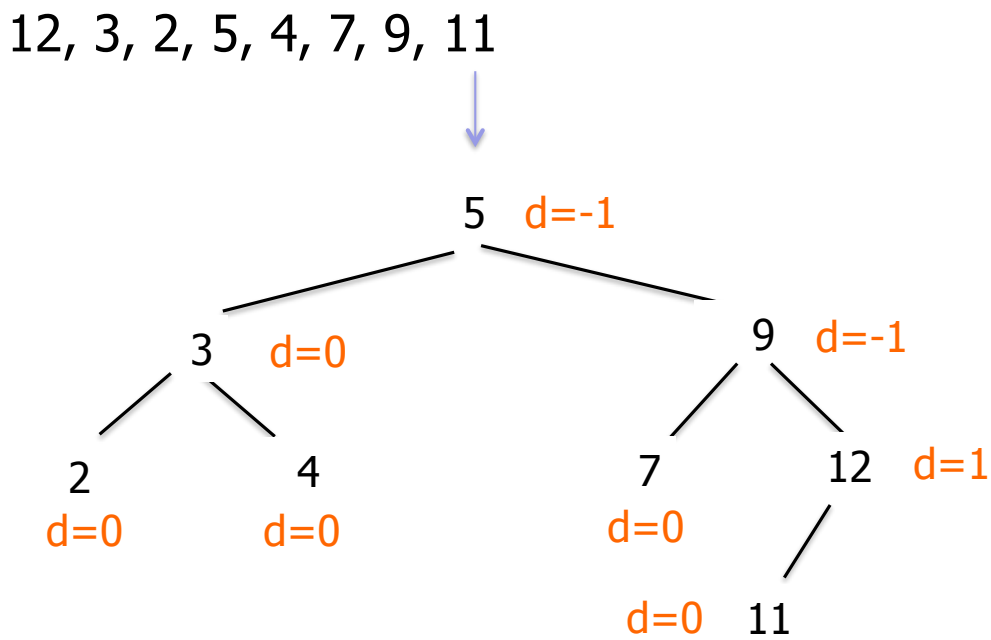



12, 3, 2, 5, 4, 7, 9
5 d=-1
3 d=0
12 d=2
2 d=0
4 d=0
d=-1 7
d=0 9

Principe 2




5
d=0
3
d=0
9
d=0
2
d=0
4
d=0
7
d=0
12
d=0


12, 3, 2, 5, 4, 7, 9, 11
5 d=-1
3 d=0
9 d=-1
2 d=0
4 d=0
7 d=0
12 d=1
d=0 11

Insertion : bilan

- **Rotation simple** gauche ou droite ou **rotation double** gauche-droite ou droite-gauche
  - Remplace le nœud où déséquilibre par son fils ou son petit-fils
- Le choix dépend du **signe du déséquilibre** + du **signe pour le sous-arbre** concerné
  - Même signe $\Longrightarrow$ rotation simple, signes opposés $\Longrightarrow$ rotation double
  - Déséquilibre positif $\Longrightarrow$ la dernière rotation est droite
- Une rotation double suffit toujours : complexité constante

  $\Longrightarrow$ complexité de l'insertion en $O(h) = O(\log_2 n)$

Insertion : bilan

- **Rotation simple** gauche ou droite ou **rotation double** gauche-droite ou droite-gauche
  - Remplace le nœud où déséquilibre par son fils ou son petit-fils
- Le choix dépend du **signe du déséquilibre** + du **signe pour le sous-arbre** concerné
  - Même signe $\Longrightarrow$ rotation simple, signes opposés $\Longrightarrow$ rotation double
  - Déséquilibre positif $\Longrightarrow$ la dernière rotation est droite
- Une rotation double suffit toujours : complexité constante

  $\Longrightarrow$ complexité de l'insertion en $O(h) = O(\log_2 n)$

Insertion : bilan

- **Rotation simple** gauche ou droite ou **rotation double** gauche-droite ou droite-gauche
  - Remplace le nœud où déséquilibre par son fils ou son petit-fils
- Le choix dépend du **signe du déséquilibre** + du **signe pour le sous-arbre** concerné
  - Même signe $\Longrightarrow$ rotation simple, signes opposés $\Longrightarrow$ rotation double
  - Déséquilibre positif $\Longrightarrow$ la dernière rotation est droite
- Une rotation double suffit toujours : complexité constante

  $\Longrightarrow$ complexité de l'insertion en $O(h) = O(\log_2 n)$

Insertion : bilan

- **Rotation simple** gauche ou droite ou **rotation double** gauche-droite ou droite-gauche
  - Remplace le nœud où déséquilibre par son fils ou son petit-fils
- Le choix dépend du **signe du déséquilibre** + du **signe pour le sous-arbre** concerné
  - Même signe $\Longrightarrow$ rotation simple, signes opposés $\Longrightarrow$ rotation double
  - Déséquilibre positif $\Longrightarrow$ la dernière rotation est droite
- Une rotation double suffit toujours : complexité constante

  $\Longrightarrow$ complexité de l'insertion en $O(h) = O(\log_2 n)$

Suppression : exemple

- Encore des rotations !

Complexité

- On ajoute donc un coût kh au coût de la suppression dans un ABR, avec k une constante
- Cela reste du $\Theta(h)$
- La **complexité des trois opérations** recherche, insertion et suppression est donc bien en $\Theta(\log_2 n)$
- Place mémoire ?
  - **Besoin de stocker la hauteur en chaque nœud**
  - "Double" la place mémoire nécessaire

Complexité

- On ajoute donc un coût kh au coût de la suppression dans un ABR, avec k une constante
- Cela reste du $\Theta(h)$
- La **complexité des trois opérations** recherche, insertion et suppression est donc bien en $\Theta(\log_2 n)$
- Place mémoire ?
  - **Besoin de stocker la hauteur en chaque nœud**
  - "Double" la place mémoire nécessaire

Complexité

- On ajoute donc un coût kh au coût de la suppression dans un ABR, avec k une constante
- Cela reste du $\Theta(h)$
- La **complexité des trois opérations** recherche, insertion et suppression est donc bien en $\Theta(\log_2 n)$
- Place mémoire ?
  - **Besoin de stocker la hauteur en chaque nœud**
  - "Double" la place mémoire nécessaire

Conclusion

- Arbres binaires de recherche : intéressants si **équilibrés**
  - Garantie sur la **hauteur**
- **AVL** : une proposition d'ABR équilibré mais qui coûte cher en place mémoire
- D'autres définitions d'ABR équilibré existent
  - **Arbre rouge-noir** (Bayer 1972) : 1 booléen par nœud mais opérations plus compliquées
    - Utilisé par les bibliothèques standard C++ et Java
  - Arbretas (*treap*, Aragon et Seidel 1989) : coût **en moyenne** des 3 opérations en $O(\log n)$
  - Arbre splay (Sleator et Tarjan 1985) : coût **amorti** des 3 opérations en $O(\log n)$

Conclusion

- Arbres binaires de recherche : intéressants si **équilibrés**
  - Garantie sur la **hauteur**
- **AVL** : une proposition d'ABR équilibré mais qui coûte cher en place mémoire
- D'autres définitions d'ABR équilibré existent
  - **Arbre rouge-noir** (Bayer 1972) : 1 booléen par nœud mais opérations plus compliquées
    - Utilisé par les bibliothèques standard C++ et Java
  - Arbretas (*treap*, Aragon et Seidel 1989) : coût **en moyenne** des 3 opérations en $O(\log n)$
  - Arbre splay (Sleator et Tarjan 1985) : coût **amorti** des 3 opérations en $O(\log n)$

Conclusion

- Arbres binaires de recherche : intéressants si **équilibrés**
  - Garantie sur la **hauteur**
- **AVL** : une proposition d'ABR équilibré mais qui coûte cher en place mémoire
- D'autres définitions d'ABR équilibré existent
  - **Arbre rouge-noir** (Bayer 1972) : 1 booléen par nœud mais opérations plus compliquées
    - Utilisé par les bibliothèques standard C++ et Java
  - Arbretas (*treap*, Aragon et Seidel 1989) : coût **en moyenne** des 3 opérations en $O(\log n)$
  - Arbre splay (Sleator et Tarjan 1985) : coût **amorti** des 3 opérations en $O(\log n)$