Arbres II

Arbres de recherche équilibrés

INF2010

Structures de données et algorithmes

Arbres équilibrés

Arbre AVL

- Concepts de base de l’arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l’arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

Arbres équilibrés

Arbre AVL

- Concepts de base de l’arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l’arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

Concepts de base

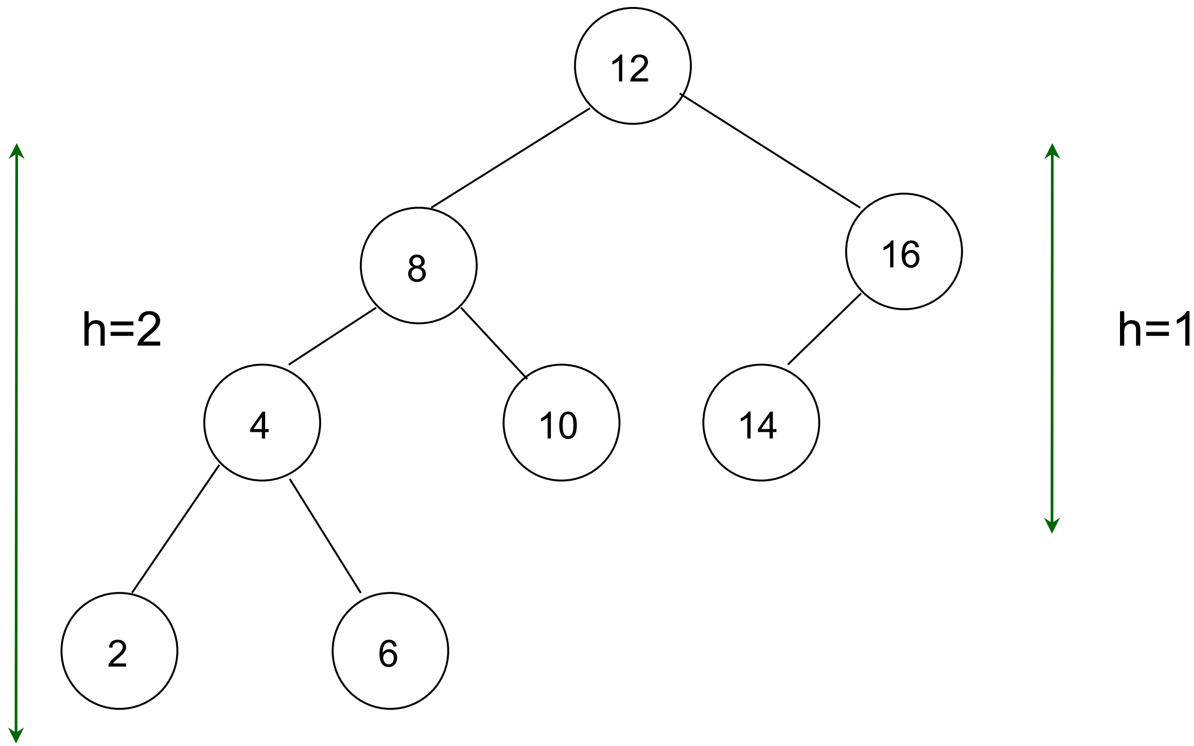
- Situation idéale visée: le sous-arbre de gauche et le sous-arbre de droite ont la même hauteur
  - Ce principe devrait s’appliquerait à tous les noeuds de manière récursive

- Si on appliquait ceci à chaque insertion ou retrait, ce serait très coûteux
  - Il faut donc établir des conditions plus faibles, mais qui nous assurent des gains en performance

Arbres AVL

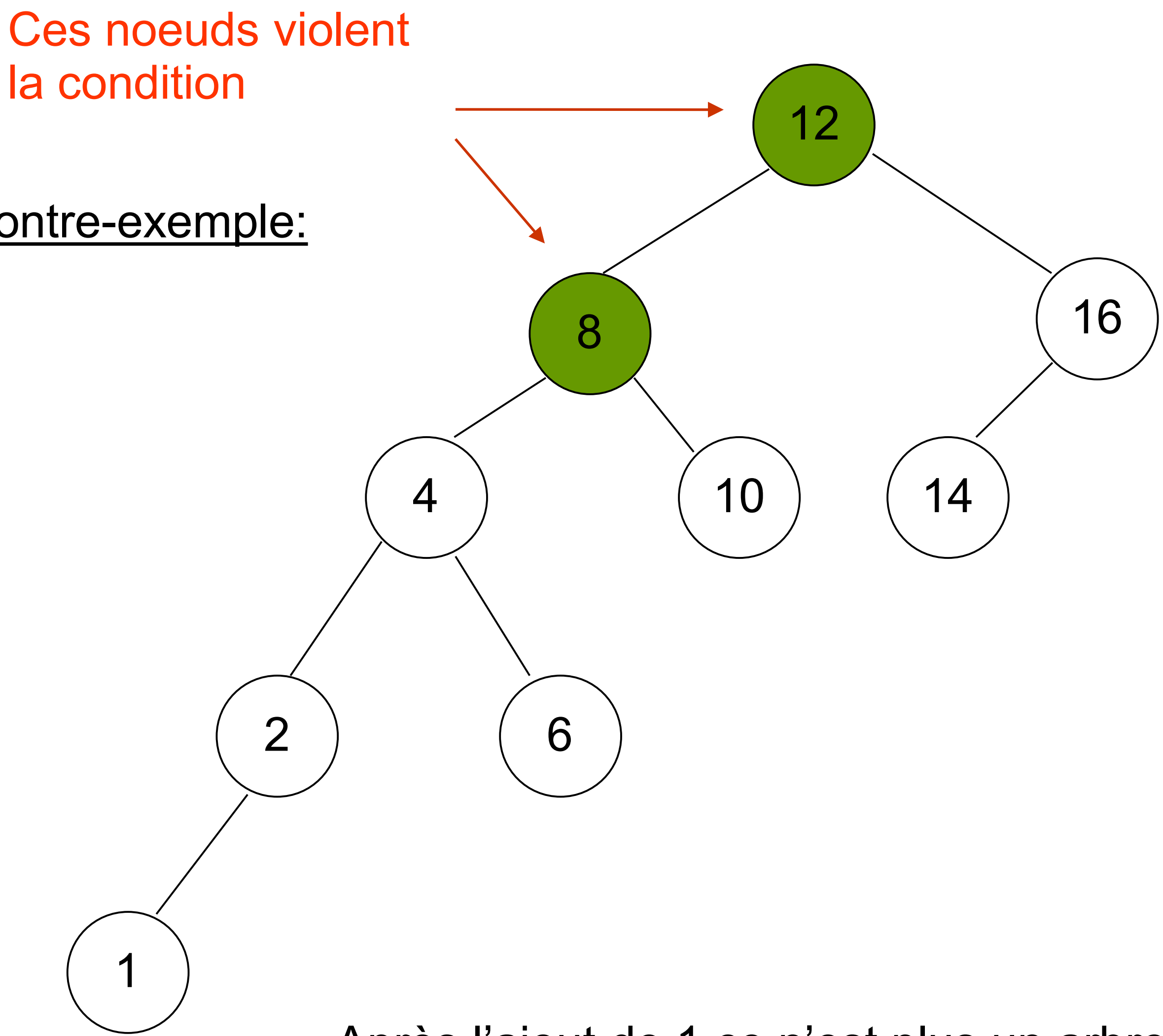
- Définition: arbre de recherche binaire tel que, pour chaque noeud, les hauteurs des ses sous-arbres gauche et droite diffèrent <u>d'au plus</u> 1

<u>Exemple:</u>



Arbres AVL – exemple

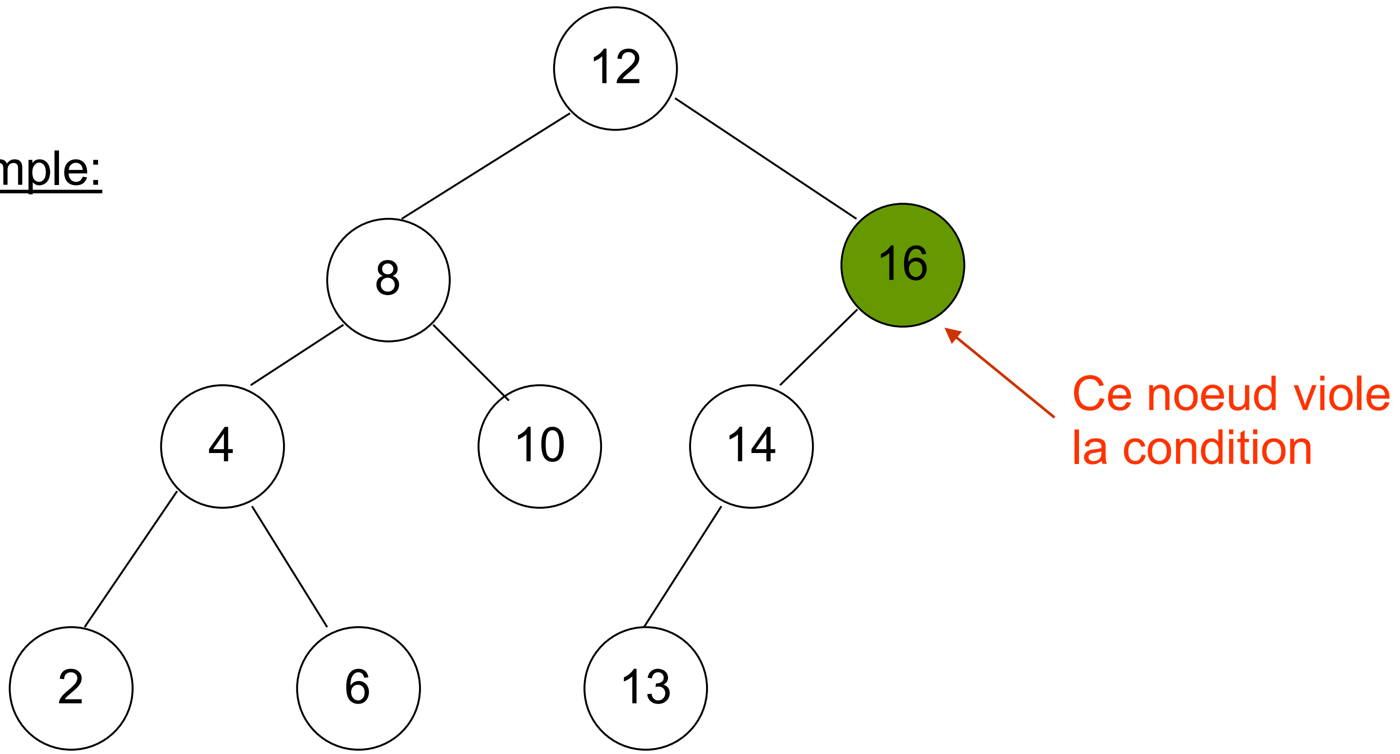
Contre-exemple:



Après l'ajout de 1 ce n'est plus un arbre AVL

Arbres AVL - exemple

<u>Contre-exemple:</u>



Après l'ajout de 13 ce n'est plus un arbre AVL

Arbres AVL

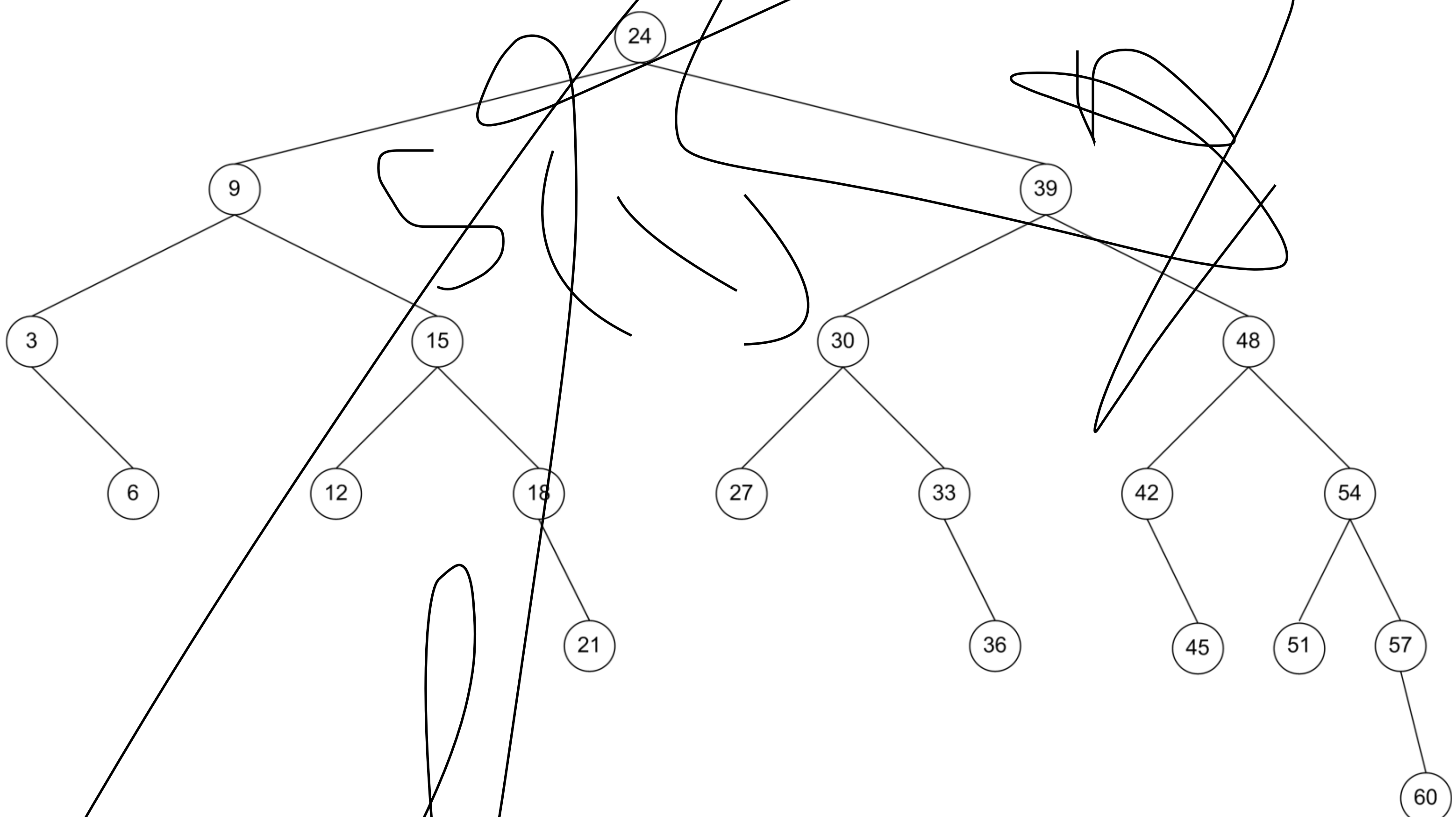
- Avec cette condition, on est assuré de toujours avoir un arbre dont la hauteur est proportionnelle à *lg N ;* La preuve de cette assertion s’obtient comme suit. Soit :

S(h) : le nombre minimal de nœuds dans un AVL de hauteur h

$$\begin{cases} S(0) = 1 \\ S(1) = 2 \\ S(h) = S(h-1) + S(h-2) + 1 \end{cases}$$

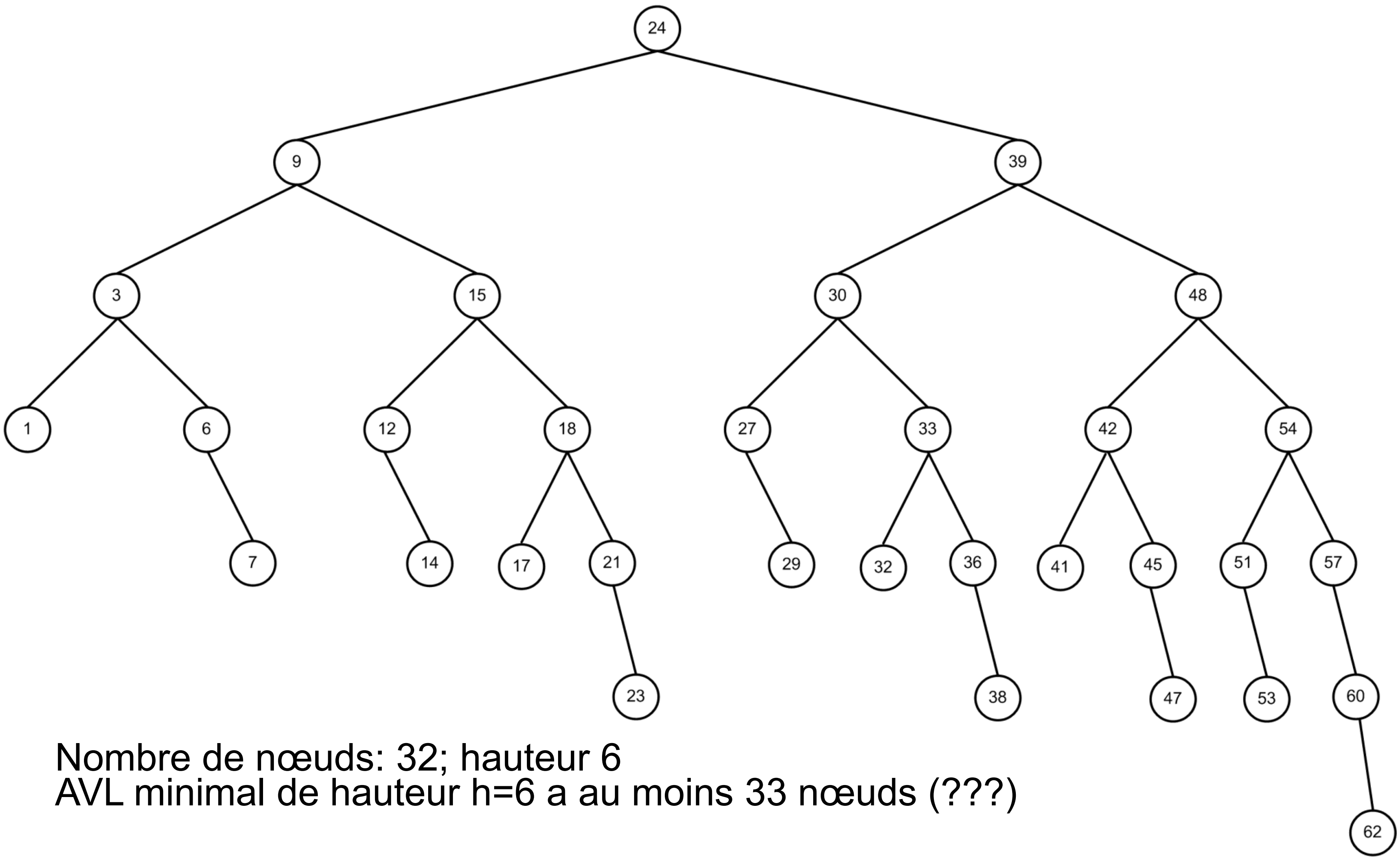
h	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
S(h)	1	2	4	7	12	20	33	54	88

Exemple 1:



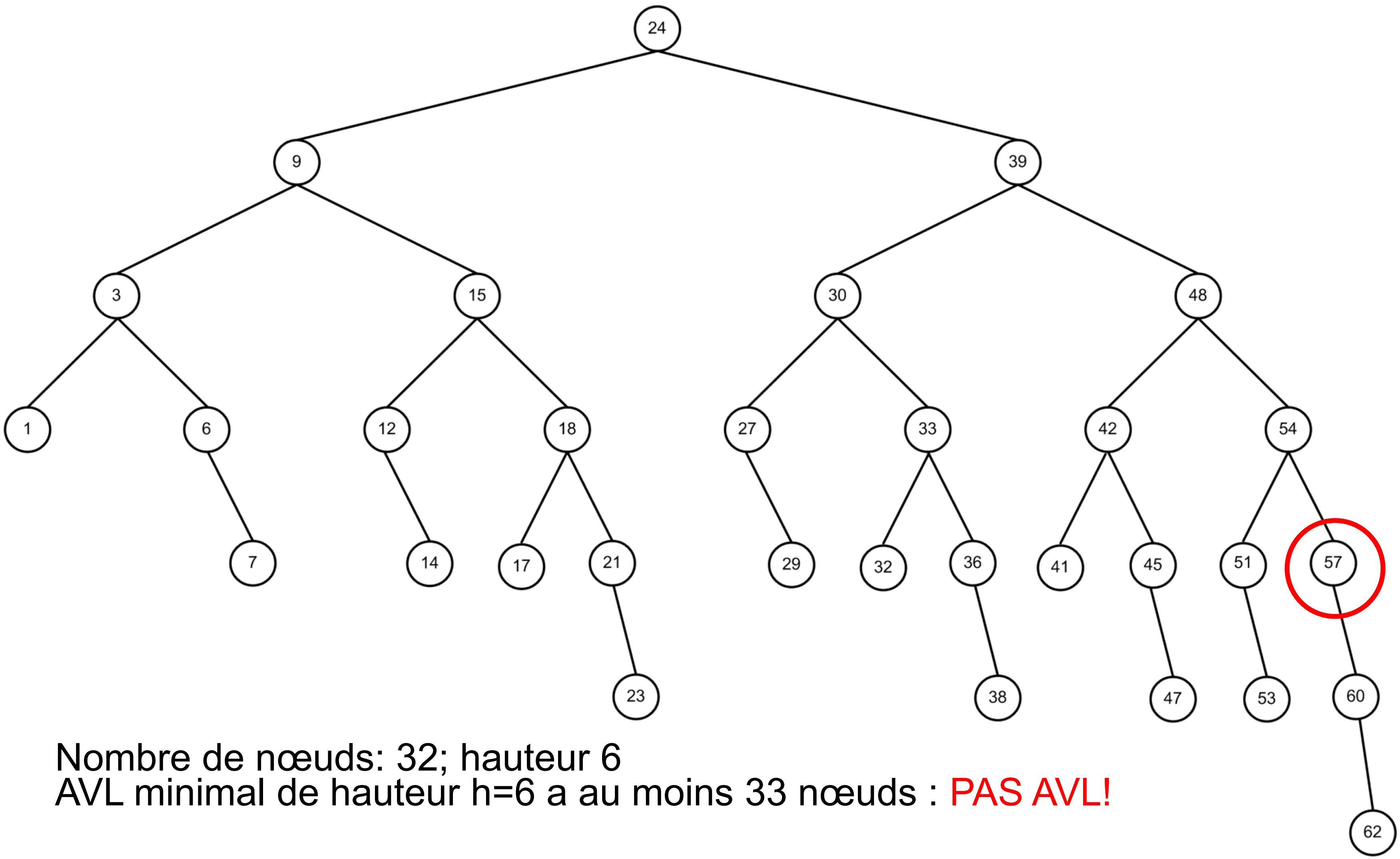
AVL de hauteur h=5 contenant 20 nœud (le minimum pour cette hauteur)

Exemple 2:



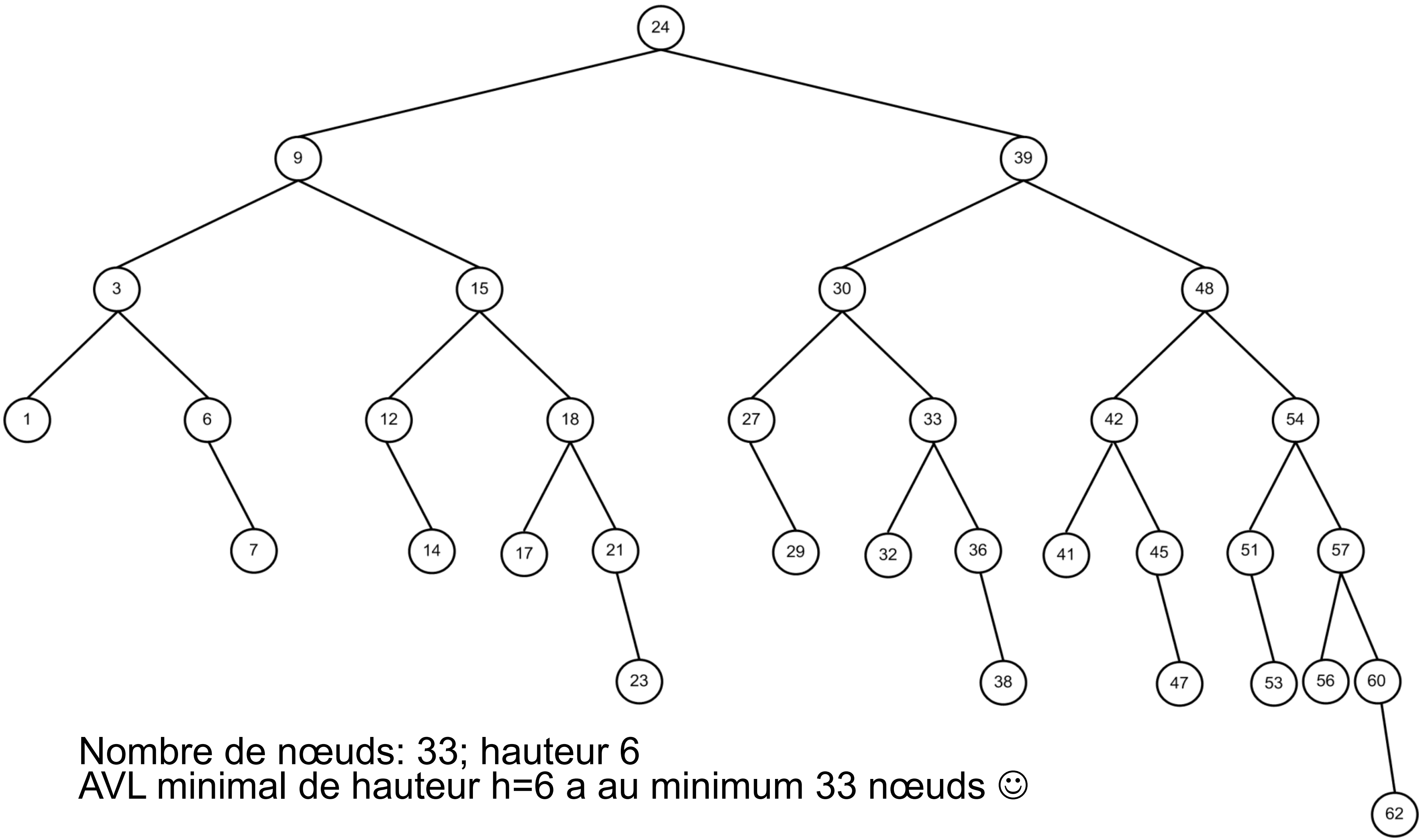
Nombre de nœuds: 32; hauteur 6
AVL minimal de hauteur h=6 a au moins 33 nœuds (???)

Exemple 2:



Nombre de nœuds: 32; hauteur 6
AVL minimal de hauteur h=6 a au moins 33 nœuds : PAS AVL!

Exemple 3:



Nombre de nœuds: 33; hauteur 6
AVL minimal de hauteur h=6 a au minimum 33 nœuds ☺

AVL: Preuve h α lg(N)

Connaissant la fonction de Fibonacci

$F(h) = F(h-1) + F(h-2)$, $F(1) = 1$ et $F(0) = 1$

On remarque que

$S(h) = F(h+2) – 1$

h	0	1	2	3	4	5	6	7	8
F(h)	1	1	2	3	5	8	13	21	34

h	0	1	2	3	4	5	6	7	8
S(h)	1	2	4	7	12	20	33	54	88

$F(h) \approx \Phi^{h+1}/ \sqrt{5}$; $\Phi = (1+ \sqrt{5}) /2$ (Φ est appelé le nombre d’or)

$S(h) \approx \Phi^{h+3}/ \sqrt{5} – 1$

AVL: Preuve h α lg(N)

Si S(h) et le nombre minimal de nœuds qu'un AVL de hauteur h peut contenir, alors l'AVL à N nœuds le plus haut a une hauteur h_{max}.

Pour tout N > 0, on peut poser $S(h_1) \leq N < S(h_2) \Rightarrow h_{max} = h_1$

On a donc $N \geq \Phi^{h_{max}+3} / \sqrt{5} - 1$, il en ressort que

$$h_{max}+3 \leq [\log_2(N+1)+\log_2(\sqrt{5})]/\log_2(\Phi)$$

Ainsi, un AVL contenant N nœuds a au plus une hauteur h_{max}

$$h_{max} \leq 1.440 \cdot \log_2(N+1) - 1.328$$

Le temps d'accès en O(lg(N)) est donc garanti!

Arbres équilibrés

Arbre AVL

- Concepts de base de l'arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l'arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

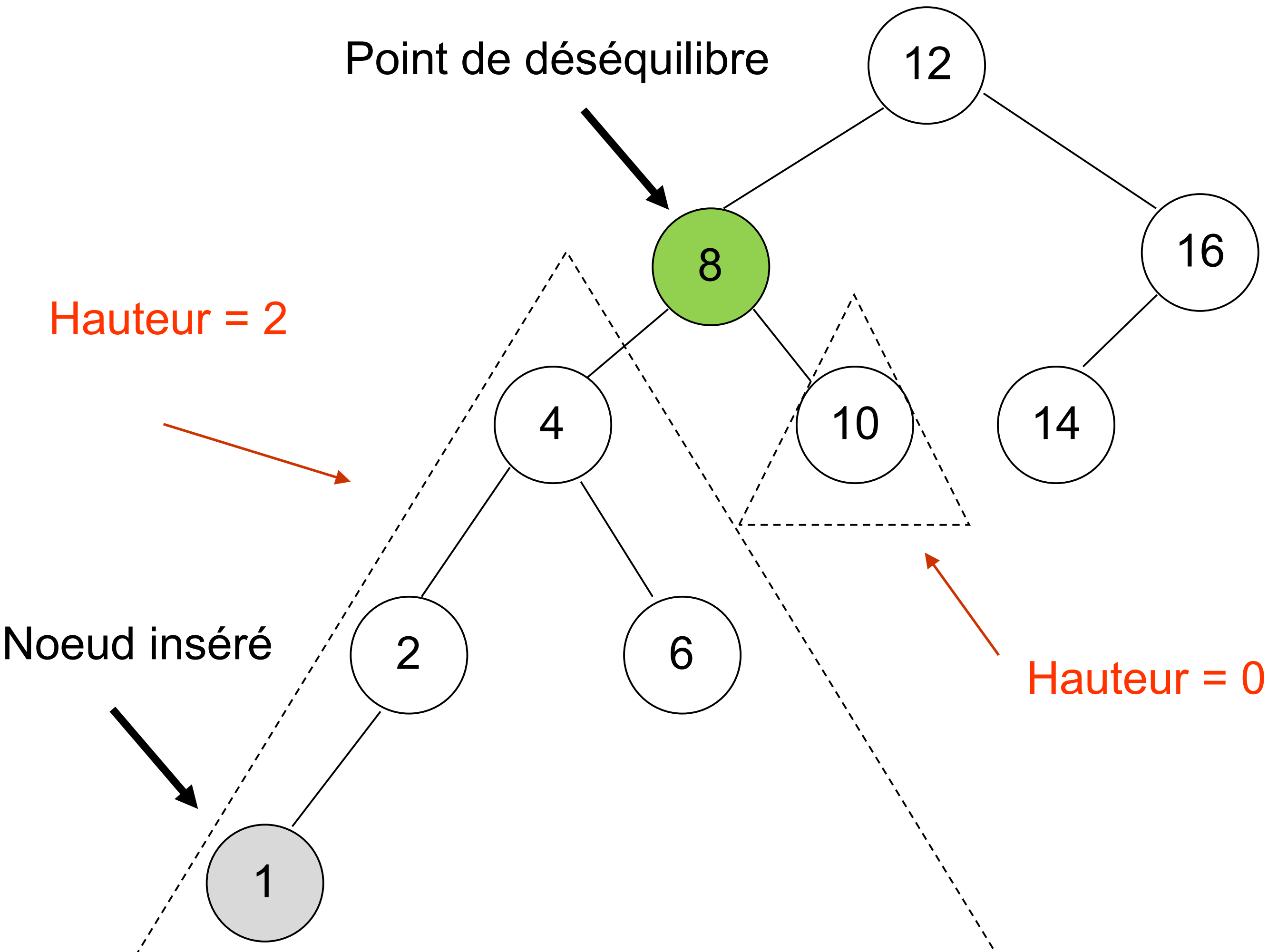
Arbres AVL

- Après chaque insertion ou retrait, il faut rétablir l'équilibre s'il a été rompu par l'opération

- Observation importante: après une insertion/retrait, seuls les noeuds qui sont sur le chemin du point d'insertion/retrait à la racine sont susceptibles d'être déséquilibrés

- Le rééquilibrage se fait au moyen de rotations simples ou doubles

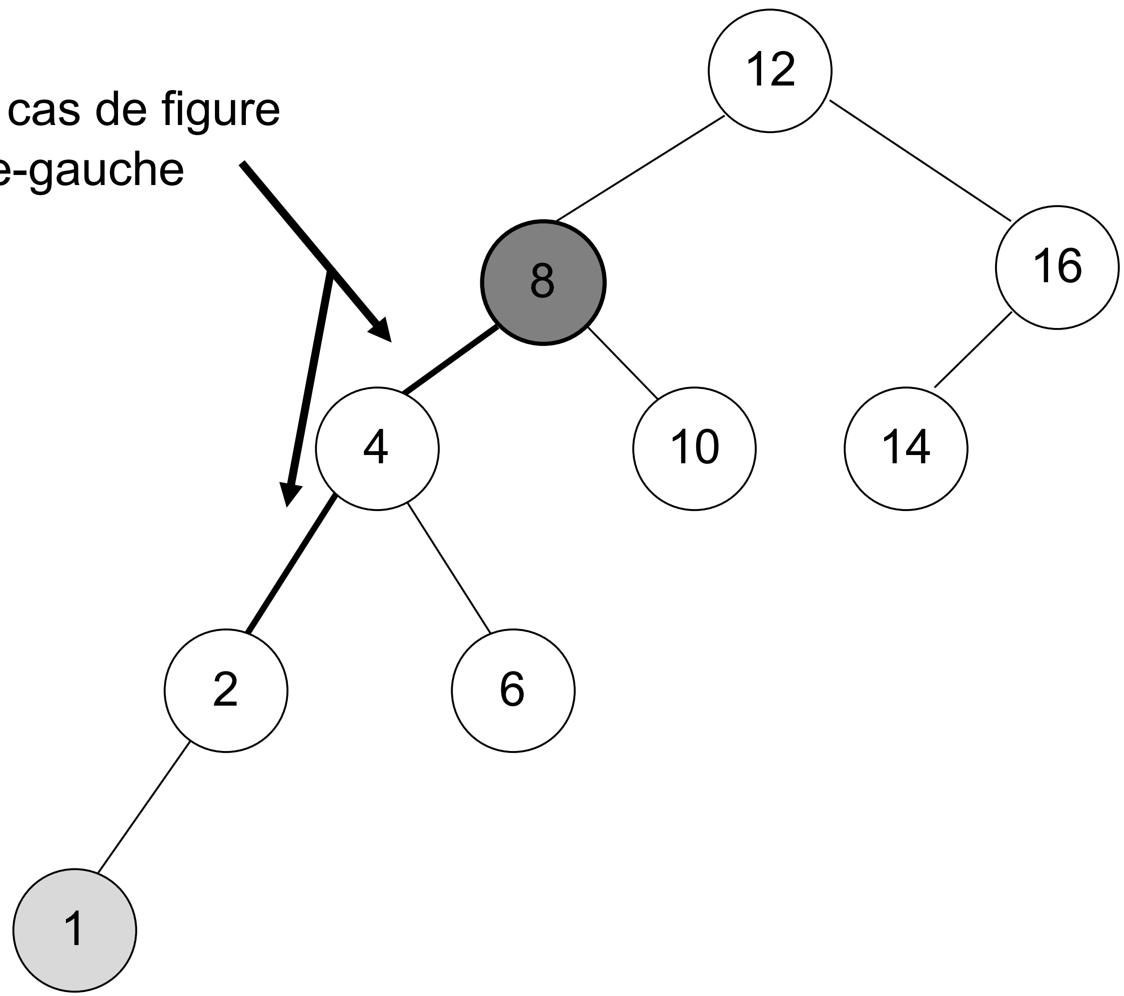
1. Exemple de rotation simple



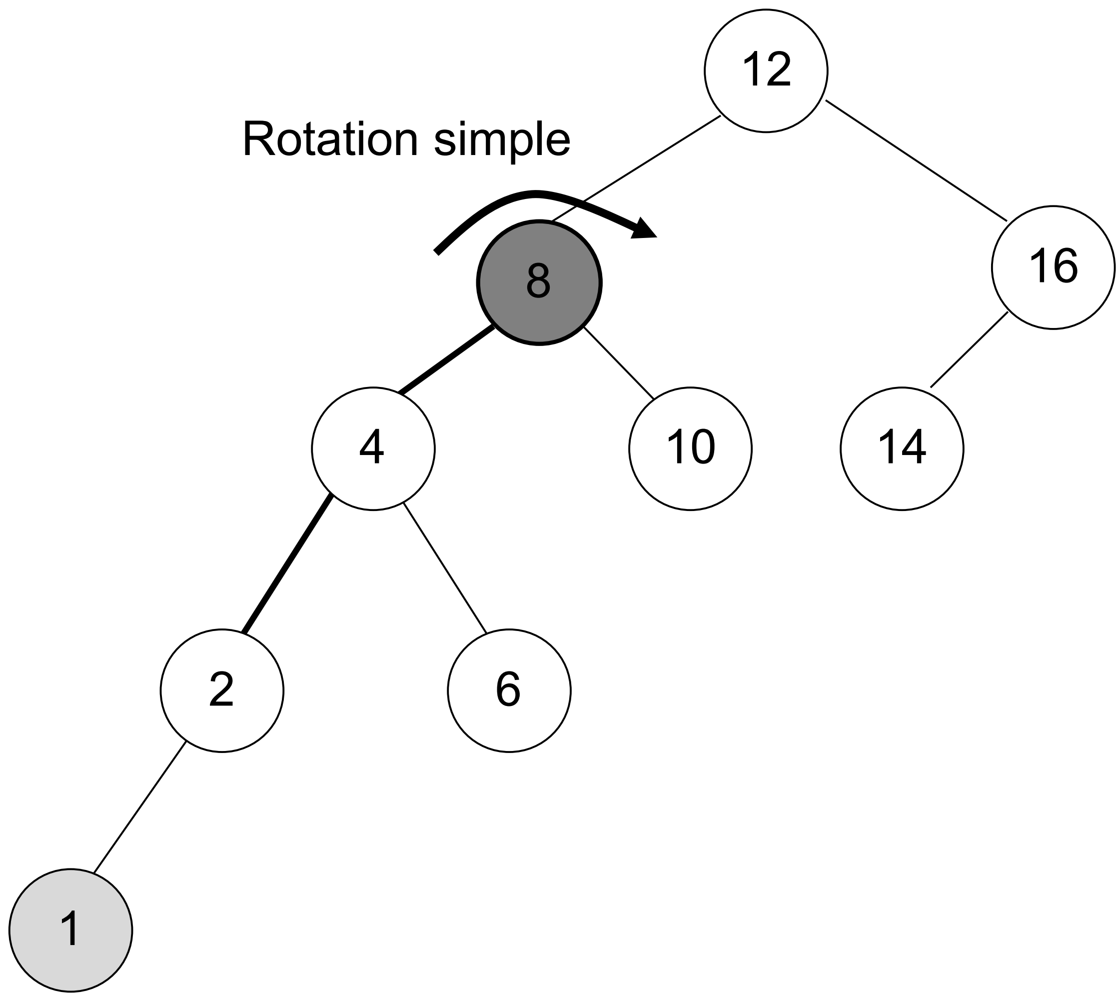
Après l'ajout de 1 ce n'est plus un arbre AVL

1. Exemple de rotation simple

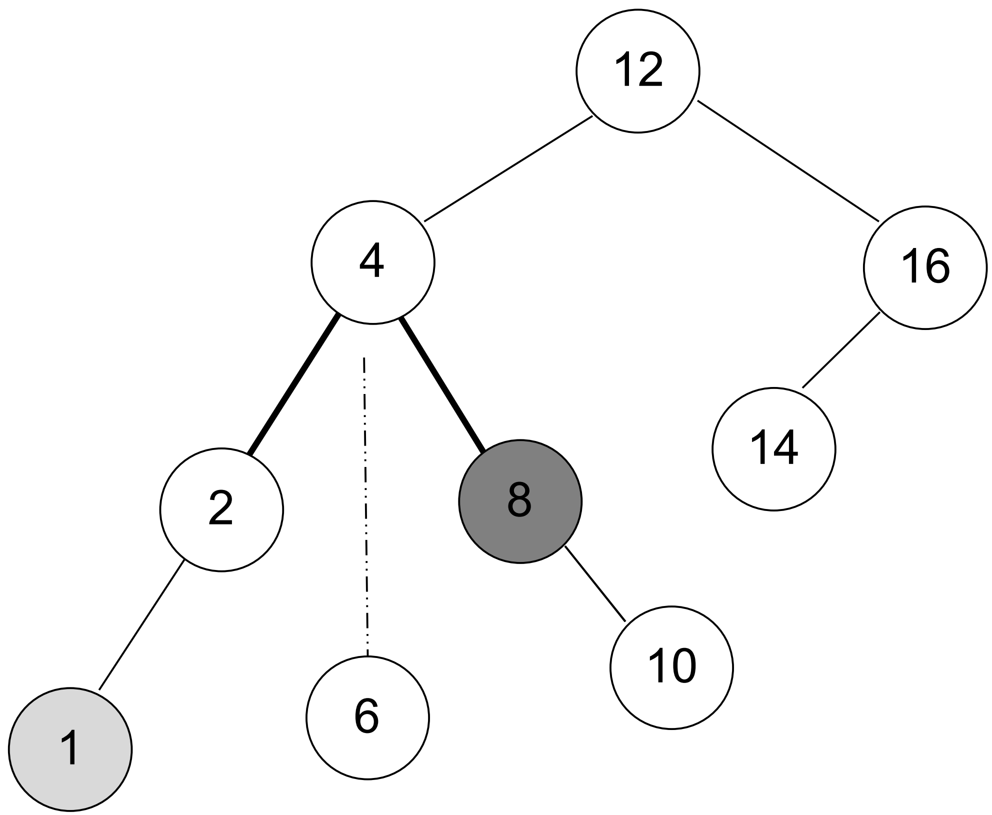
Premier cas de figure gauche-gauche



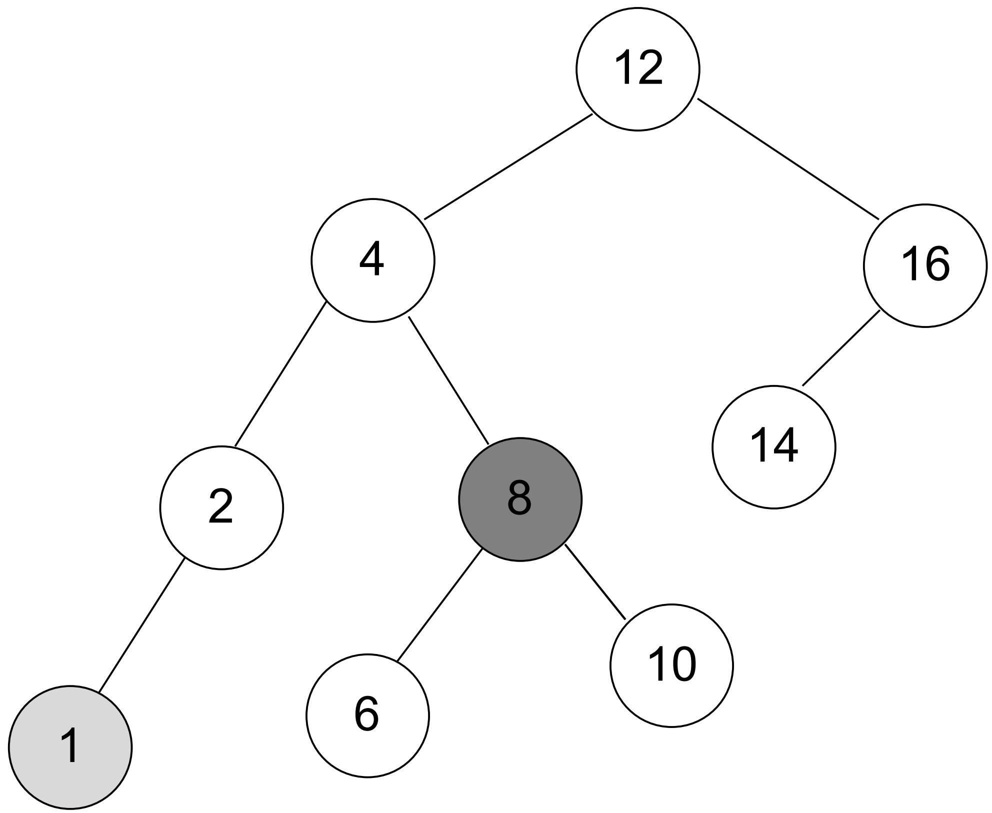
1. Exemple de rotation simple



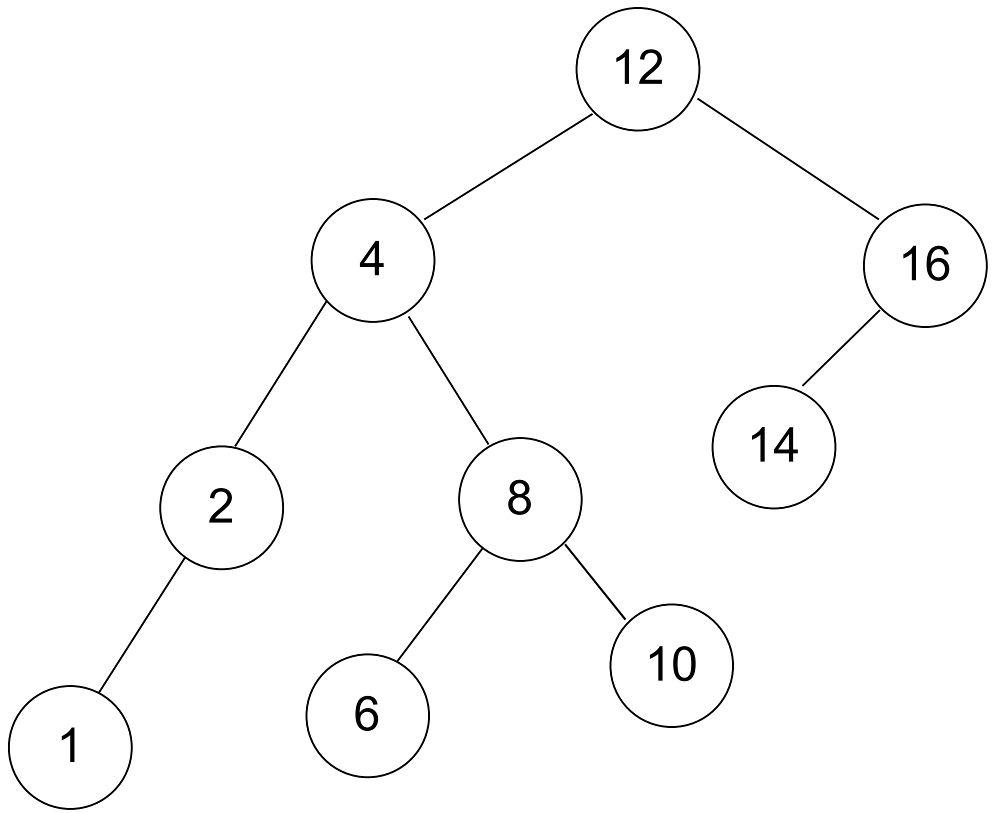
1. Exemple de rotation simple



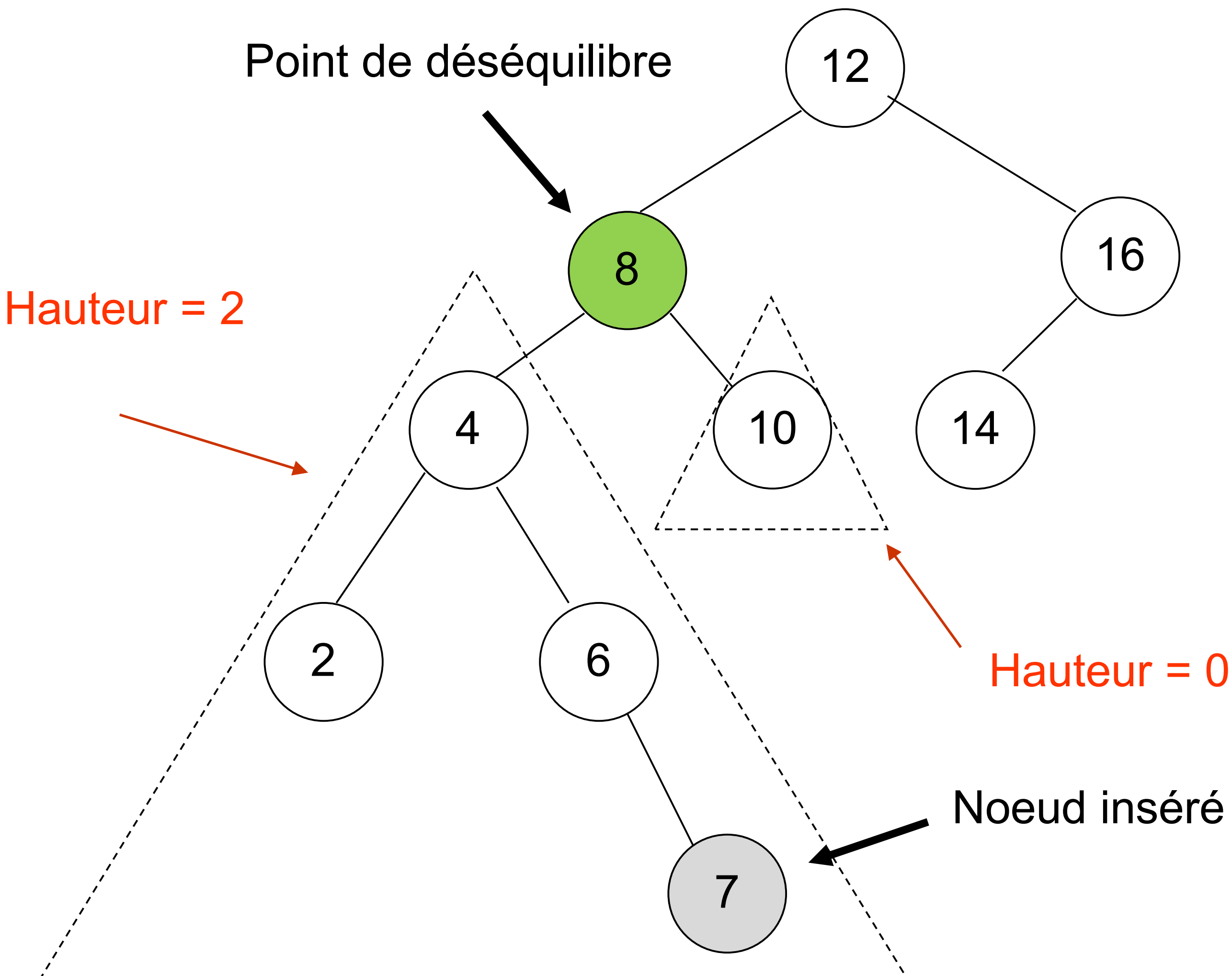
1. Exemple de rotation simple



1. Exemple de rotation simple



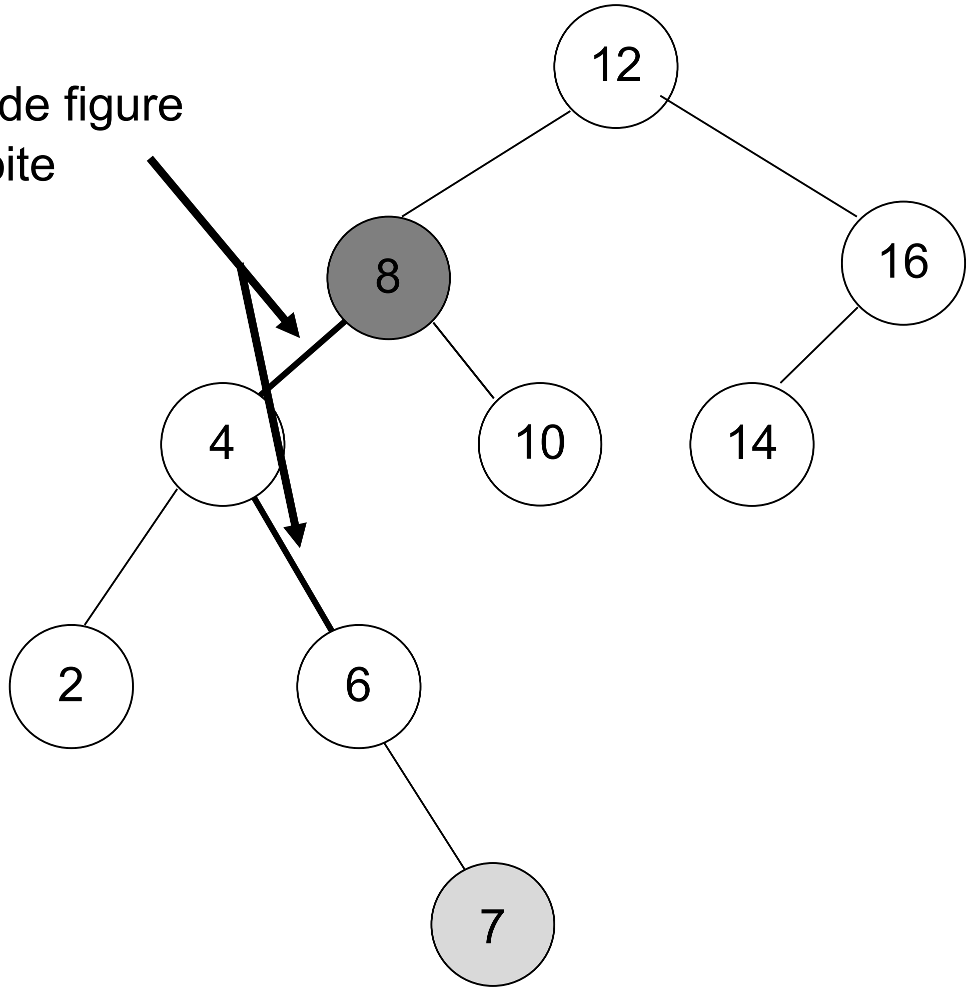
2. Exemple de rotation double



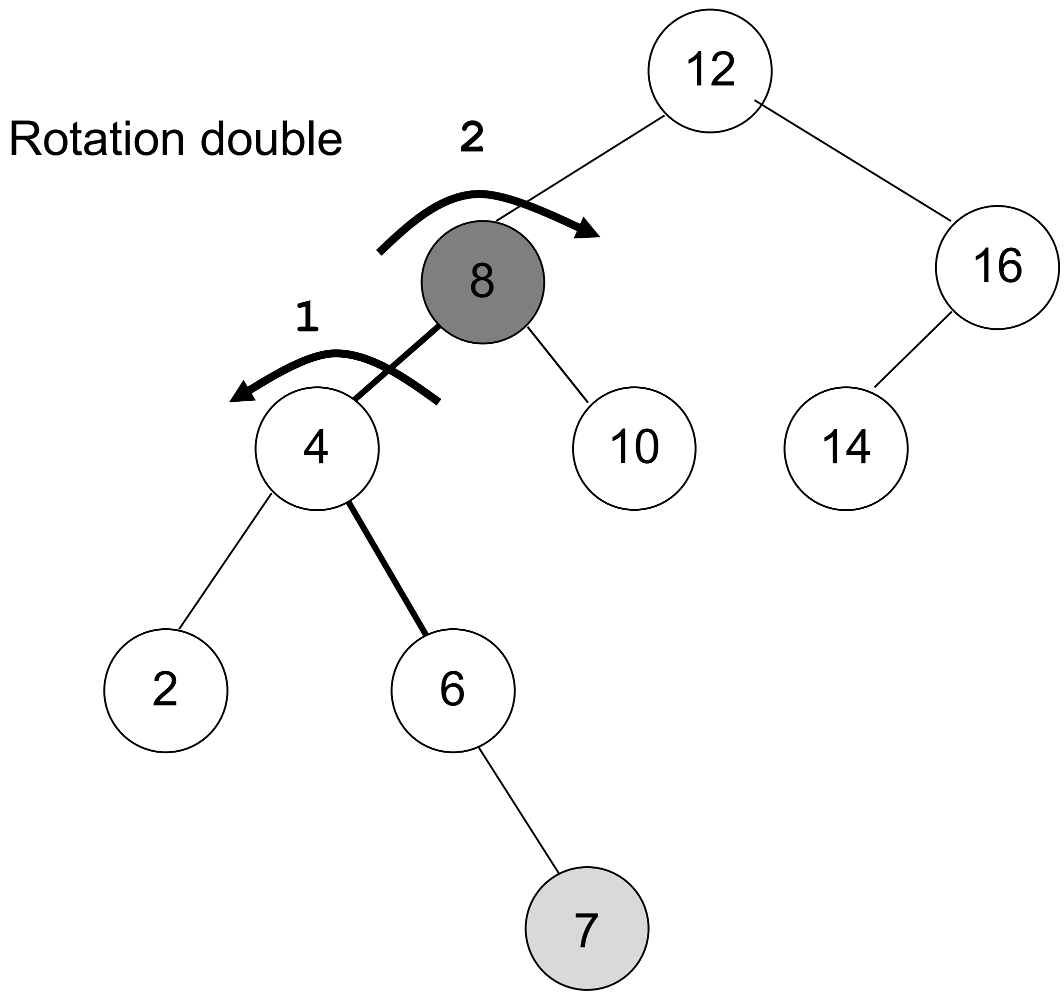
Après l'ajout de 1 ce n'est plus un arbre AVL

2. Exemple de rotation double

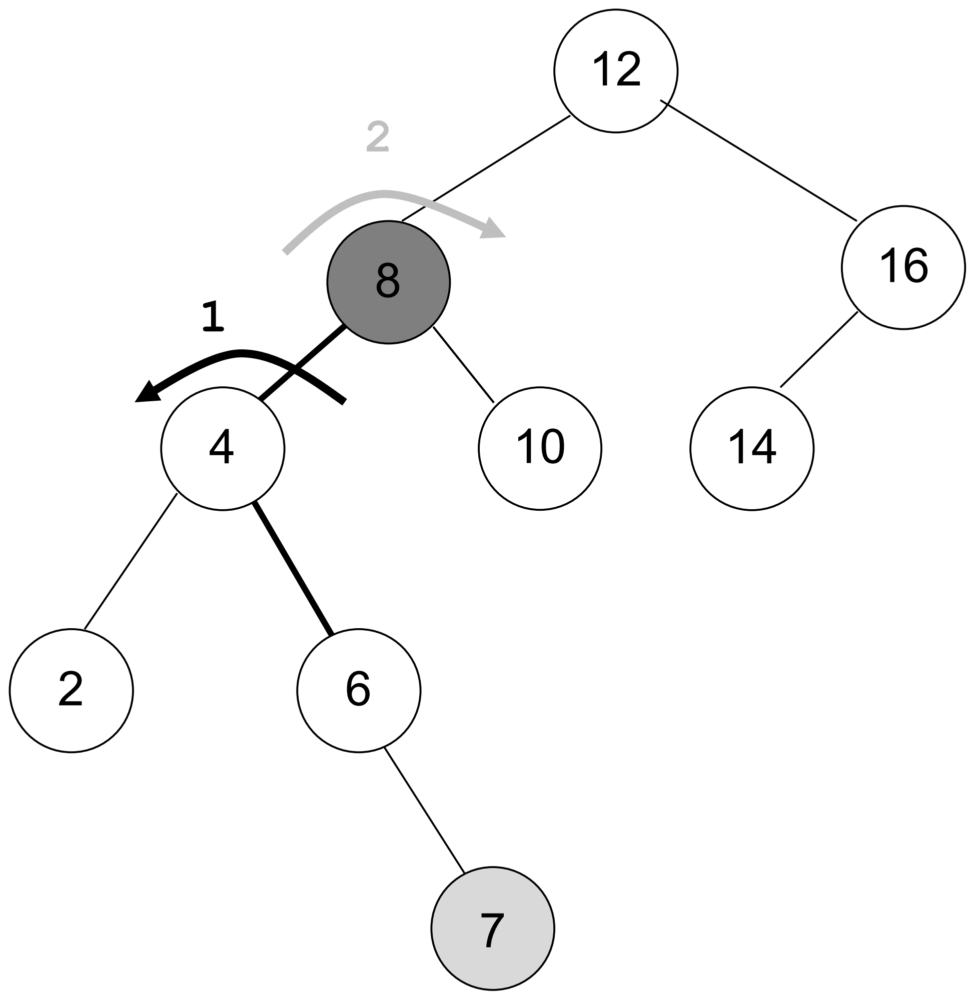
Second cas de figure
gauche-droite



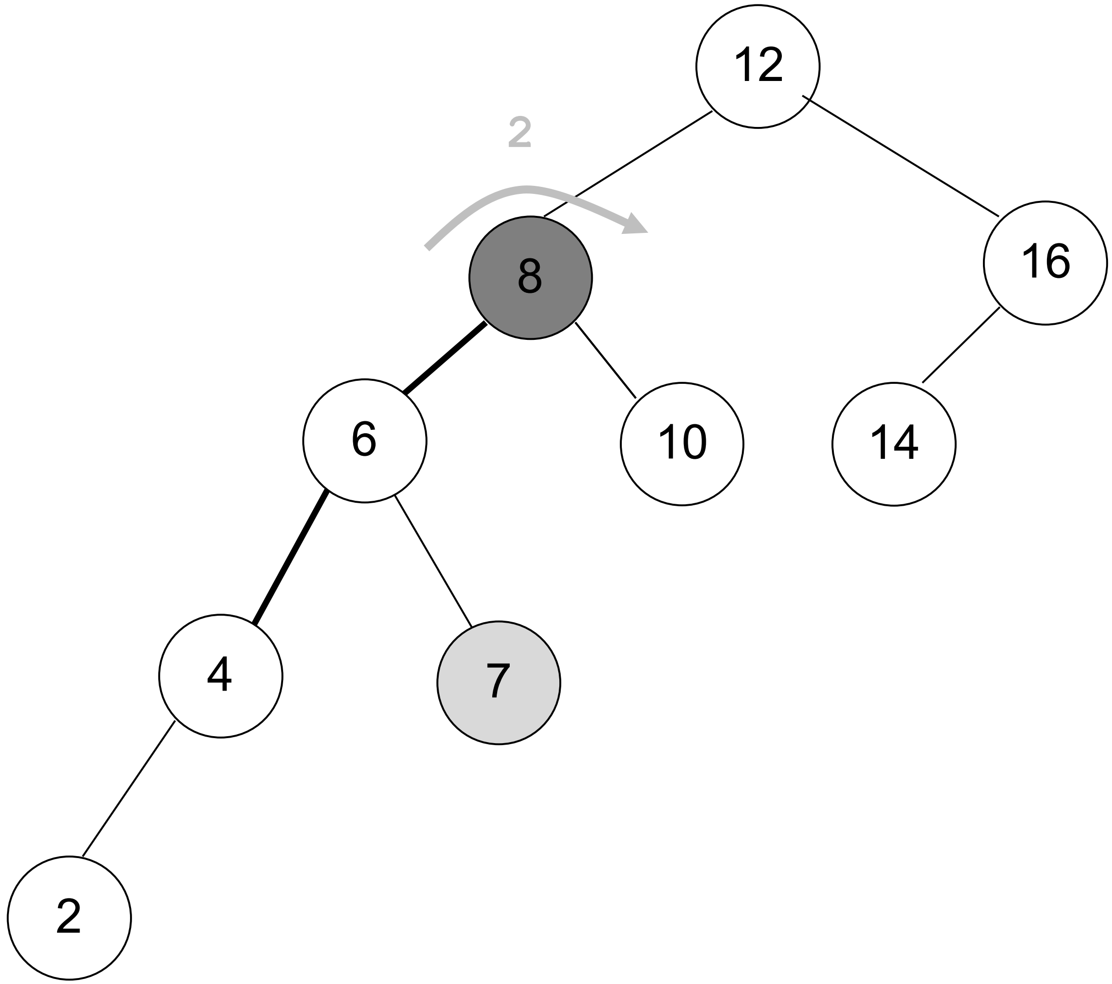
2. Exemple de rotation double



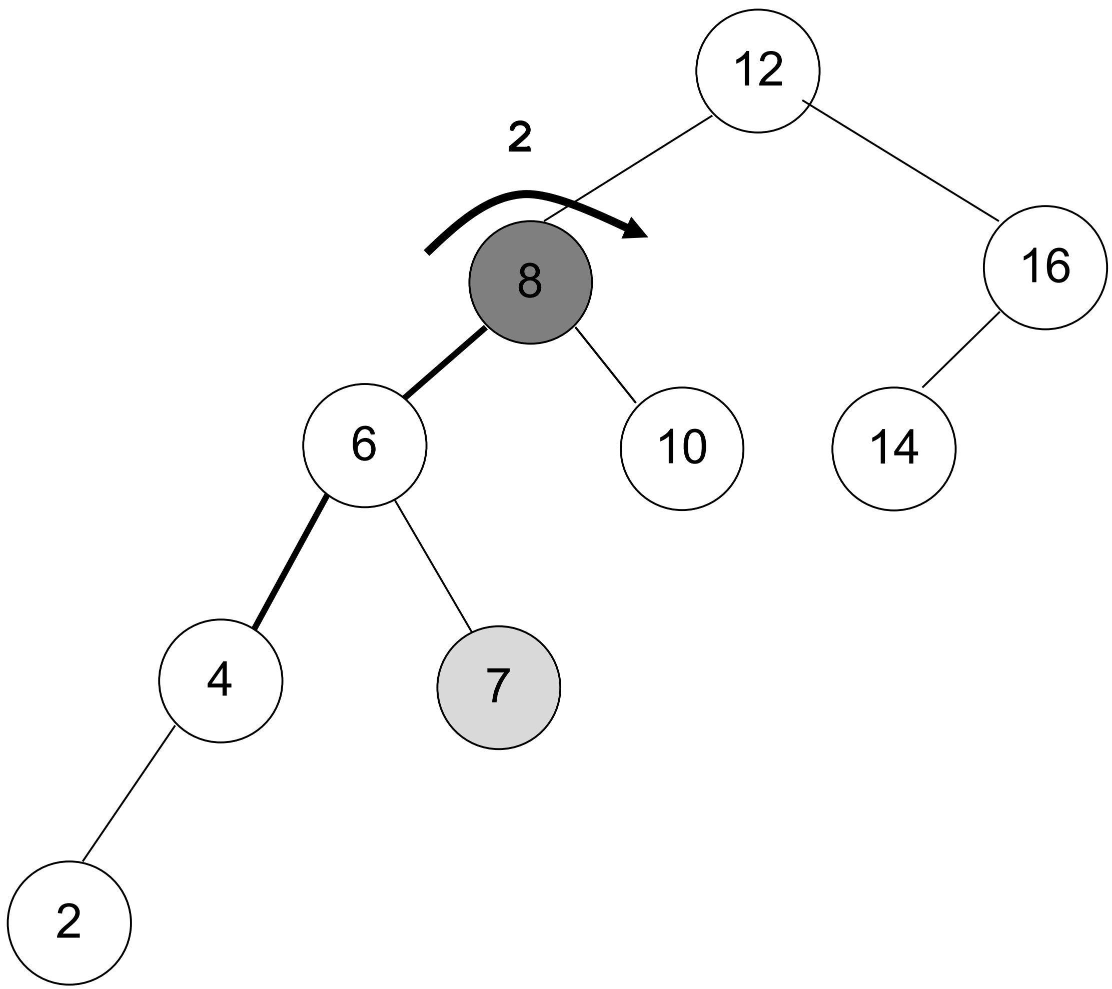
2. Exemple de rotation double



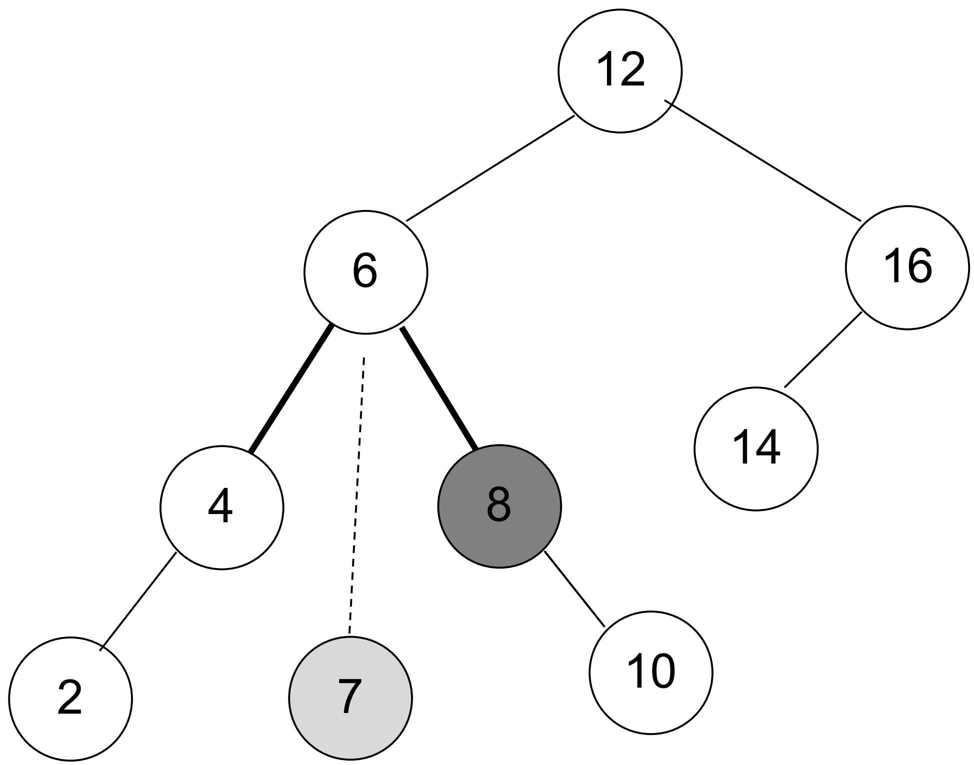
2. Exemple de rotation double



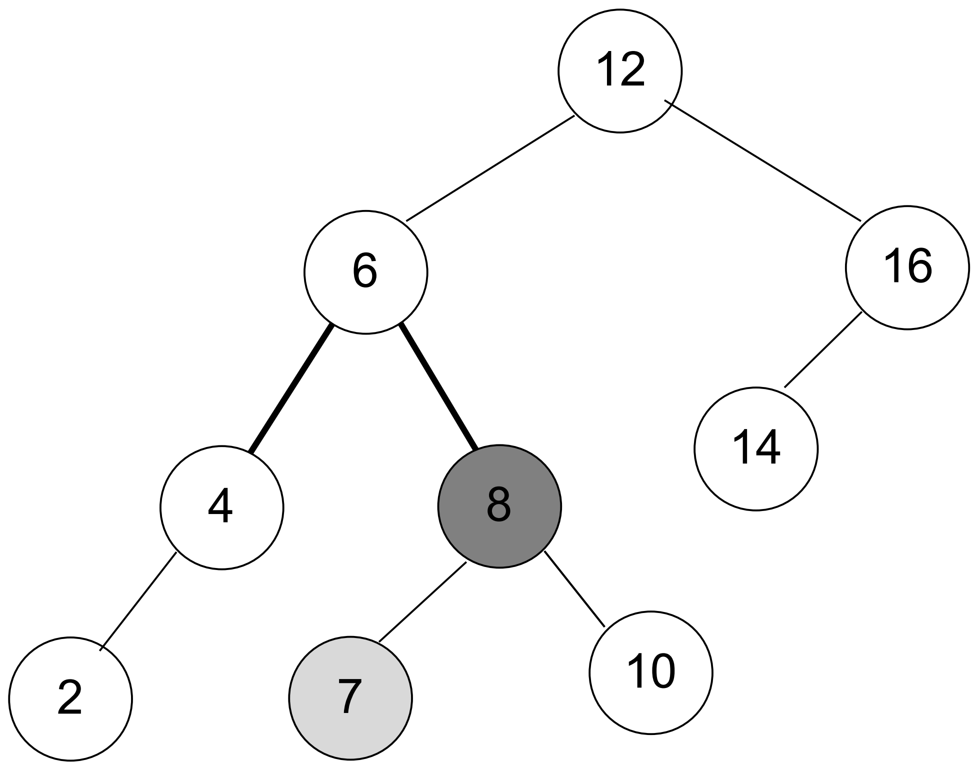
2. Exemple de rotation double



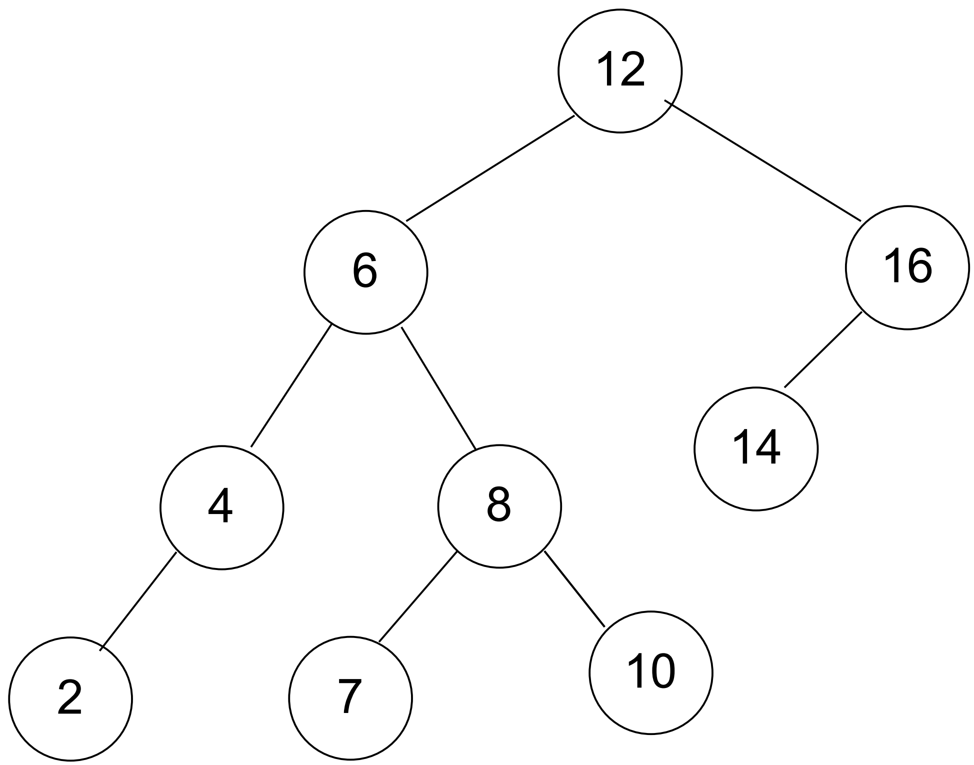
2. Exemple de rotation double



2. Exemple de rotation double



2. Exemple de rotation double



Arbres équilibrés

Arbre AVL

- Concepts de base de l’arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l’arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

AVL – implémentation

- Une fois le noeud inséré/retiré, en revenant sur notre chemin, il faut vérifier, pour chaque noeud parcouru, les différences de profondeur des sous-arbres de gauche et de droite
- La rotation est requise une seule fois sur le chemin qui mène de la racine au point d’insertion

Noeud AVL – implémentation

```
private static class AvlNode<AnyType>
{
    // Constructors
    AvlNode( AnyType theElement )
    {
        this( theElement, null, null );
    }

    AvlNode(AnyType theElement,
               BinaryNode<AnyType> lt, BinaryNode<AnyType> rt )
    {
        element  = theElement;
        left   = lt;
        right = rt;
        height = 0;
    }

    AnyType element;              // The data in the node
    BinaryNode<AnyType> left;     // Left child
    BinaryNode<AnyType> right;    // Right child
    int height;                   // Height
}

private int height( AvlNode<AnyType> t )
{
    return t == null ? -1 : t.height;
}
```

Arbre AVL – insertion

Old implementation (2nd edition)

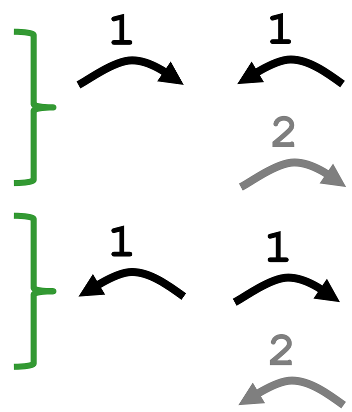
```
private AvlNode<AnyType> insert( AnyType x, AvlNode<AnyType> t )
{
   if( t == null )
      return new AvlNode<AnyType>( x, null, null );

   int compareResult = x.compareTo( t.element );

   if( compareResult < 0 )
   {
      t.left = insert( x, t.left );

      if( height( t.left ) - height( t.right ) == 2 )
          if( x.compareTo( t.left.element ) < 0 )
              t = rotateWithLeftChild( t ); // premier cas de figure
          else
              t = doubleWithLeftChild( t ); // second cas de figure
   }
   else if( compareResult > 0 )
   {
      t.right = insert( x, t.right );

      if( height( t.right ) - height( t.left ) == 2 )
          if( x.compareTo( t.right.element ) > 0 )
              t = rotateWithRightChild( t );  // premier cas de figure
          else
              t = doubleWithRightChild( t );  // second cas de figure
   }
   else
      ;  // Pas de doublons

   // Mettre à jour les hauteurs en remontant
   t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

   return t;
}
```

Arbre AVL – insertion

```
private AvlNode<AnyType> insert( AnyType x, AvlNode<AnyType> t )
{
     if( t == null )
         return new AvlNode<>( x, null, null );

     int compareResult = x.compareTo( t.element );

     if( compareResult < 0 )
         t.left = insert( x, t.left );
     else if( compareResult > 0 )
         t.right = insert( x, t.right );
     else
         ;  // Pas de doublons

     return balance( t );
}
```

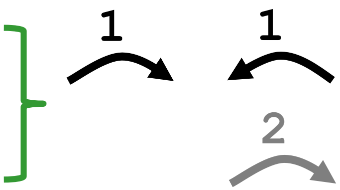
Arbre AVL – rotations

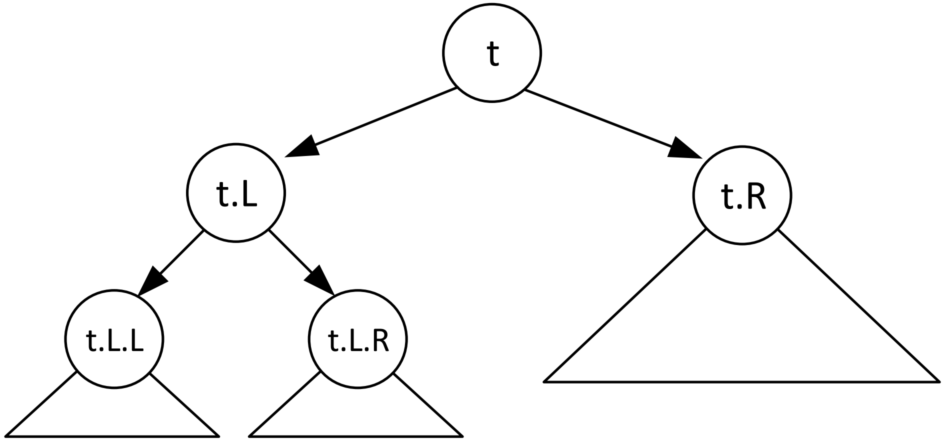
```java
private AvlNode<AnyType> balance( AvlNode<AnyType> t )
{
    if( t == null )
        return t;

    if( height( t.left ) - height( t.right ) > 1 )
    {
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 )
    {
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }

    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```



Arbre AVL – rotations

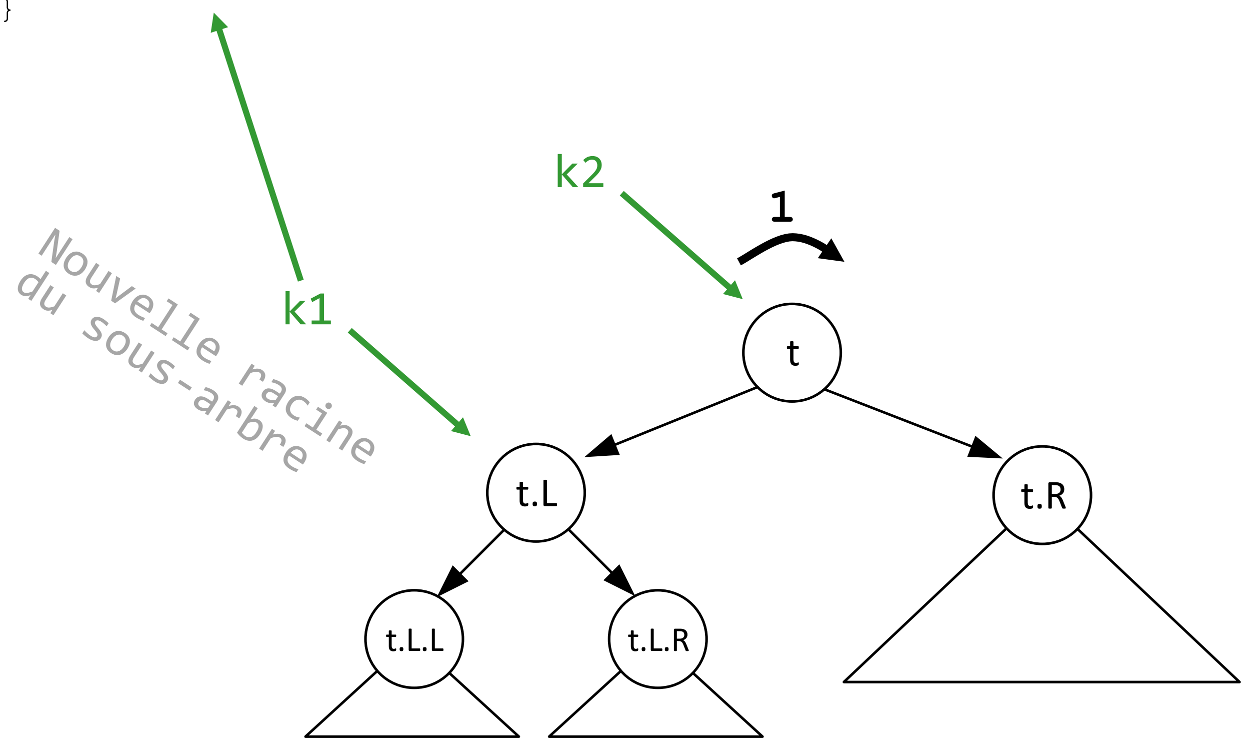
```java
private AvlNode<AnyType> balance( AvlNode<AnyType> t )
{
    if( t == null )
        return t;

    if( height( t.left ) - height( t.right ) > 1 )
    {
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 )
    {
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }

    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```





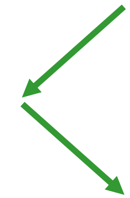
Arbre AVL – rotation simple

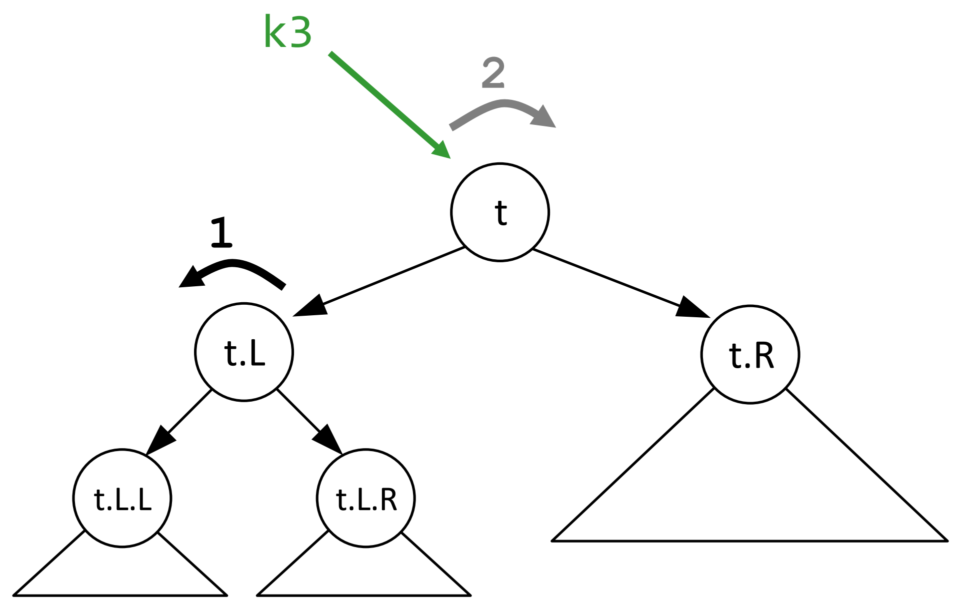
```
// Gauche - Gauche
private AvlNode<AnyType> rotateWithLeftChild( AvlNode<AnyType> k2 )
{
   AvlNode<AnyType> k1 = k2.left;

   k2.left = k1.right;
   k1.right = k2;

   // Mettre à jour les hauteurs à la rotation
   k2.height = Math.max( height( k2.left ), height( k2.right ) ) + 1;
   k1.height = Math.max( height( k1.left ), k2.height ) + 1;

   return k1;
}
```



Arbre AVL – rotation double

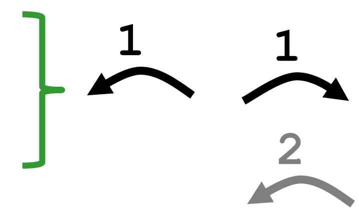
```java
// Gauche - Droite
private AvlNode<AnyType> doubleWithLeftChild( AvlNode<AnyType> k3 )
{
   k3.left = rotateWithRightChild( k3.left );
   return rotateWithLeftChild( k3 );
}
```

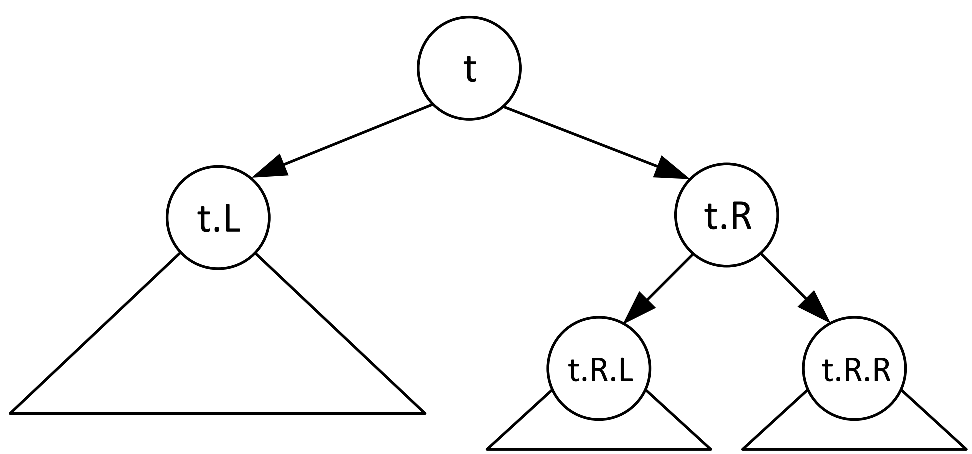




Arbre AVL – insertion

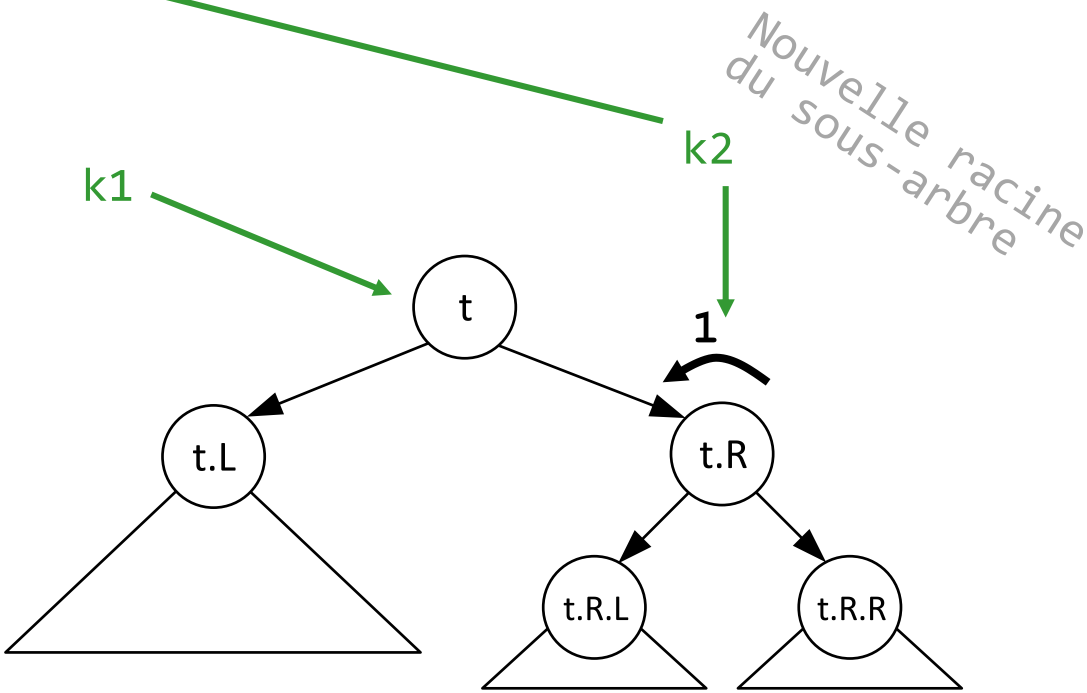
```
private AvlNode<AnyType> balance( AvlNode<AnyType> t )
{
    if( t == null )
        return t;

    if( height( t.left ) - height( t.right ) > 1 )
    {
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 )
    {
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }

    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```





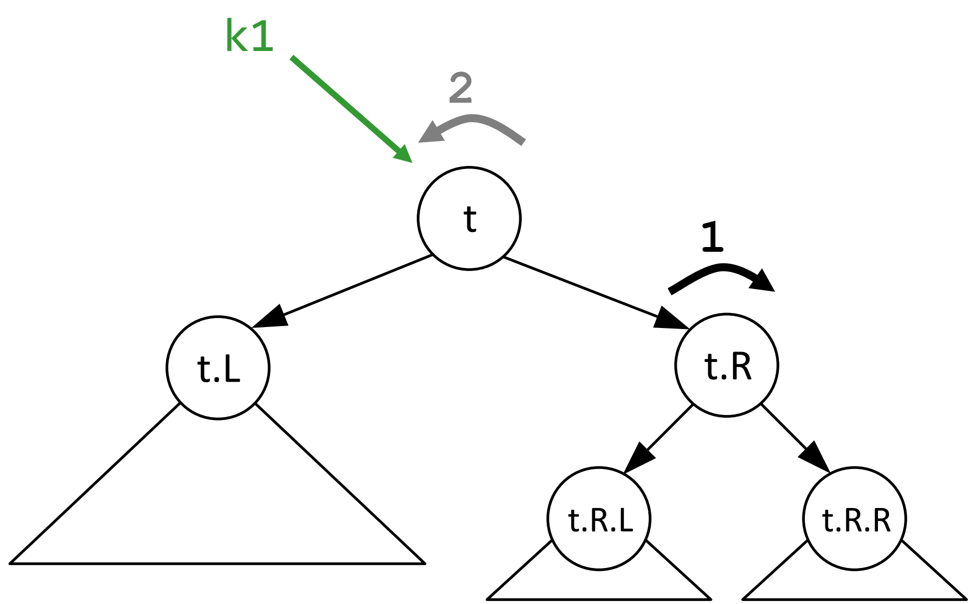
Arbre AVL – rotations simples

```
// Droite - Droite
private AvlNode<AnyType> rotateWithRightChild( AvlNode<AnyType> k1 )
{
   AvlNode<AnyType> k2 = k1.right;

   k1.right = k2.left;
   k2.left = k1;

   // Mettre à jour les hauteurs à la rotation
   k1.height = Math.max( height( k1.left ), height( k1.right ) ) + 1;
   k2.height = Math.max( height( k2.right ), k1.height ) + 1;

   return k2;
}
```



Arbre AVL – rotation double

```
// Droite - Gauche
private AvlNode<AnyType> doubleWithRightChild( AvlNode<AnyType> k1 )
{
   k1.right = rotateWithLeftChild( k1.right );
   return rotateWithRightChild( k1 );
}
```



Arbre AVL – removal

```
private AvlNode<AnyType> remove( AnyType x, AvlNode<AnyType> t )
{
    if( t == null )
        return t;   // Item not found; do nothing

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) // Two children
    {
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;

    return balance( t );
}
```

Arbres équilibrés

Arbre AVL

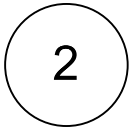
- Concepts de base de l'arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l'arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

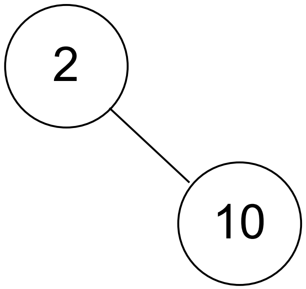
AVL – exemple détaillé

2 10 12 4 16 8 6 14



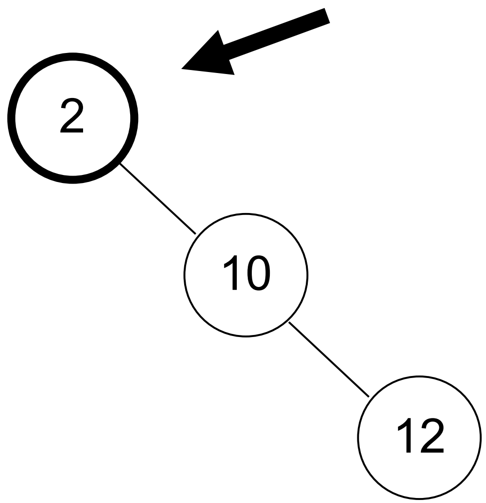
AVL – exemple détaillé

2 10 12 4 16 8 6 14



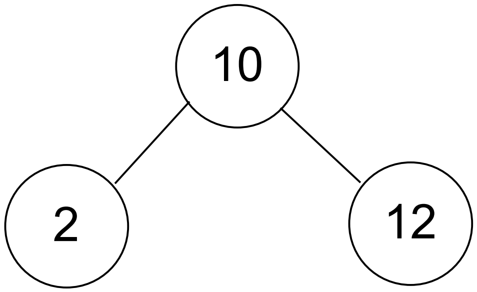
AVL – exemple détaillé

2 10 12 4 16 8 6 14



AVL – exemple détaillé

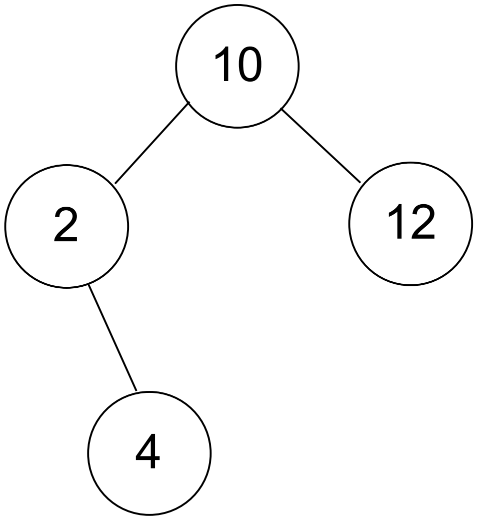
2 10 12 4 16 8 6 14



Rotation simple

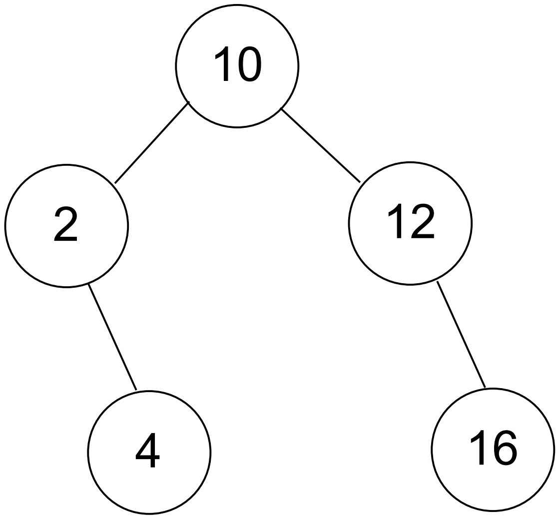
AVL – exemple détaillé

2 10 12 4 16 8 6 14



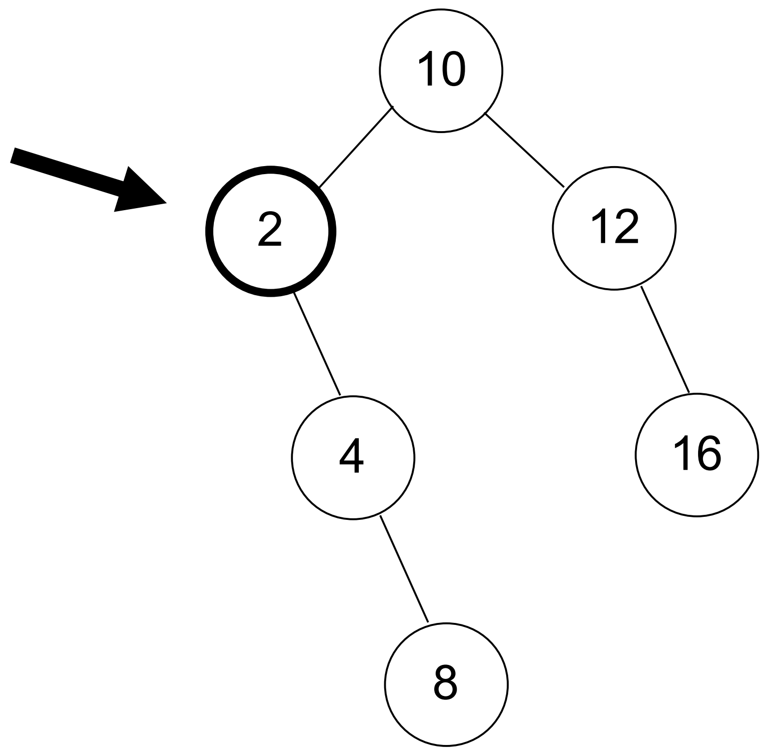
AVL – exemple détaillé

2 10 12 4 16 8 6 14



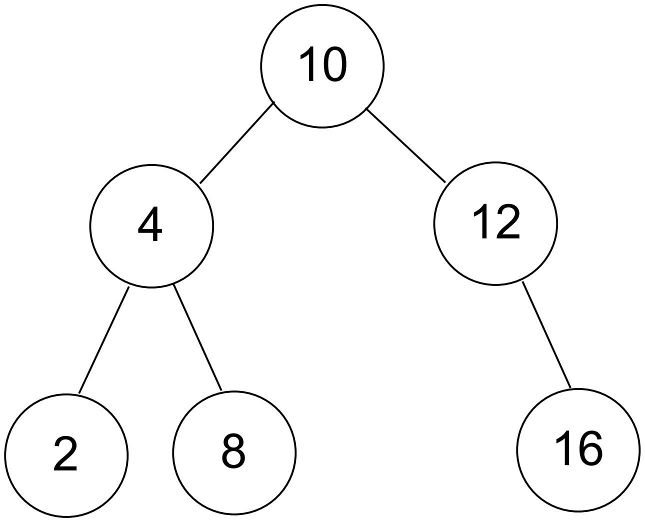
AVL – exemple détaillé

2 10 12 4 16 8 6 14



AVL – exemple détaillé

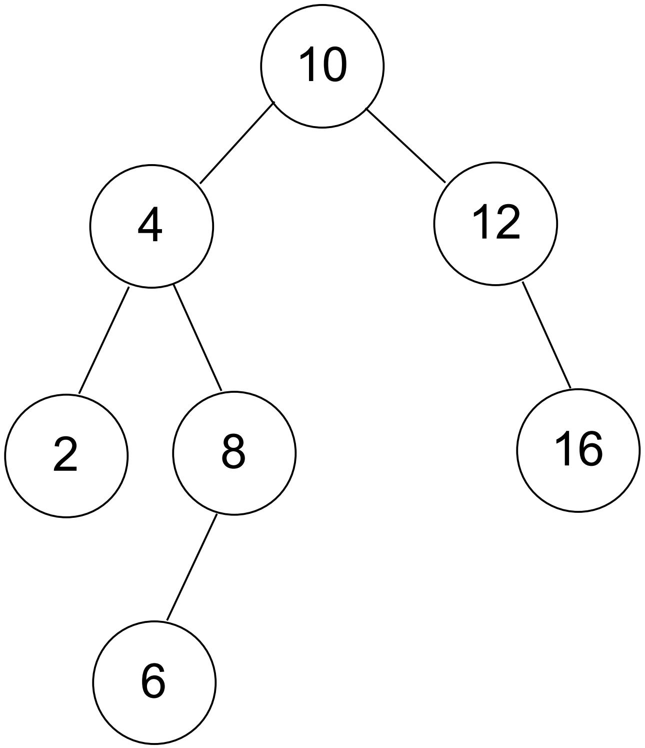
2 10 12 4 16 8 6 14



Rotation simple

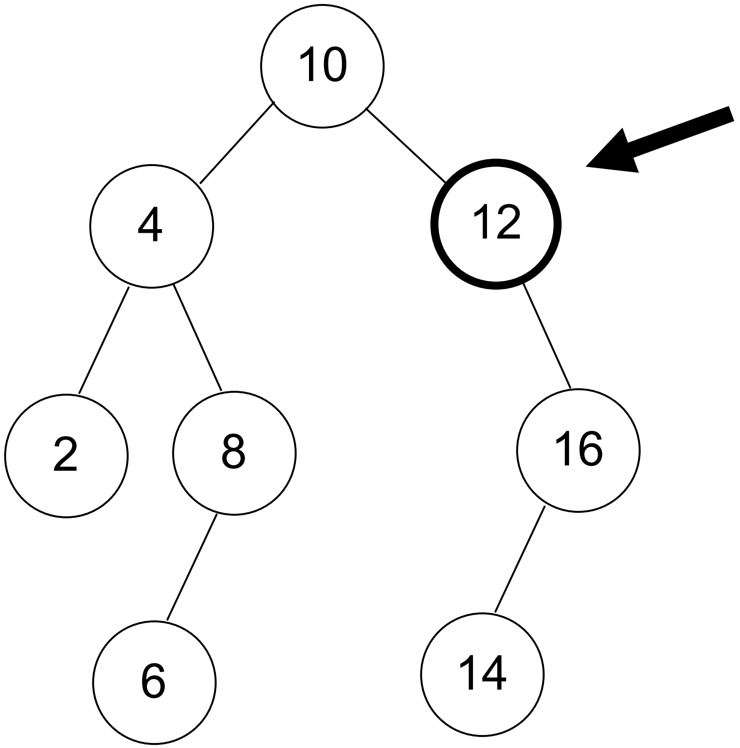
AVL – exemple détaillé

2 10 12 4 16 8 6 14



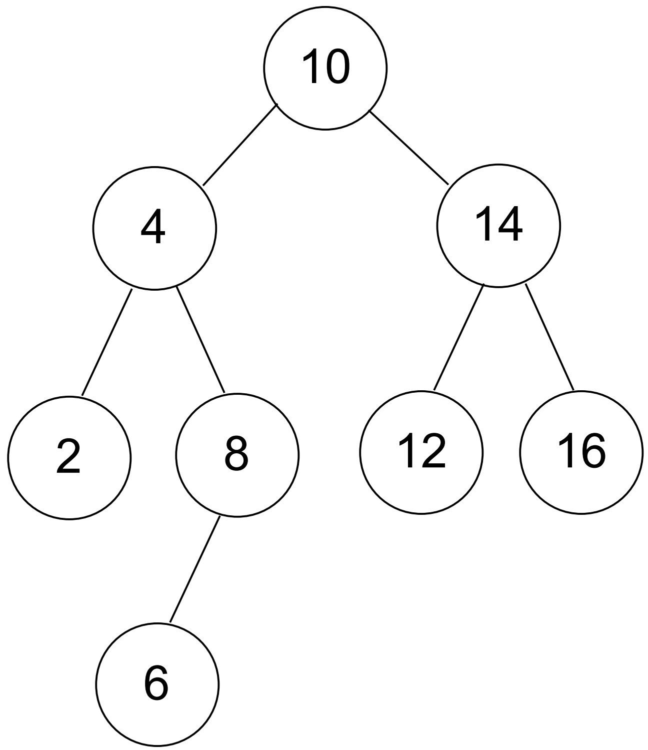
AVL – exemple détaillé

2 10 12 4 16 8 6 14



AVL – exemple détaillé

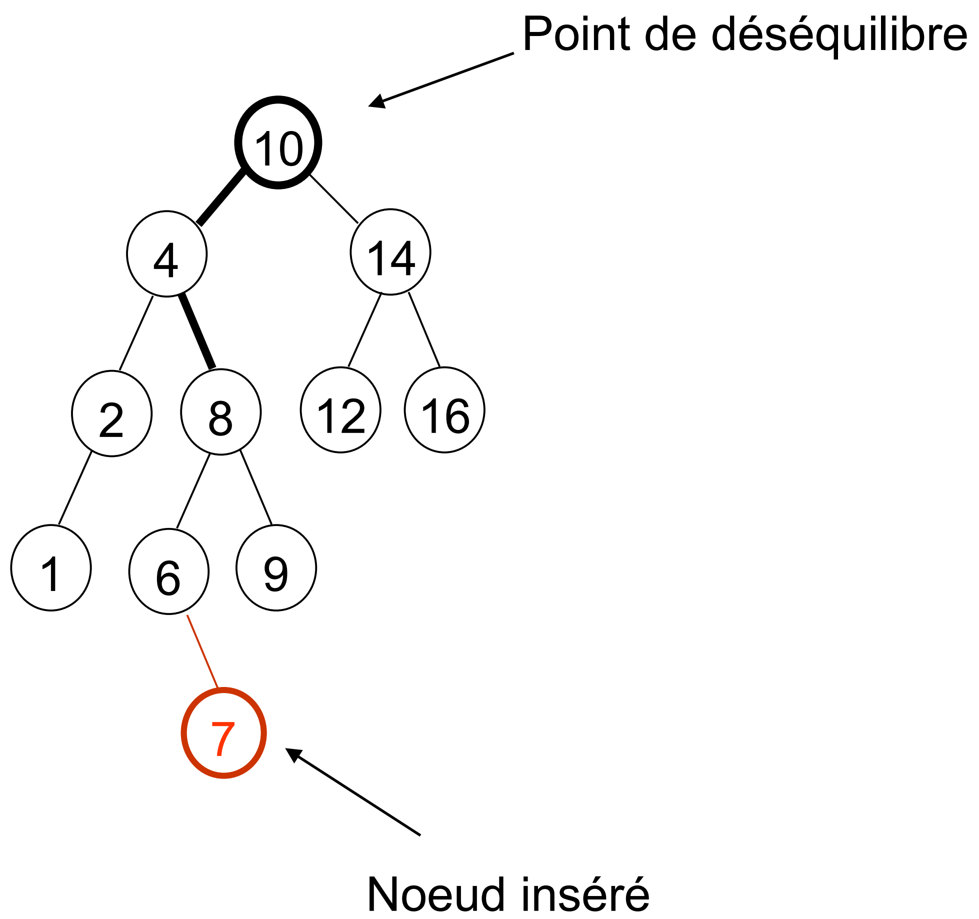
2 10 12 4 16 8 6 14



Rotation double

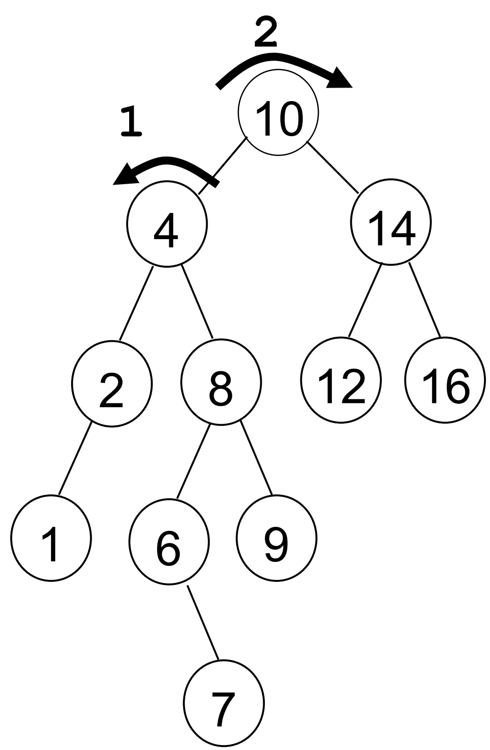
AVL –exemple

Lorsque la rotation se fait loin du point d'insertion…



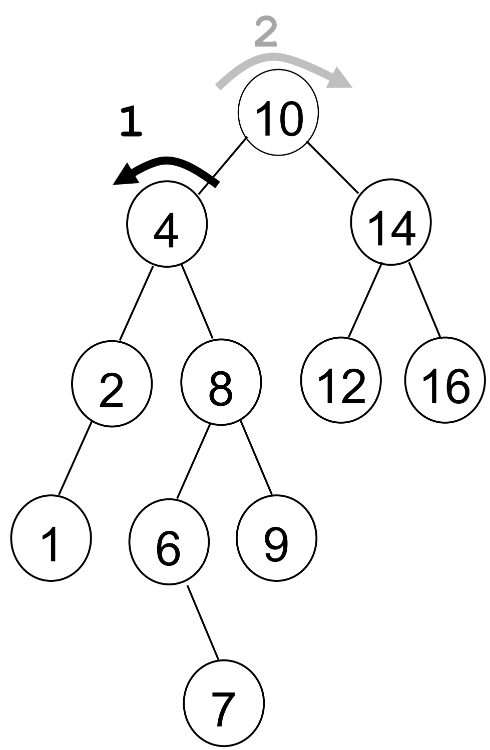
AVL –exemple

Lorsque la rotation se fait loin du point d'insertion…

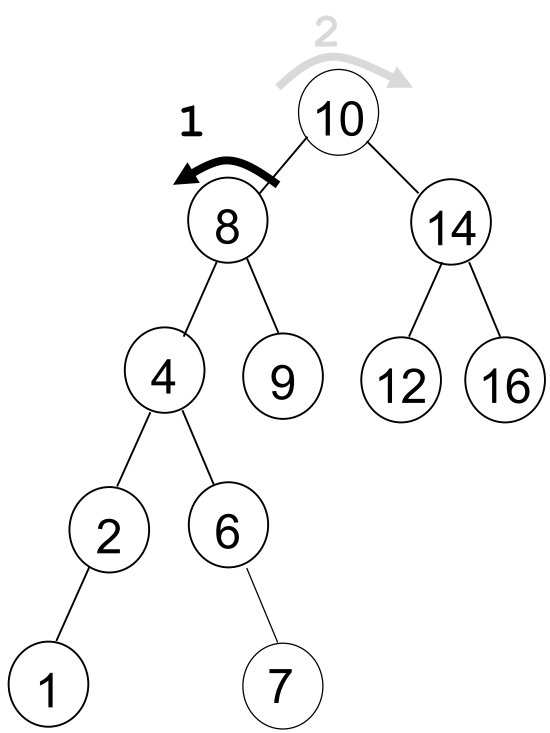


AVL –exemple

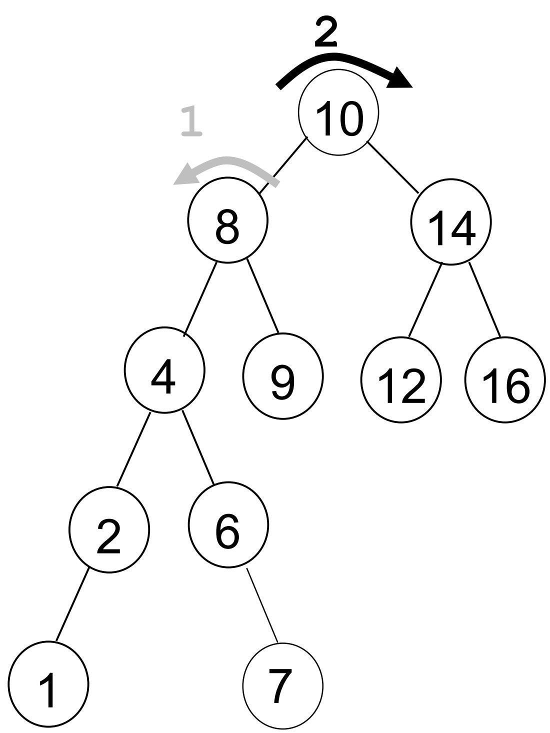
Lorsque la rotation se fait loin du point d'insertion…



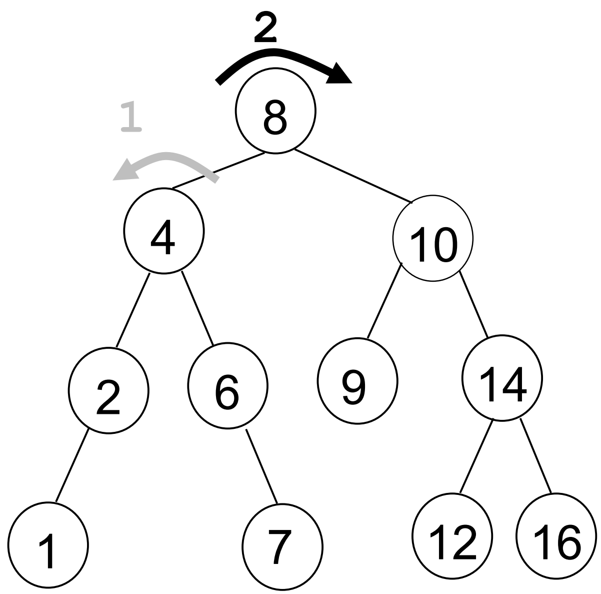
AVL –exemple



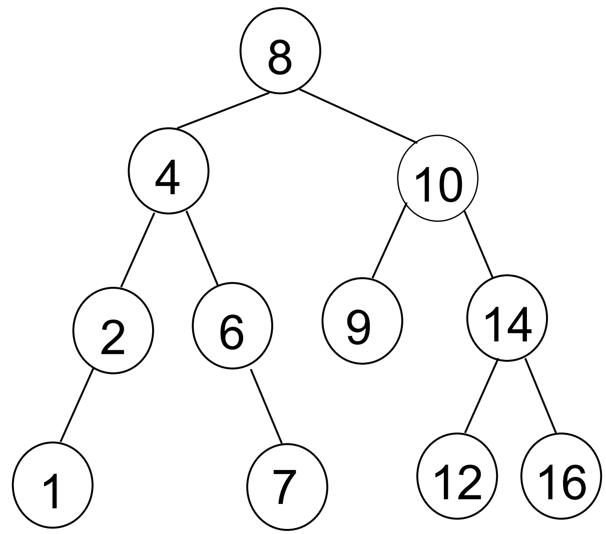
AVL –exemple



AVL –exemple

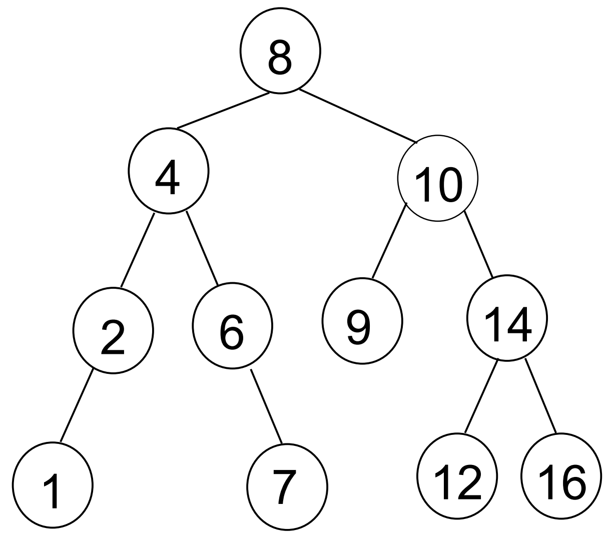


AVL –exemple



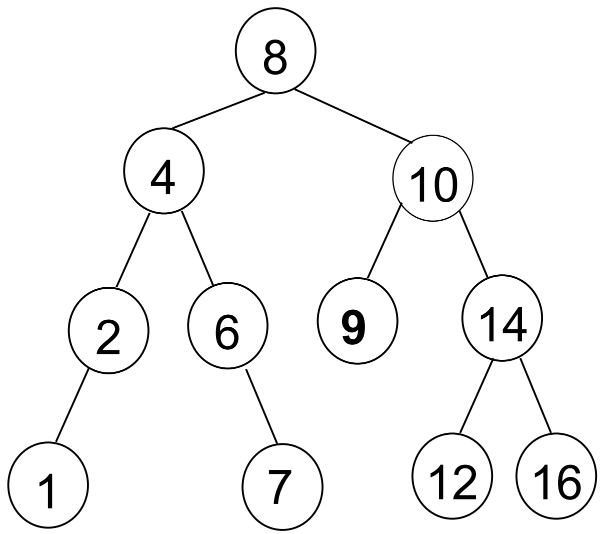
AVL –exemple

remove(9): start



AVL –exemple

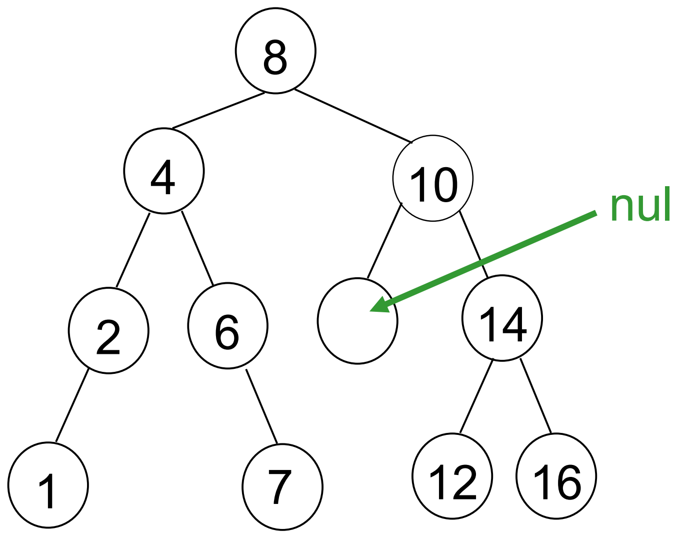
remove(9)




```
private AvlNode<AnyType> remove( AnyType x, AvlNode<AnyType> t ){
    if( t == null )
        return t;   // Item not found; do nothing

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) { // Two children
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;

    return balance( t );
}
```

AVL –exemple

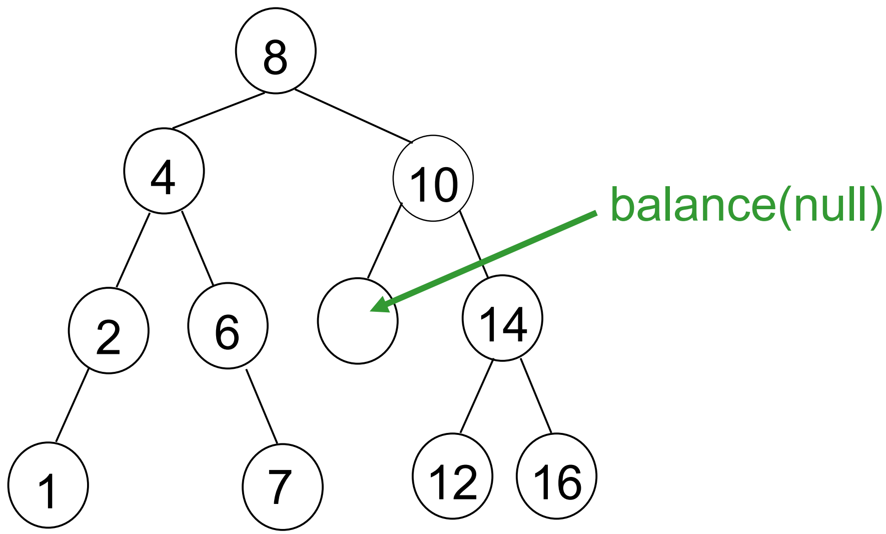
remove(9)




```
private AvlNode<AnyType> remove( AnyType x, AvlNode<AnyType> t ){
    if( t == null )
        return t;   // Item not found; do nothing

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) { // Two children
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;

    return balance( t );
}
```

AVL –exemple

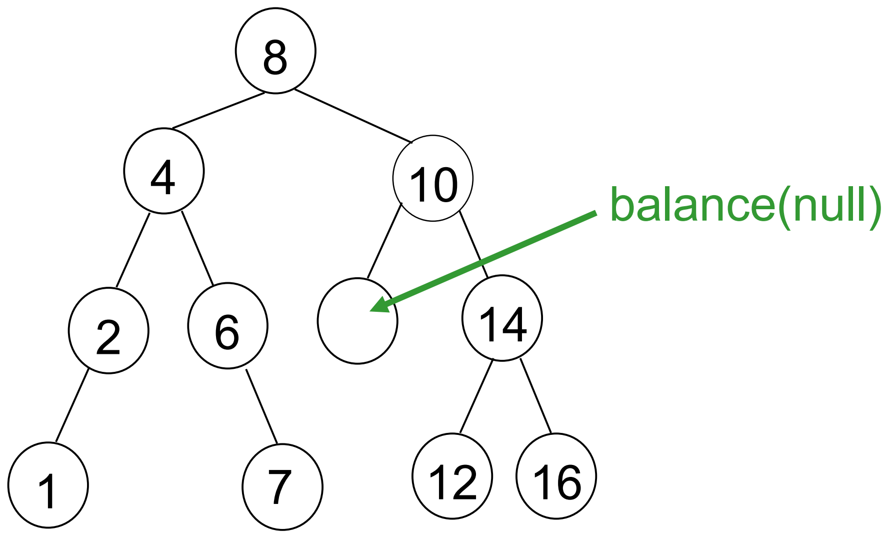
remove(9)




```
private AvlNode<AnyType> remove( AnyType x, AvlNode<AnyType> t ){
    if( t == null )
        return t;   // Item not found; do nothing

    int compareResult = x.compareTo( t.element );

    if( compareResult < 0 )
        t.left = remove( x, t.left );
    else if( compareResult > 0 )
        t.right = remove( x, t.right );
    else if( t.left != null && t.right != null ) { // Two children
        t.element = findMin( t.right ).element;
        t.right = remove( t.element, t.right );
    }
    else
        t = ( t.left != null ) ? t.left : t.right;

    return balance( t );   ← null
}
```

AVL –exemple

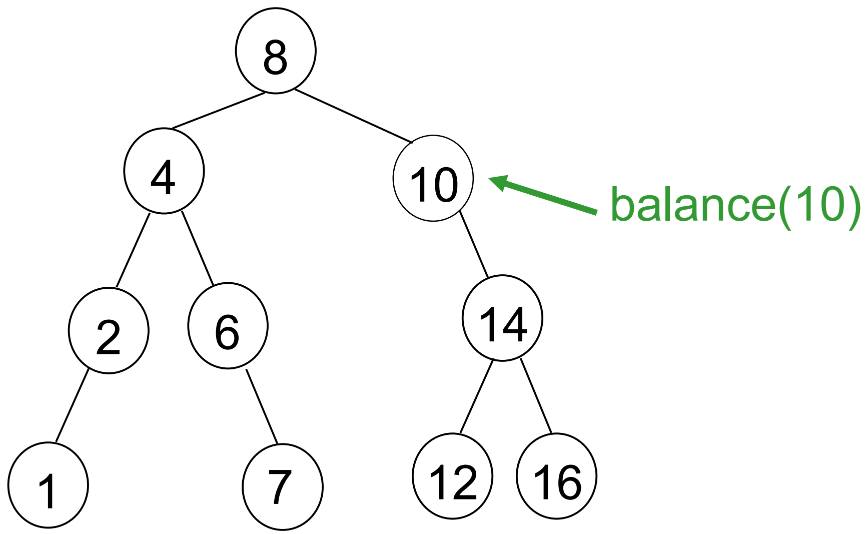
remove(9)




```
private AvlNode<AnyType> balance( AvlNode<AnyType> t ){
    if( t == null )return t;        null

    if( height( t.left ) - height( t.right ) > 1 ){
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 ){
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }
    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```

AVL –exemple

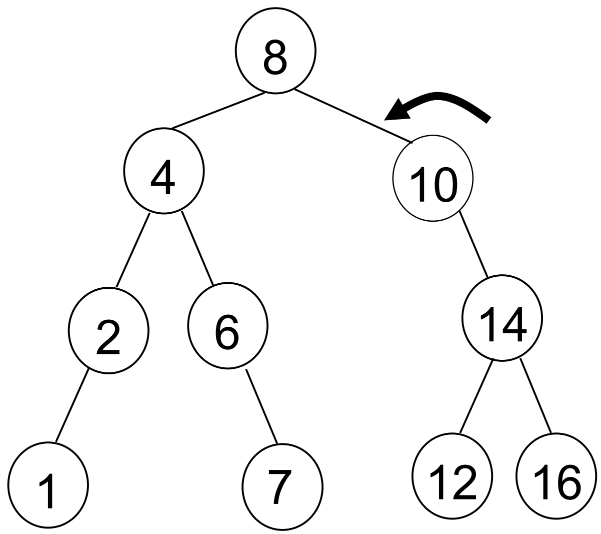
remove(9)




```
private AvlNode<AnyType> balance( AvlNode<AnyType> t ){
    if( t == null )return t;

    if( height( t.left ) - height( t.right ) > 1 ){
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 ){
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }
    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```

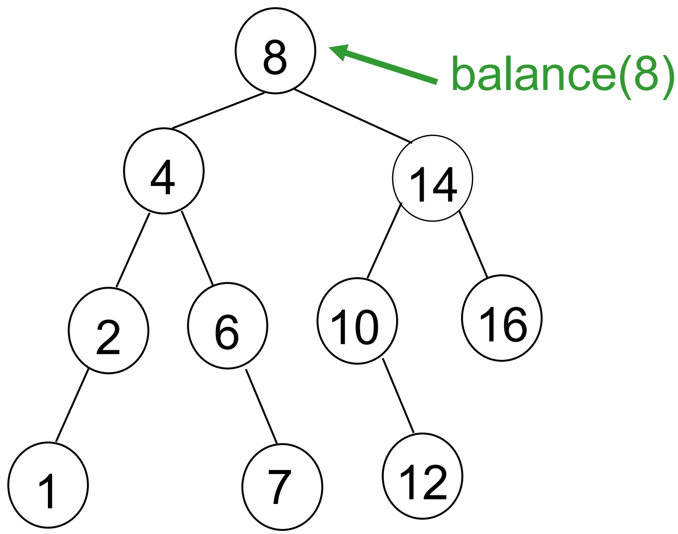
AVL –exemple

remove(9)



AVL –exemple

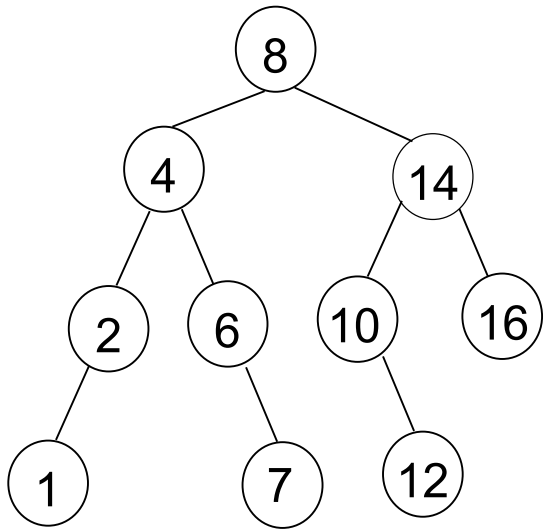
remove(9)




```
private AvlNode<AnyType> balance( AvlNode<AnyType> t ){
    if( t == null )return t;

    if( height( t.left ) - height( t.right ) > 1 ){
        if( height( t.left.left ) >= height( t.left.right ) )
            t = rotateWithLeftChild( t );
        else
            t = doubleWithLeftChild( t );
    }
    else if( height( t.right ) - height( t.left ) > 1 ){
        if( height( t.right.right ) >= height( t.right.left ) )
            t = rotateWithRightChild( t );
        else
            t = doubleWithRightChild( t );
    }
    t.height = Math.max( height( t.left ), height( t.right ) ) + 1;

    return t;
}
```

AVL –exemple

remove(9): end



Arbres équilibrés

Arbre AVL

- Concepts de base de l'arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l'arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

Arbres Splay

Motivation:

- Situation idéale visée: s’assurer que le sous-arbre de gauche et le sous-arbre de droite sont de même hauteur
- Situation plus réaliste (AVL): s’assurer que le sous-arbre de gauche et le sous-arbre de droite ont une hauteur proche à une unité près
  - Ce principe s’appliqueà tous les noeuds de manière récursive
  - Quand on applique ceci à chaque insertion ou retrait, cela peut devenir coûteux
- Il faut donc établir des conditions plus faibles, mais qui nous assurent des gains en performance

Arbres Splay

- **Définition**: arbre de recherche binaire tel que chaque séquence de M opérations consécutives effectuées sur un arbre vide est de complexité de O(M log N) où N est le nombre d'éléments dans l'arbre à la fin des opération.

- Malgré cette garantie, une opération individuelle peut être de complexité O(N). Cependant, avoir une complexité (O(M log N)) pour M opération consécutives garantit une complexité amortie par opération de O(log N).

Arbres Splay

- **Idée de base:**
  - Après un accès (get) à un noeud **A**, l'arbre est restructuré (splayed) en déplaçant **A** vers la racine de l'arbre, via une série de rotations, pour en devenir la nouvelle racine.
  - Cette restructuration a pour effet de réduire la profondeur de tous les noeuds se trouvant sur le chemin menant de la racine vers **A.**

Arbres équilibrés

Arbre AVL

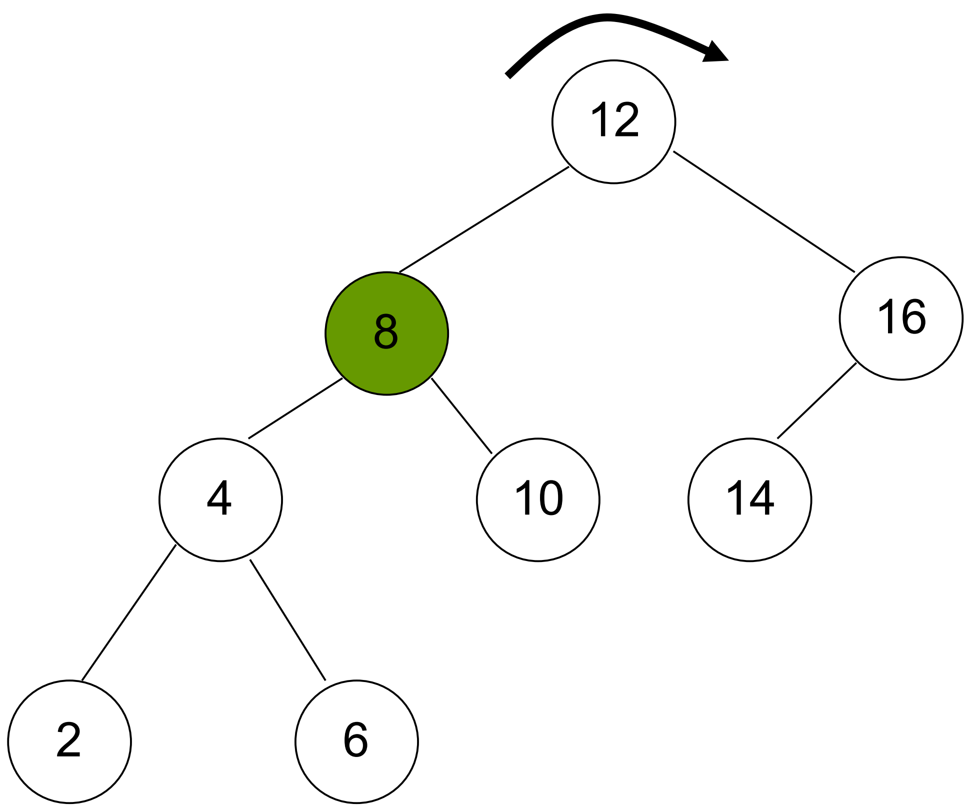
- Concepts de base de l'arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l'arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

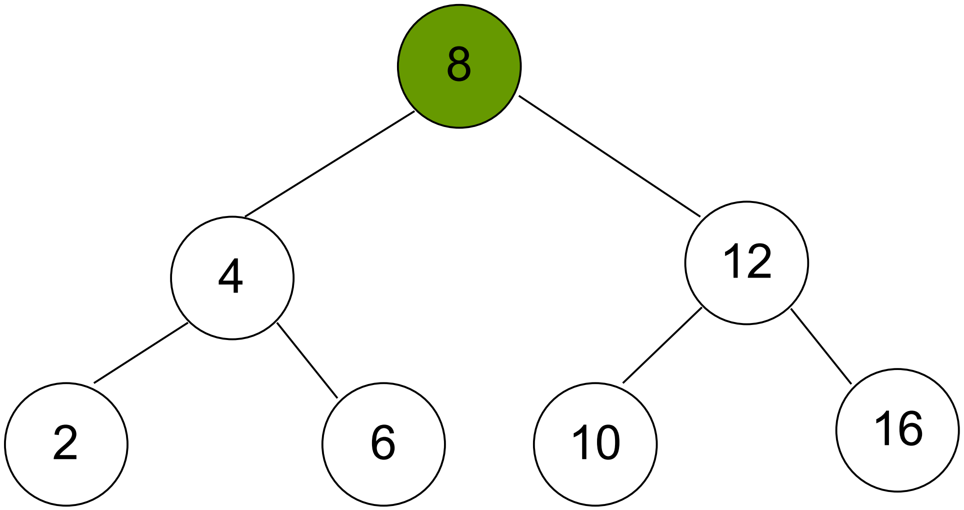
Splaying : Cas de figure 1

1. On effectue get(8) : le noeud à remonter n'a pas de grand parent:



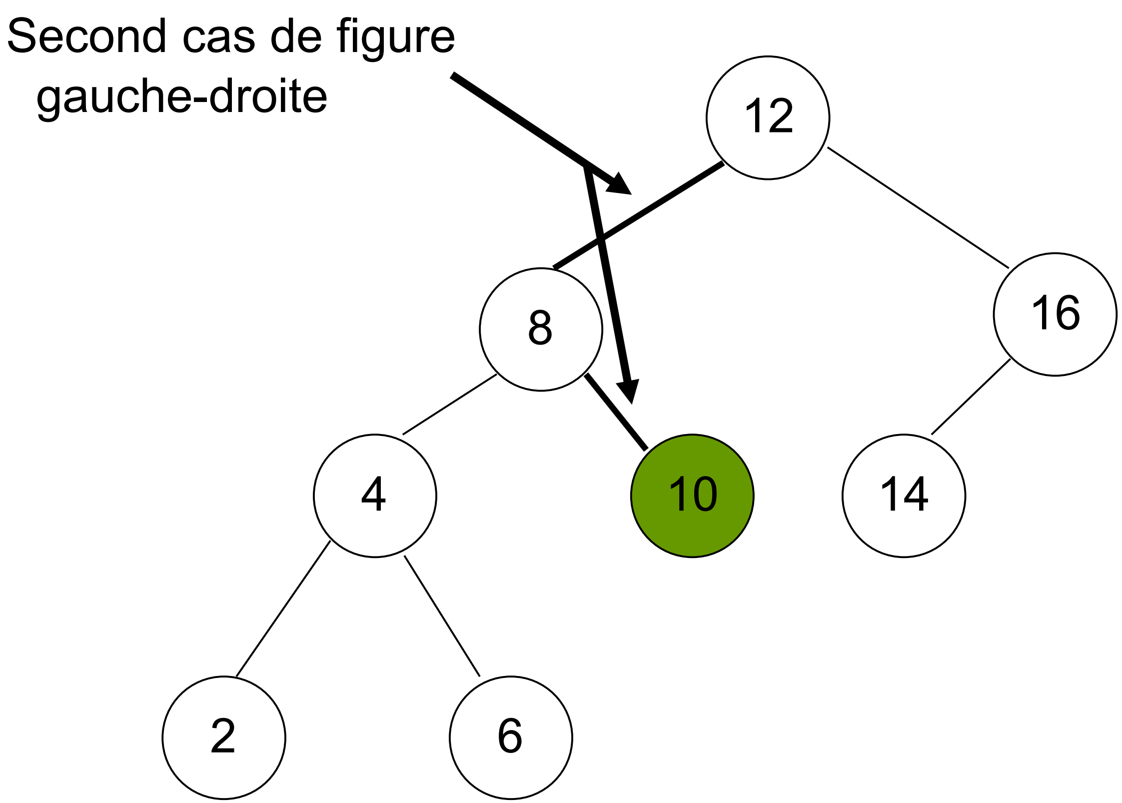
Splaying : Cas de figure 1

1. On effectue une rotation simple



Splaying : Cas de figure 2

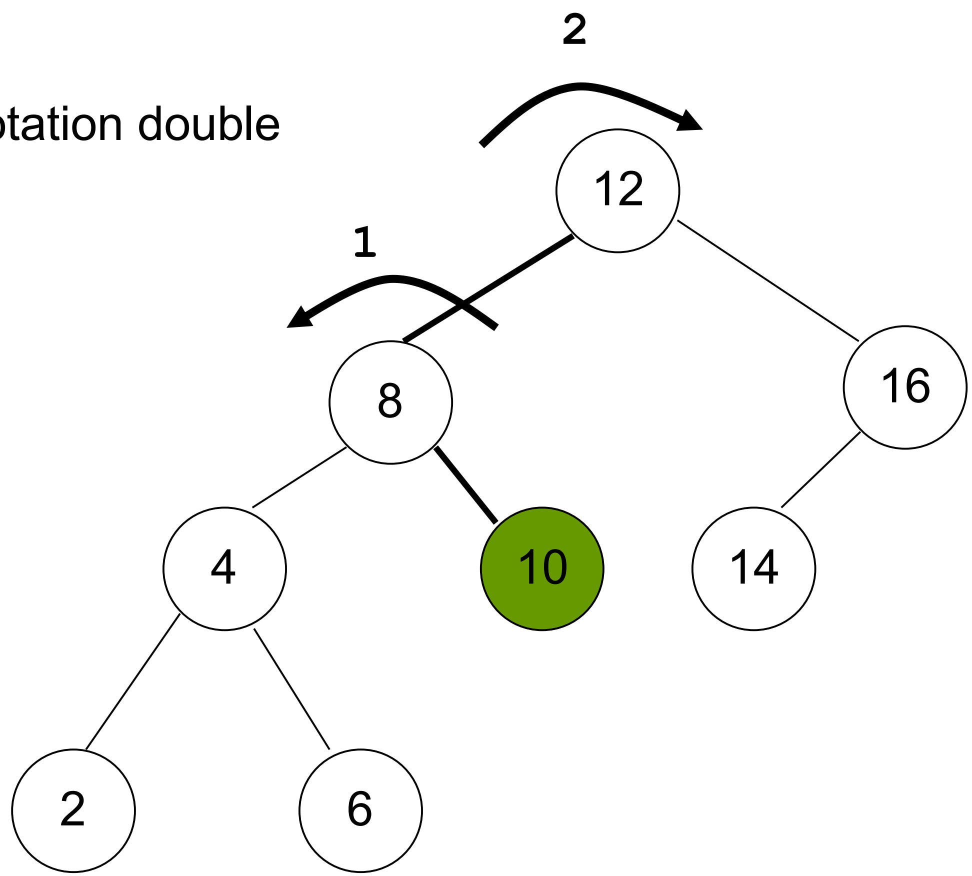
2. On effectue get(10) : le noeud à remonter a un grand parent et le chemin vers celui-ci n'est pas droit:



Splaying : Cas de figure 2

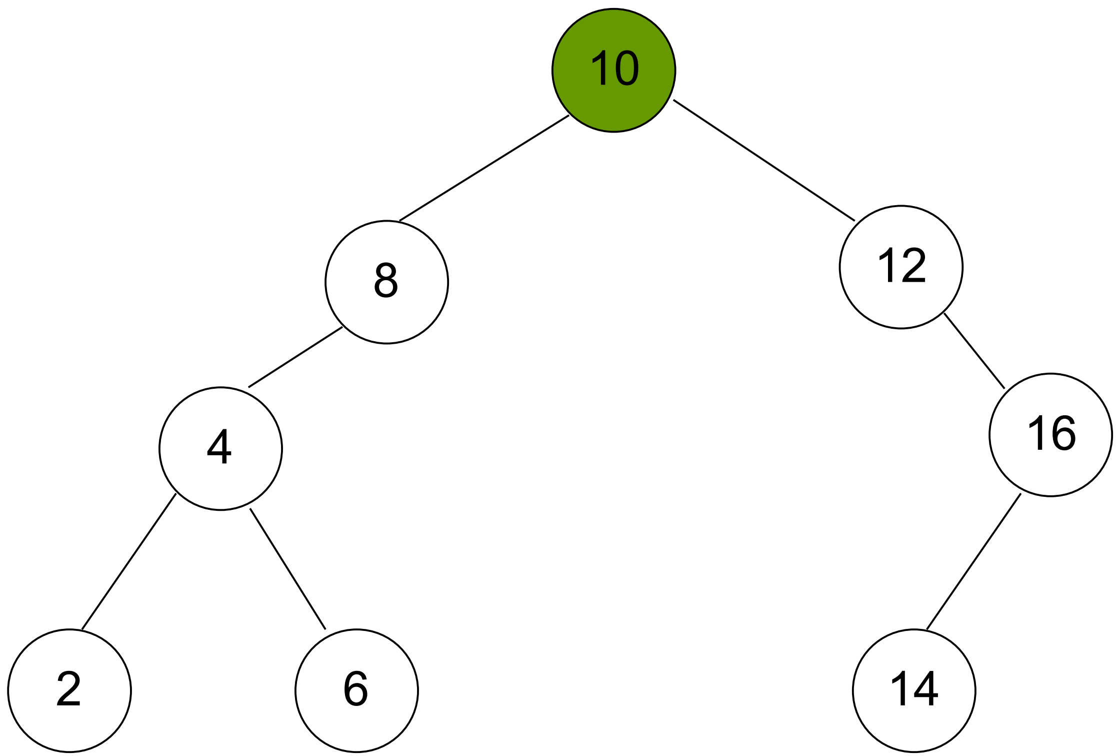
2. Le cas de figure s'appelle zig-zag: on effectue une rotation double

Rotation double



Splaying : Cas de figure 2

2. À la fin du zig-zag, on obtient ceci:

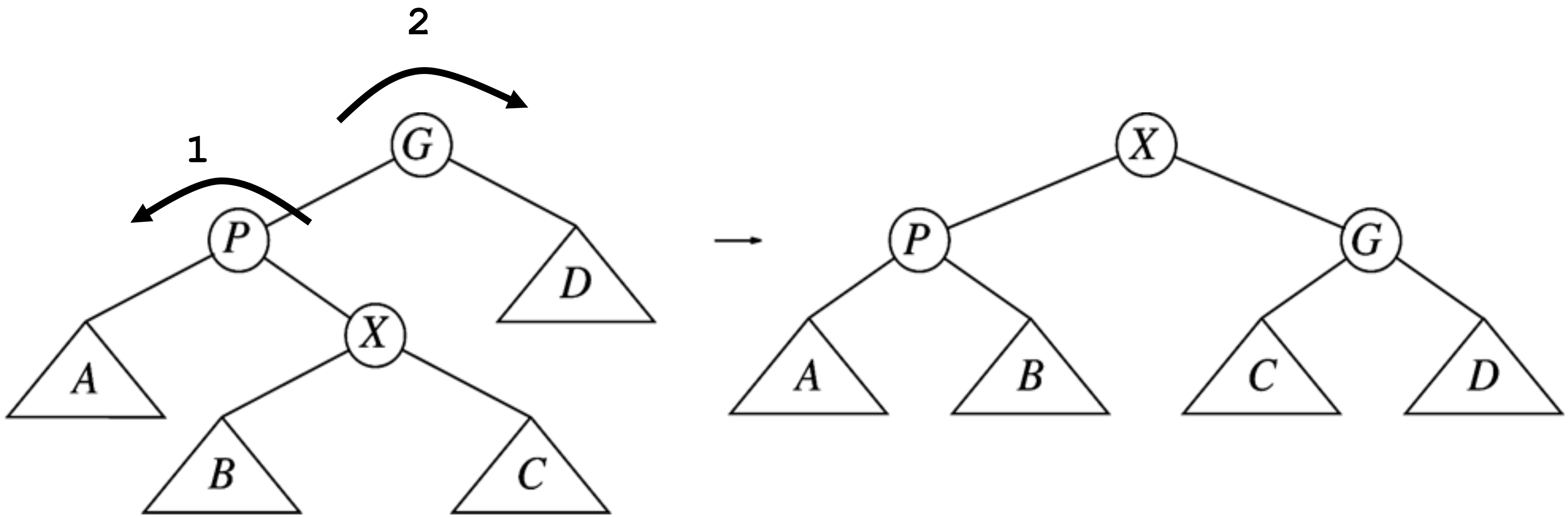


Splaying - Zig-zag

2. Représentation générale du zig-zag:

Double rotation gauche-droite après get(X).

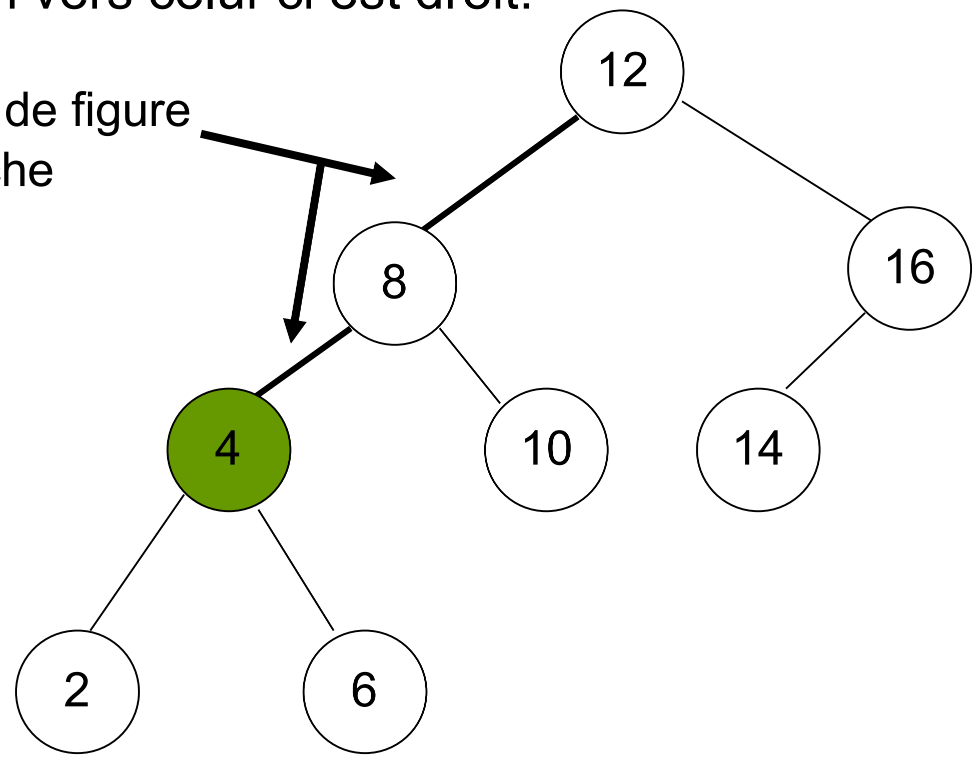
P est parent de X, G est grand-parent de X



Splaying : Cas de figure 3

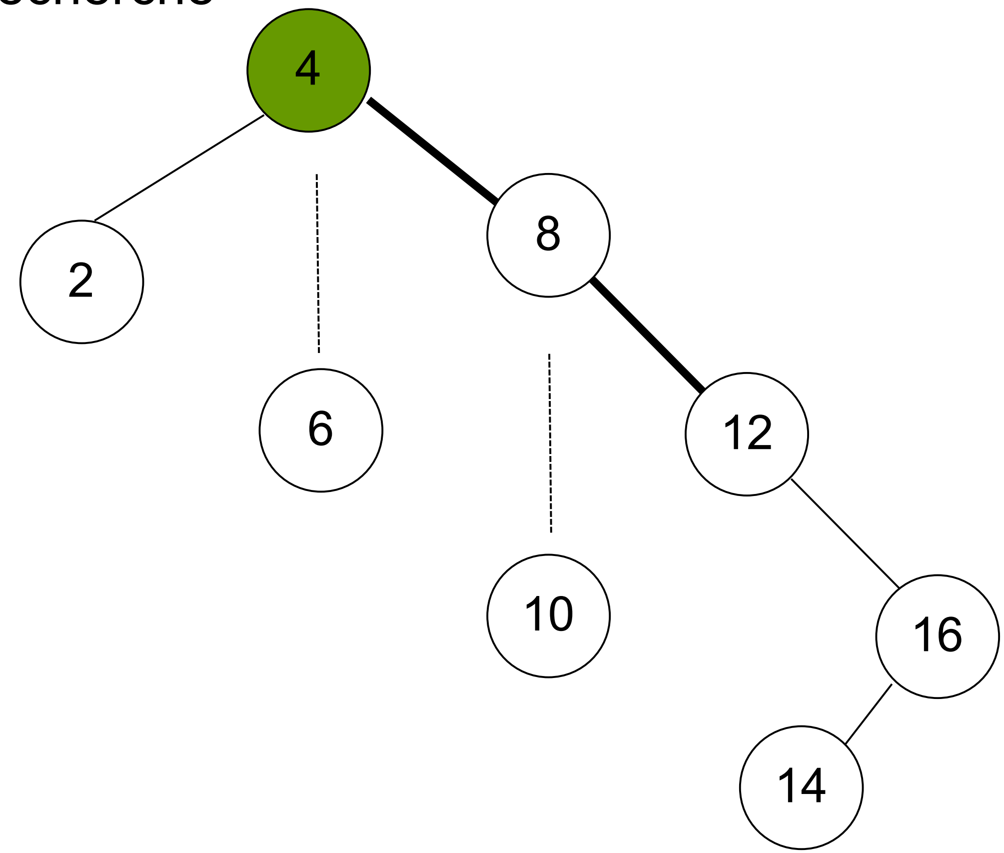
3. On effectue get(4) : le noeud à remonter a un grand parent et le chemin vers celui-ci est droit:

Troisième cas de figure
gauche-gauche



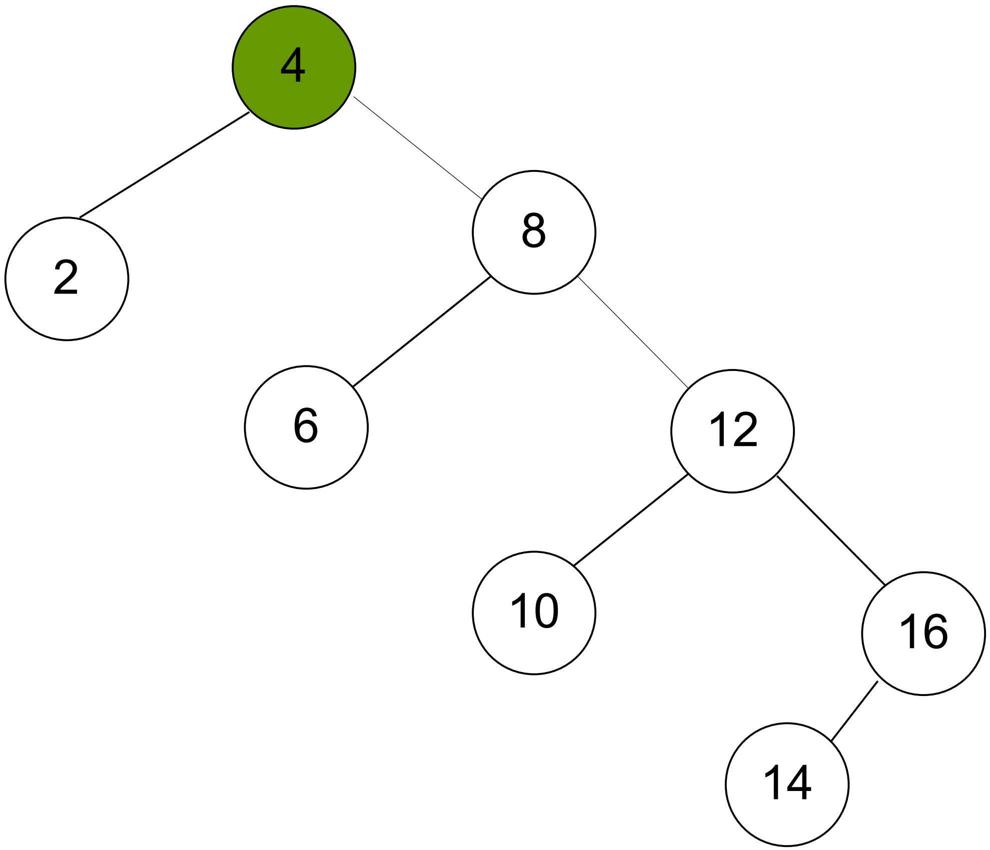
Splaying : Cas de figure 3

3. Ce cas de figure s’appelle zig-zig: on réaligne les noeuds de sorte à remonter le noeud recherché



Splaying : Cas de figure 3

3. À la fin du zig-zig, on obtient ceci:

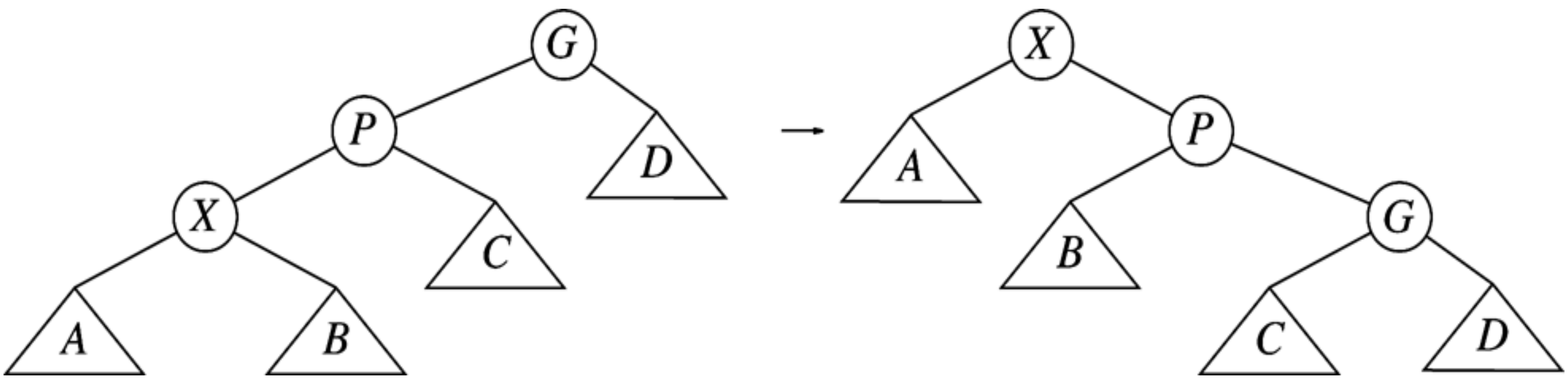


Splaying - Zig-zig

2. Représentation générale du zig-zig:

Réalignement après get(X).

P est parent de X, G est grand-parent de X



Arbres équilibrés

Arbre AVL

- Concepts de base de l'arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

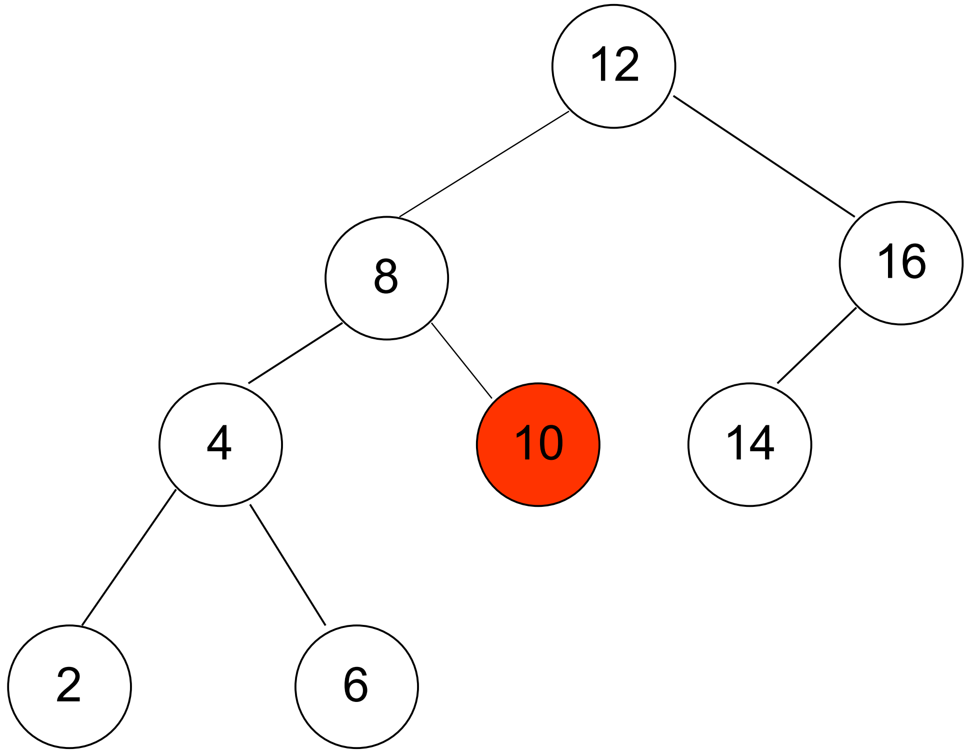
Arbre Splay

- Concepts de base de l'arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

Splaying : remove

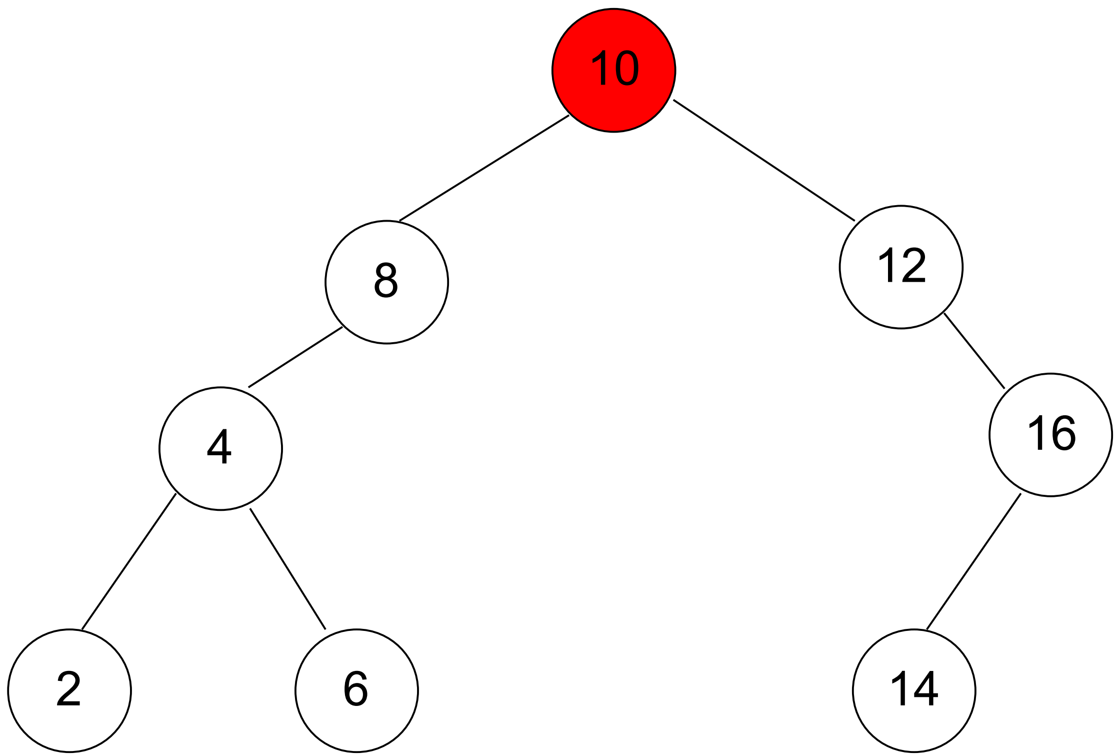
Pour effectuer un delete, il faut procéder à un get, puis enlever le noeud

Exemple: delete(10) …



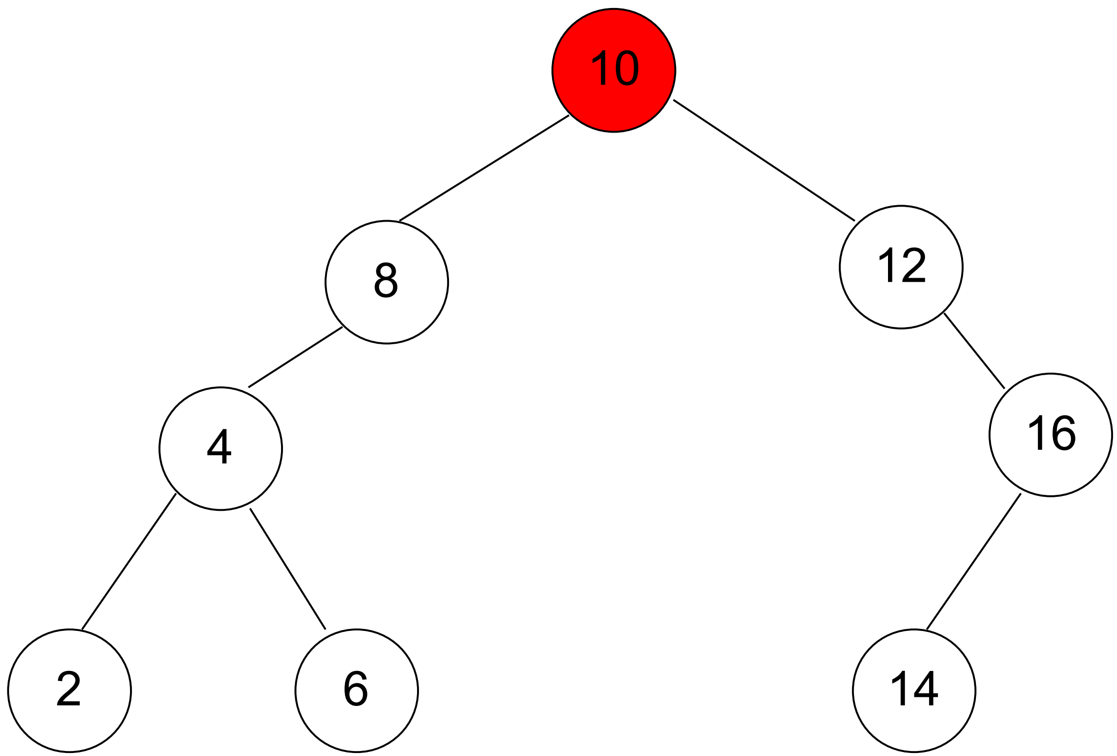
Splaying : remove

1. get(10)



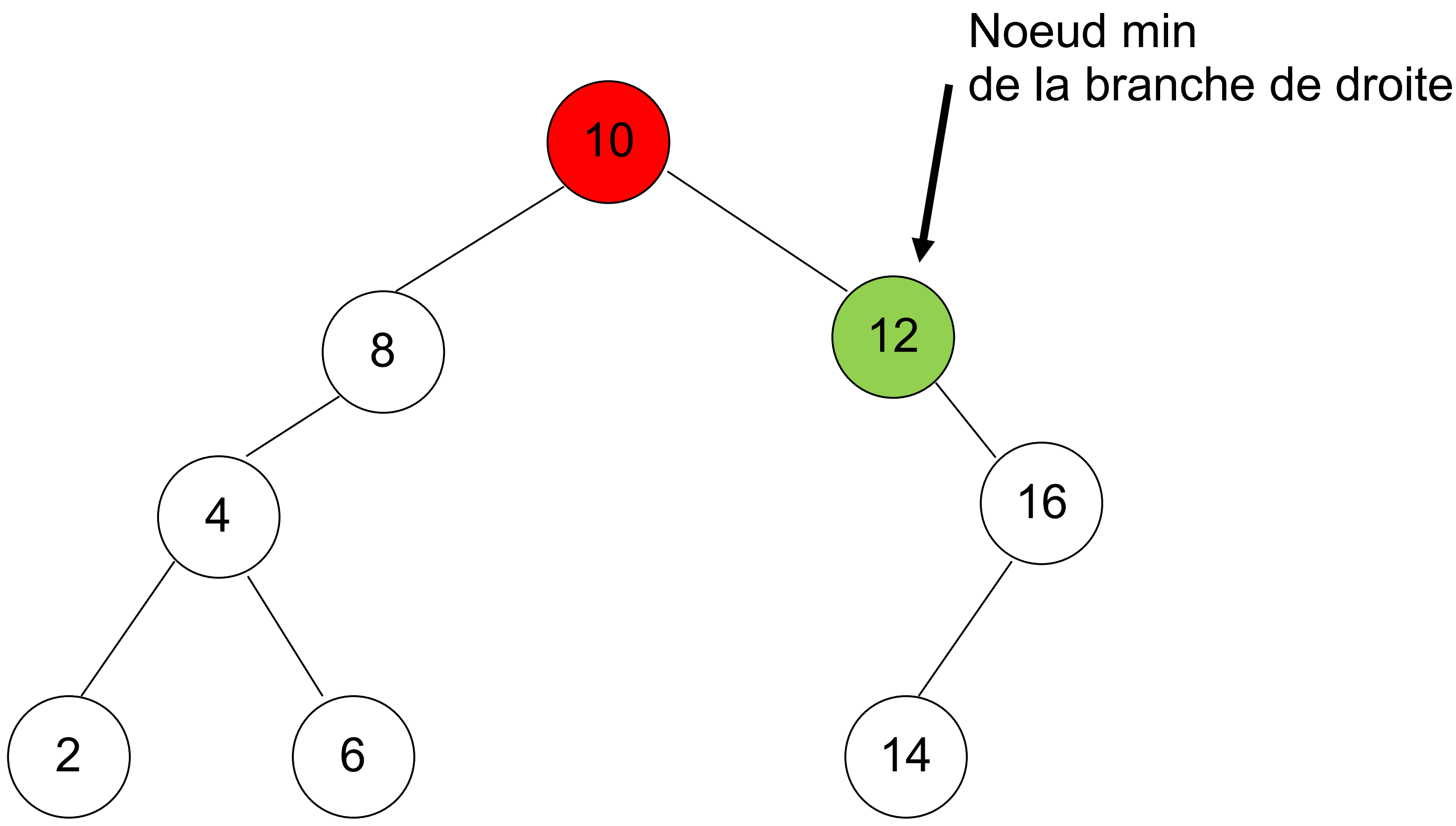
Splaying : remove

2. remove(root) …



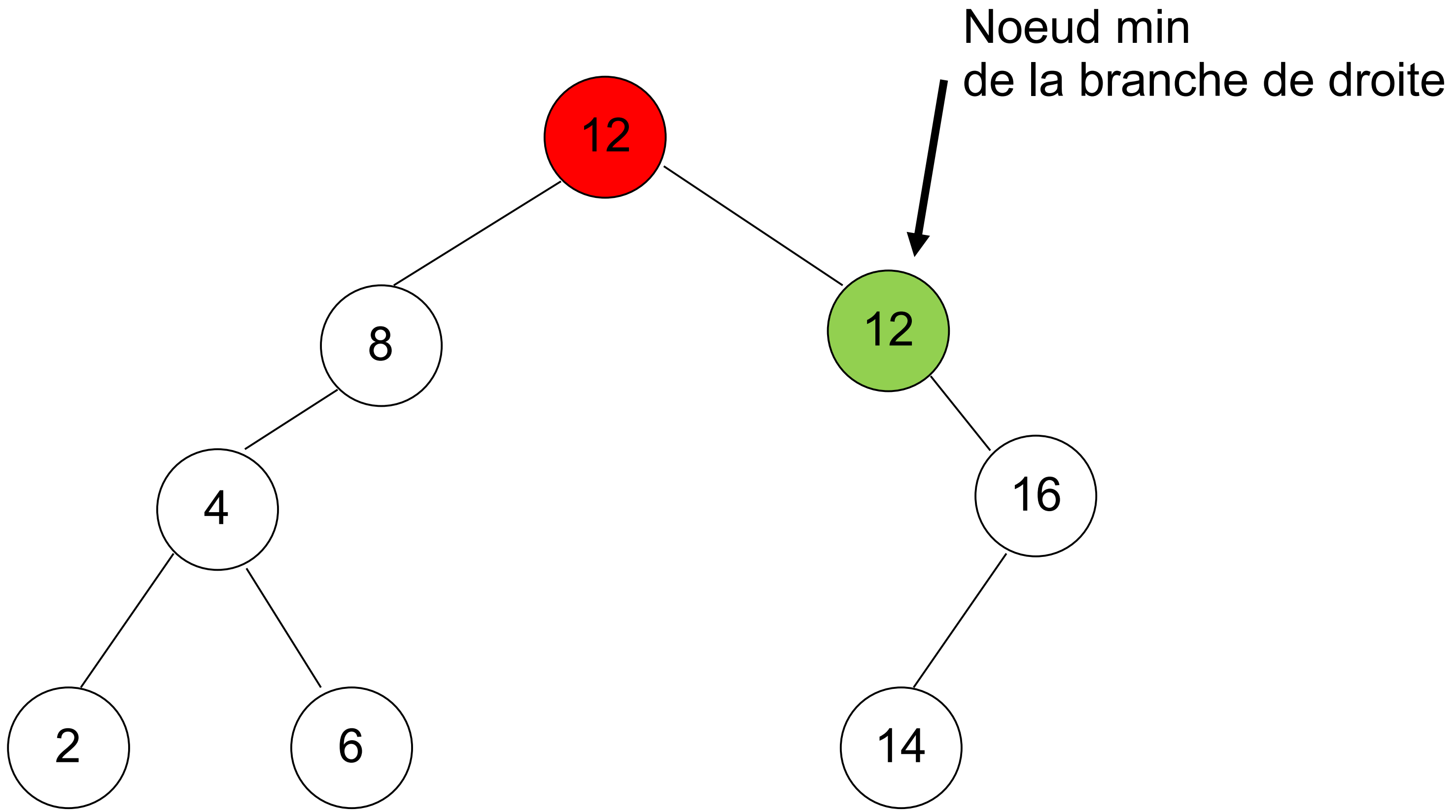
Splaying : remove

3. findMin(subTree(12))



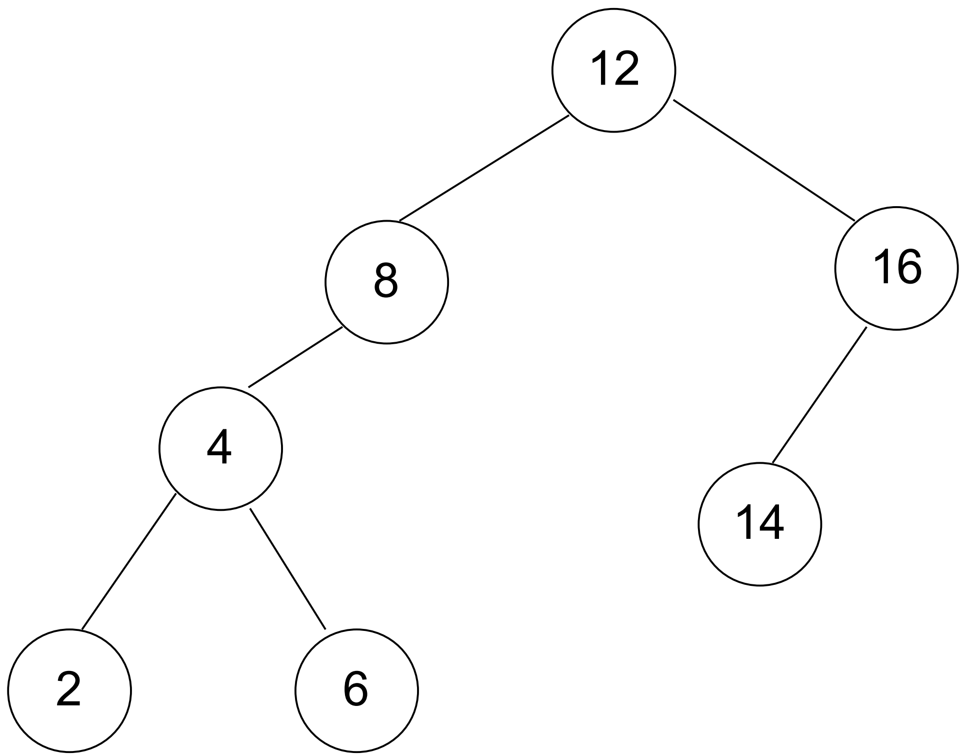
Splaying : remove

4. Overwrite root with min



Splaying : remove

4. remove(min)



Arbres équilibrés

Arbre AVL

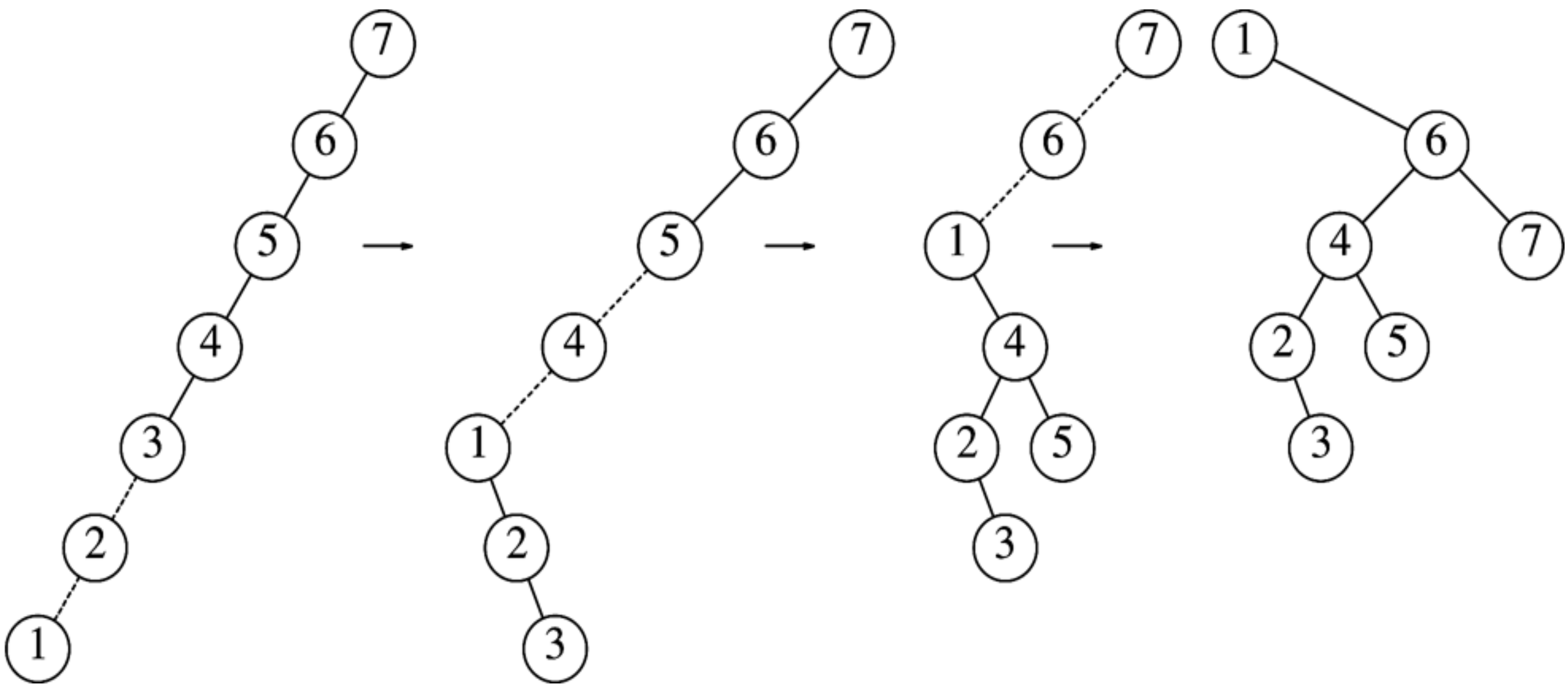
- Concepts de base de l’arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

Arbre Splay

- Concepts de base de l’arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

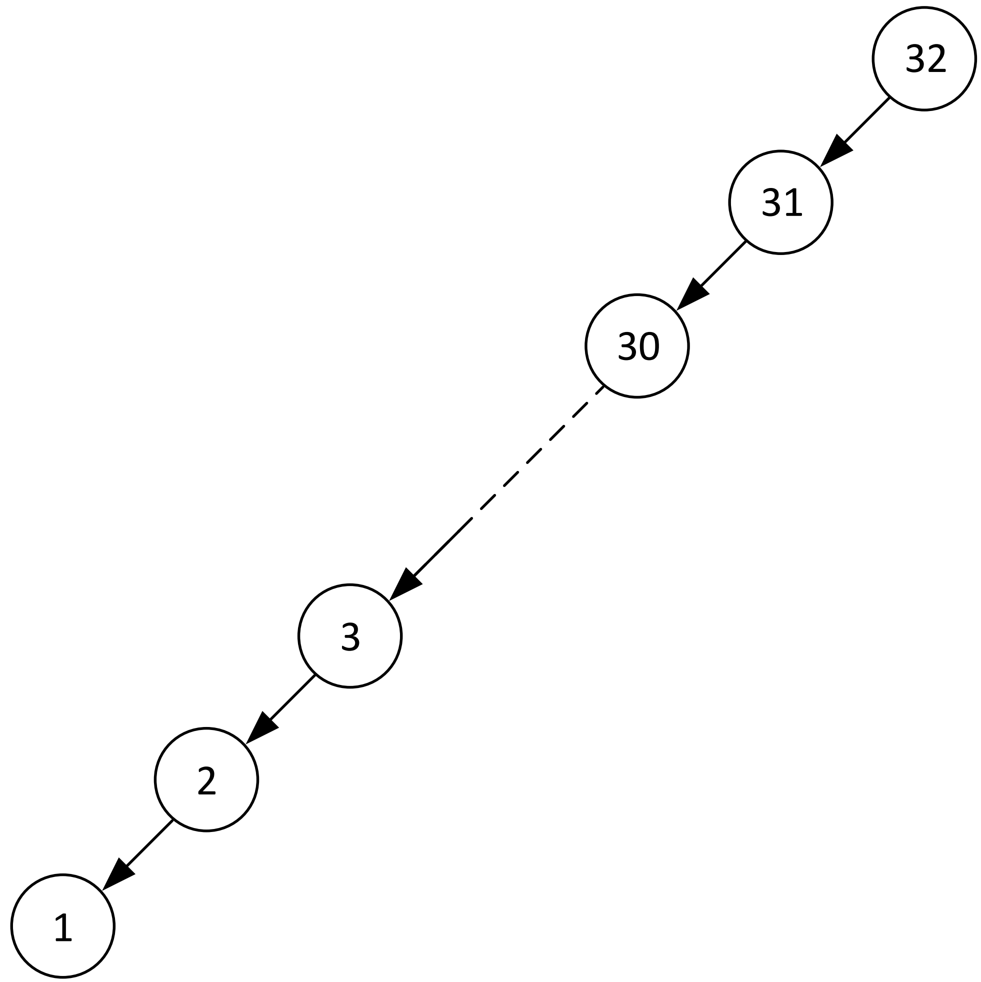
Splaying – Exemple

Les appels à zig, zig-zag et zig-zig continent tant que le noeud recherché n’est pas à la racine. Exemple avec get (1):



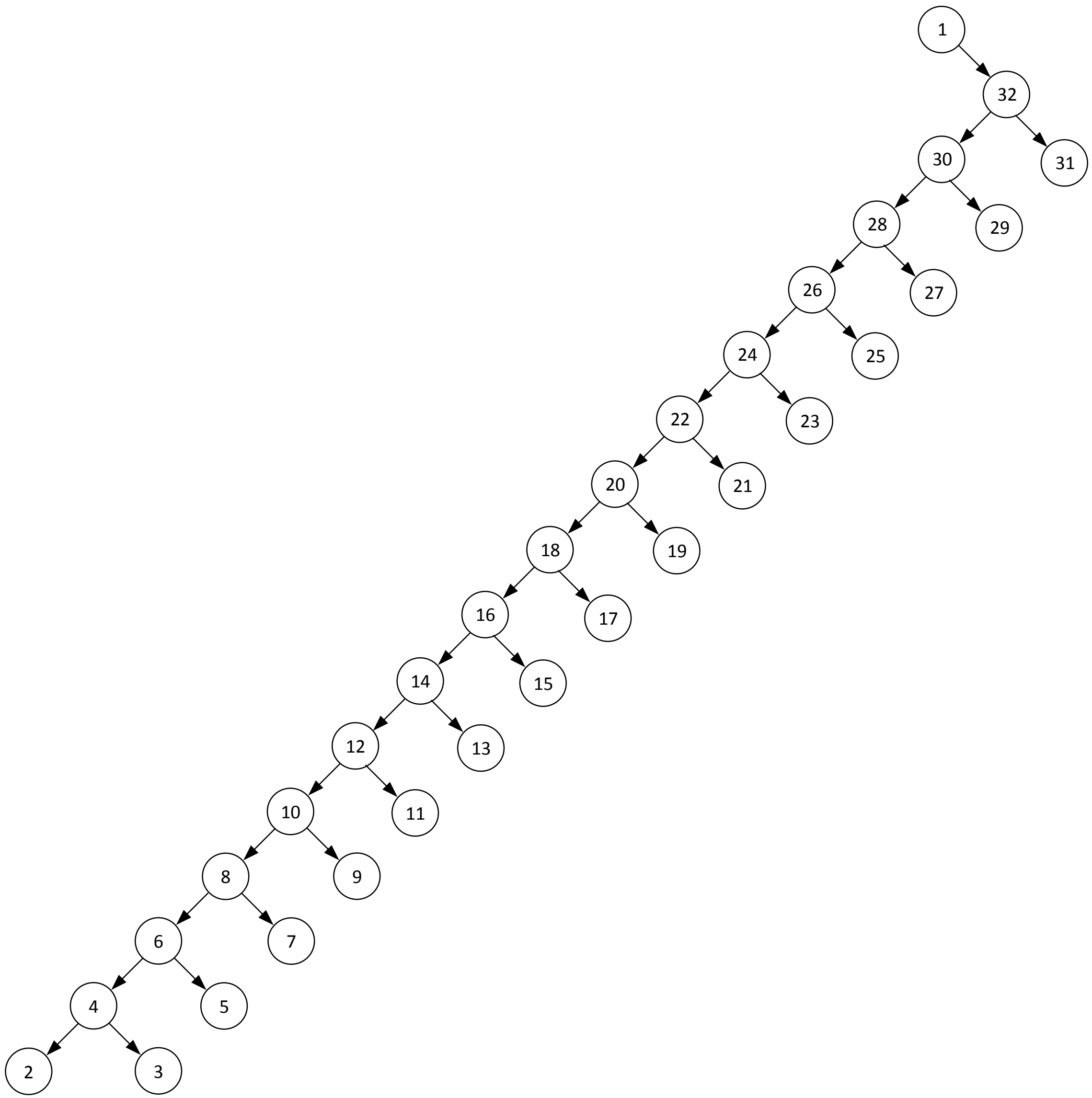
Splaying – Exemple 2

Splaying du noeud 1 d'un arbre de 32 entiers où chaque noeud possède seulement un fils gauche



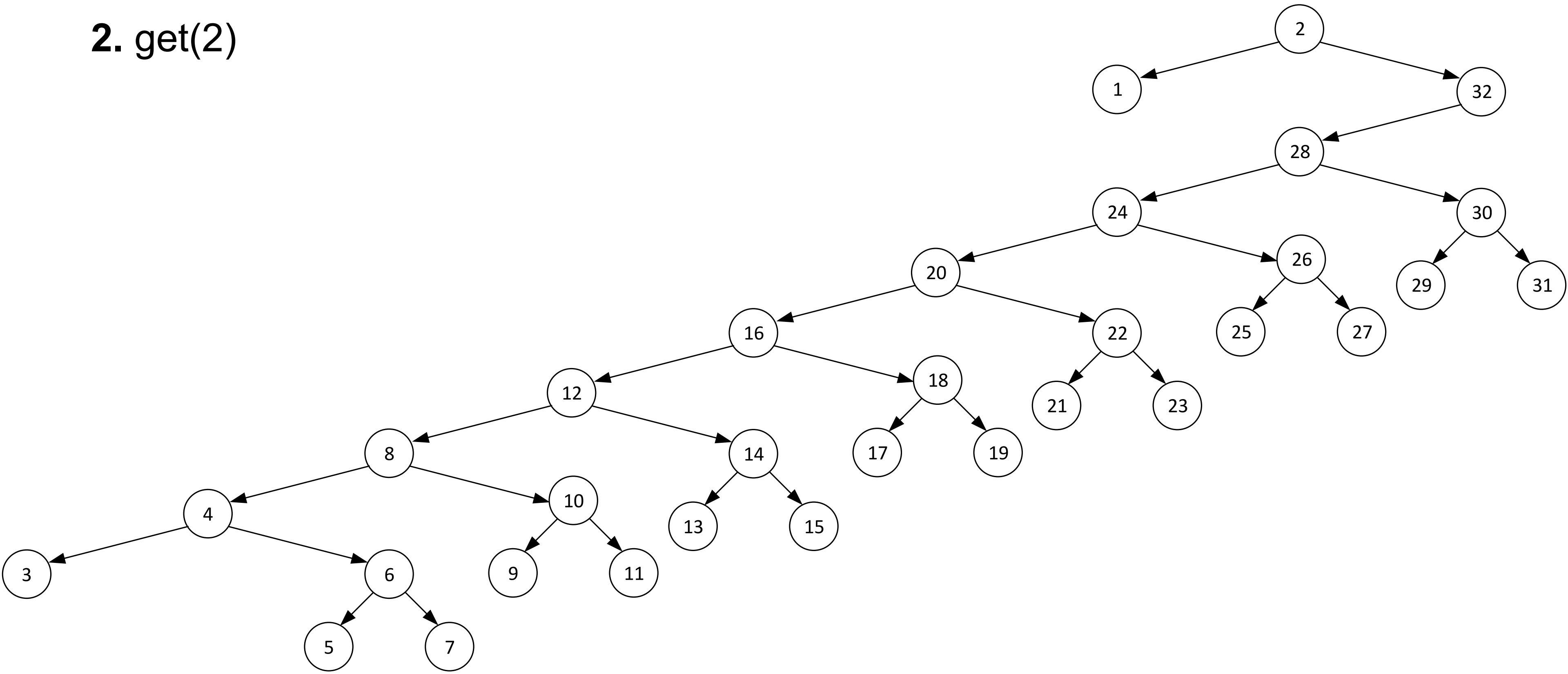
Splaying – Exemple 2

1. get(1)



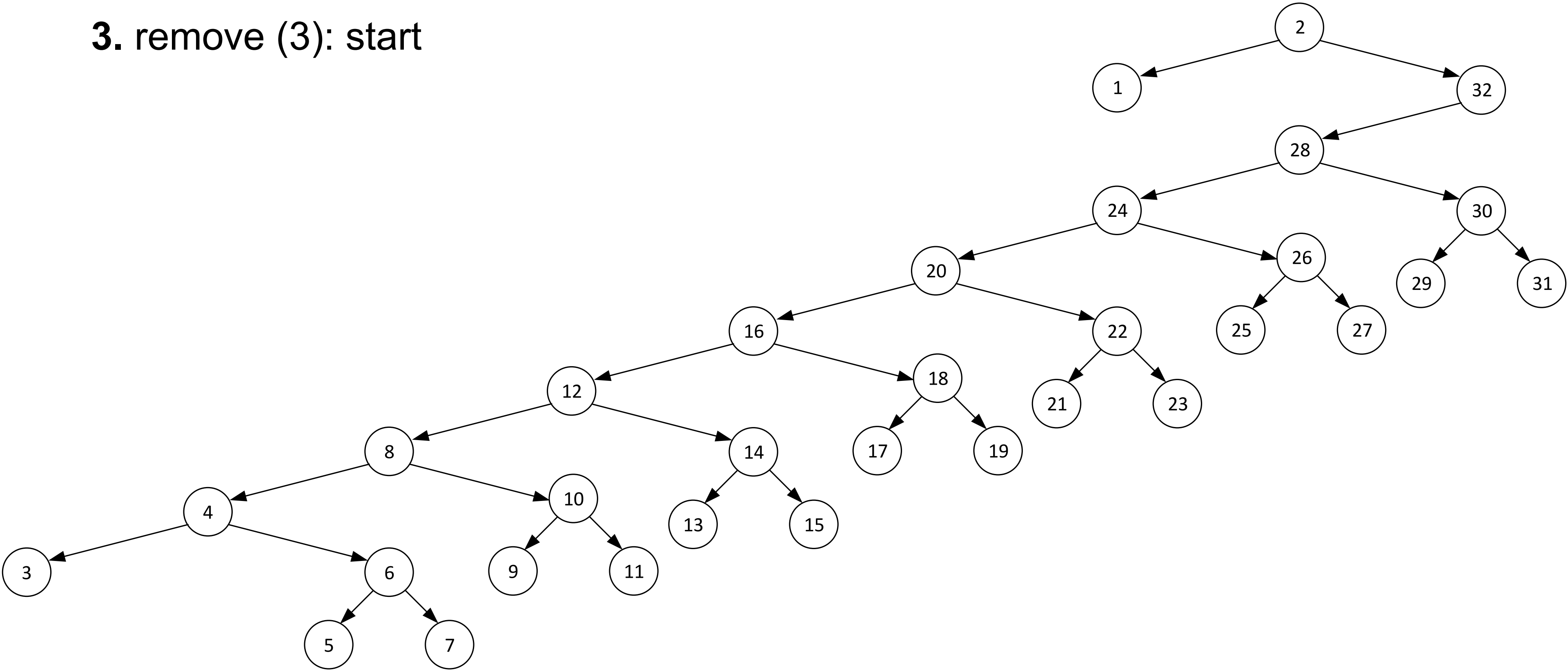
Splaying – Exemple 2

2. get(2)



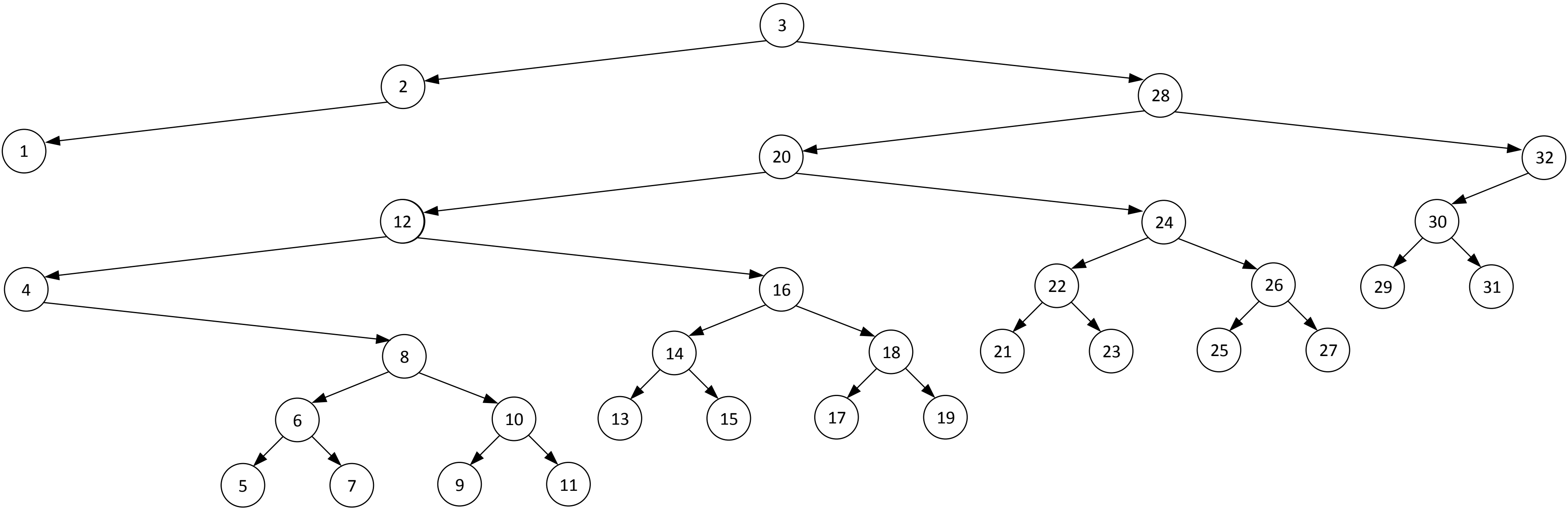
Splaying – Exemple 2

3. remove (3): start



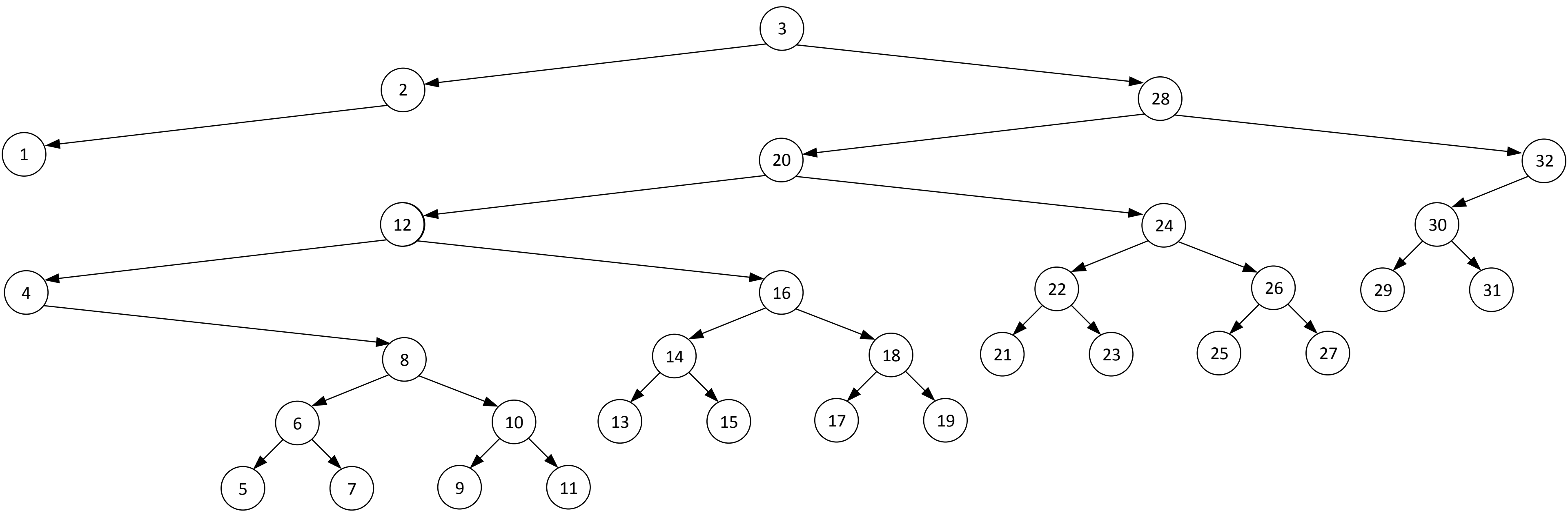
Splaying – Exemple 2

3.a remove(3): get(3)



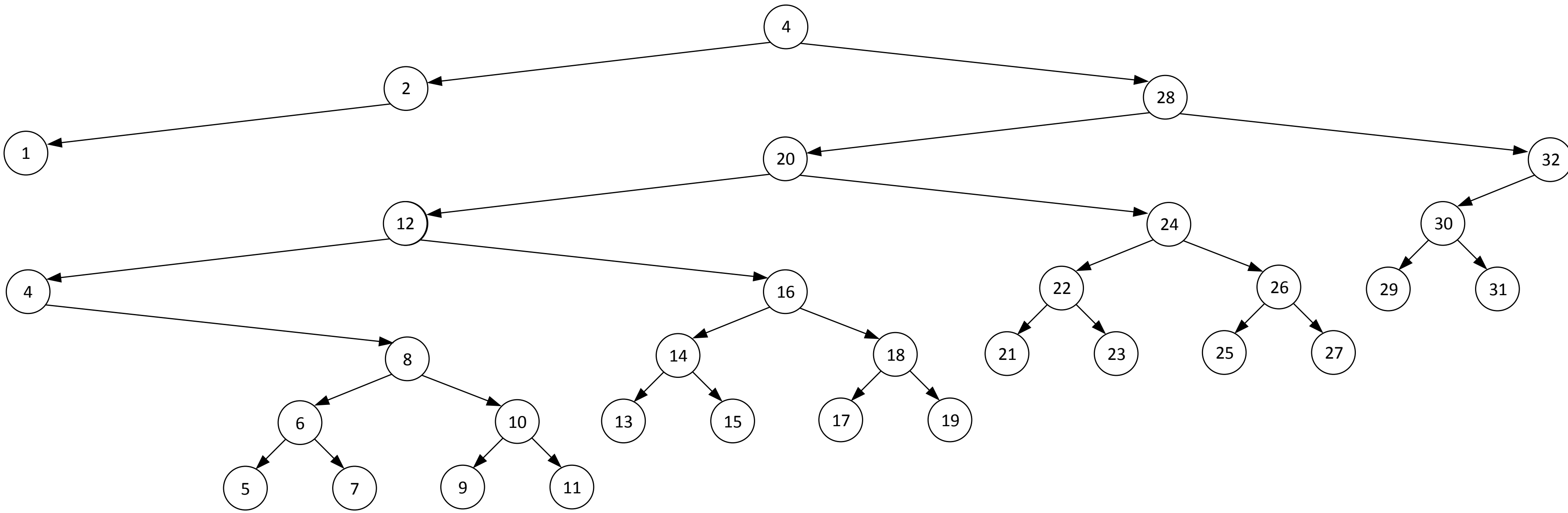
Splaying – Exemple 2

3.b remove(3): findMin(subtree(28))



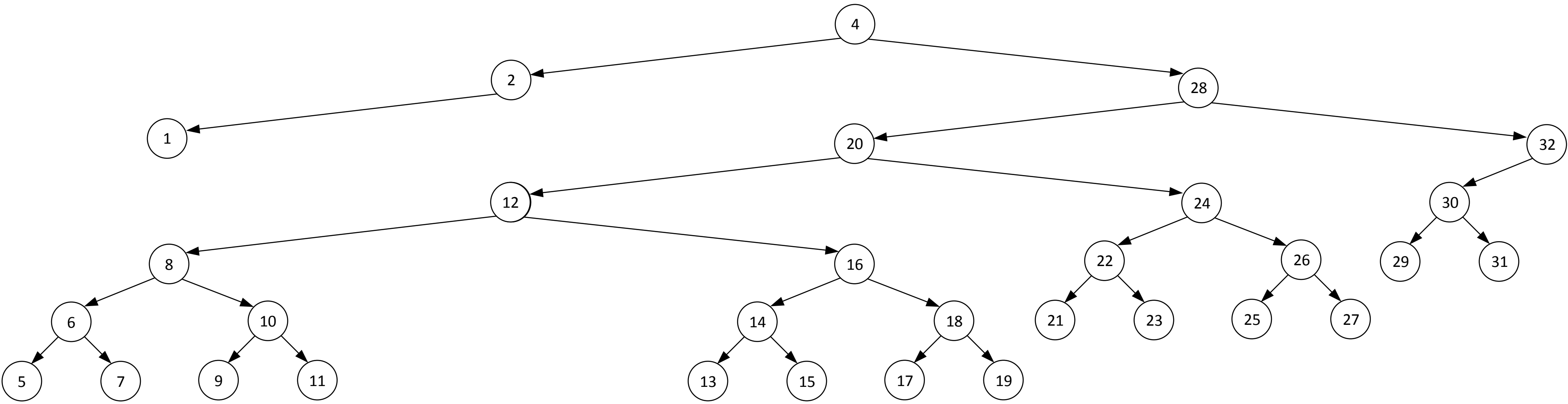
Splaying – Exemple 2

3.c remove(3): overwrite 3 with 4



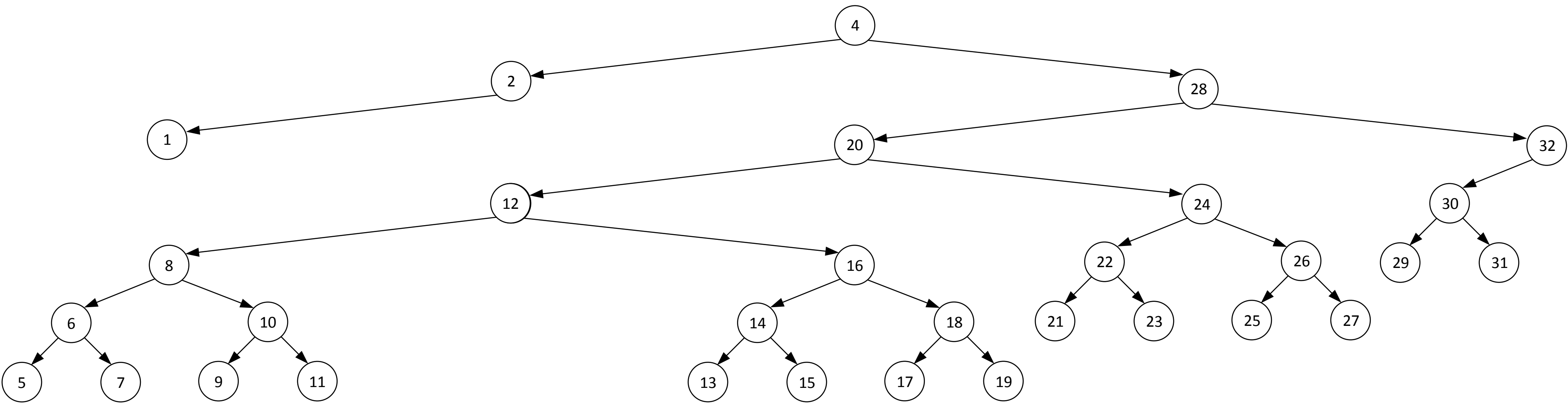
Splaying – Exemple 2

3.d remove(3): remove(4)



Splaying – Exemple 2

3. remove(3): end



Arbres équilibrés

Arbre AVL

- Concepts de base de l’arbre AVL
- Idée de rotations simple et double pour le rééquilibrage
- Implémentation
- Exemple détaillé

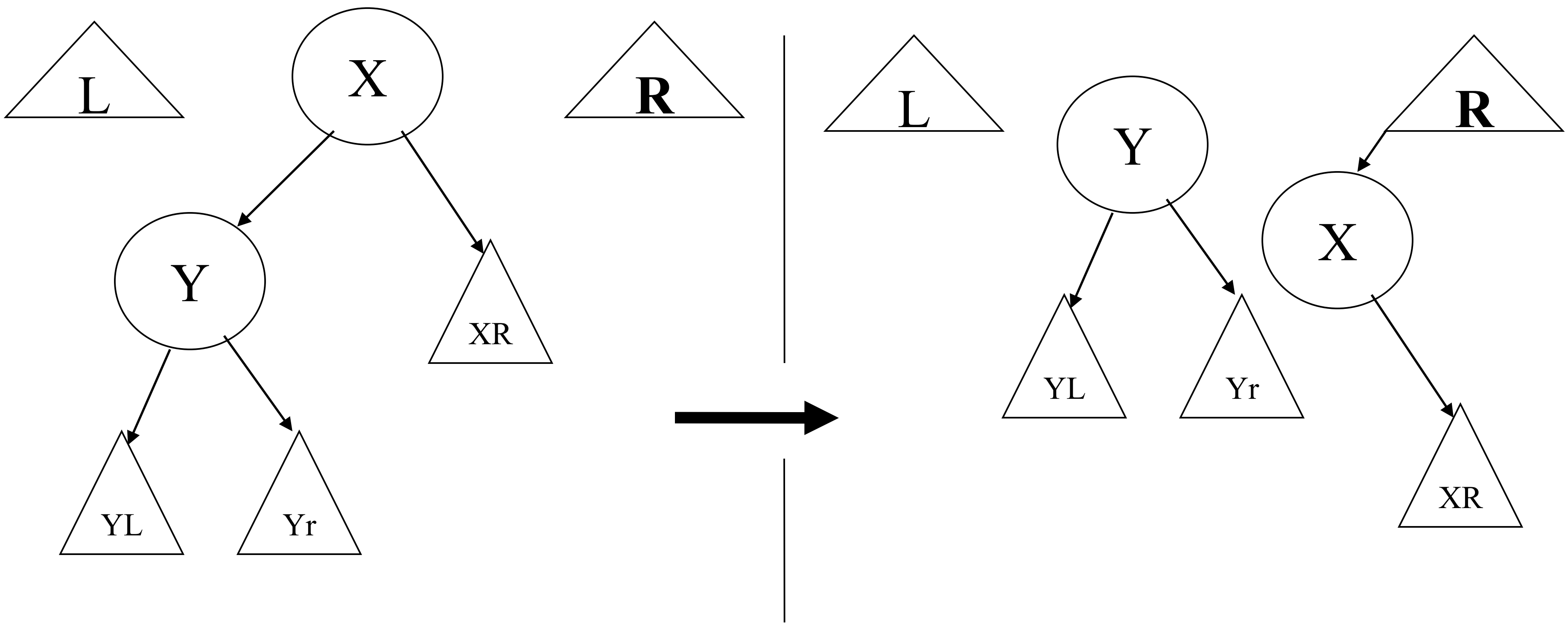
Arbre Splay

- Concepts de base de l’arbre Splay
- Types de rotation
- Procédure de retrait
- Exemples détaillés
- Implémentation top-down

Arbres Splay TOP – DOWN

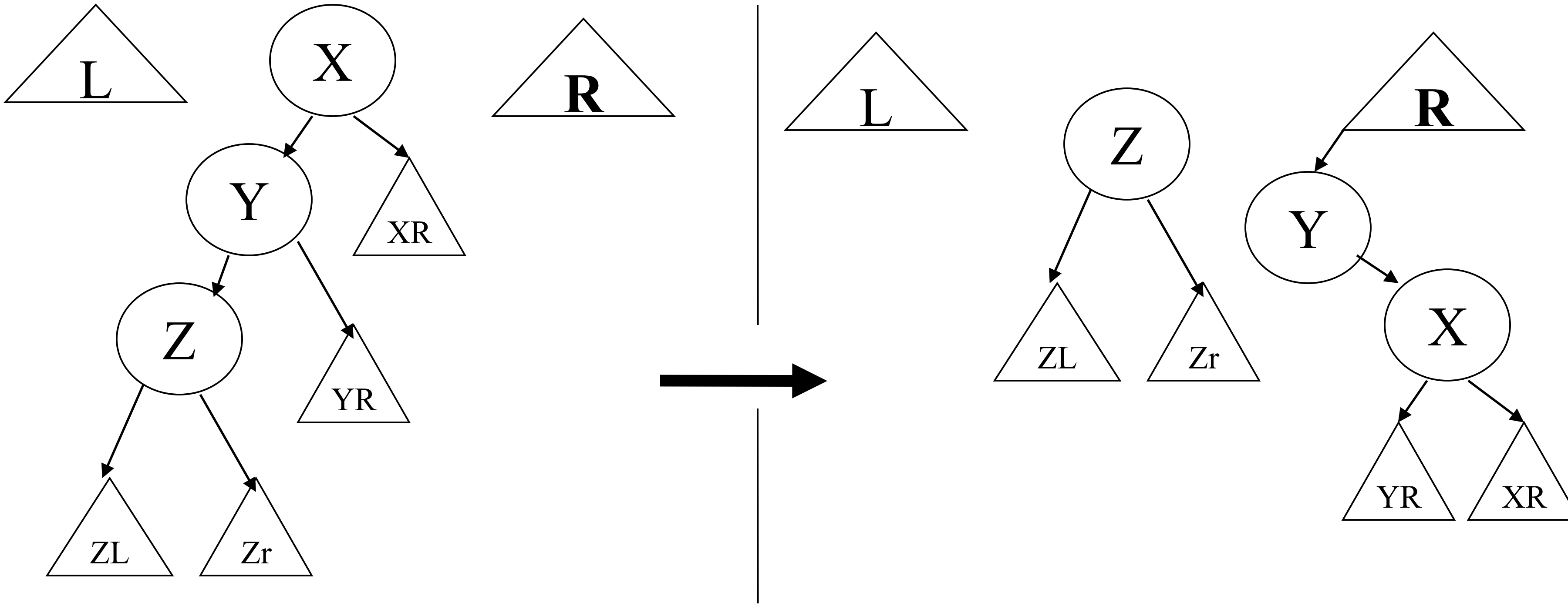
- Le splaying présenté jusqu'ici est appelé Bottom-up splaying. Il nécessite le parcours de l'arbre de la racine jusqu'au nœud concerné par le splay puis un retour vers la racine en appliquant les splay. Le chemin du splay est traversé deux fois.
- On veux éliminer une de ces traversées.
- Approche:
  - À chaque fois qu'on suit un lien gauche d'un nœud X, X et les éléments de son sous arbre droit sont tous > à l'élément qui va éventuellement devenir la racine. Dans ce cas, on sauvegarde X et son sous-arbre droit comme un arbre séparé appelé R.
  - Le cas symétrique est celui de suivre le lien droit d'un nœud X. Dans ce cas, X et son sous-arbre gauche sont sauvegardés dans un arbre séparé L.

Cas 1: Zig (TD)



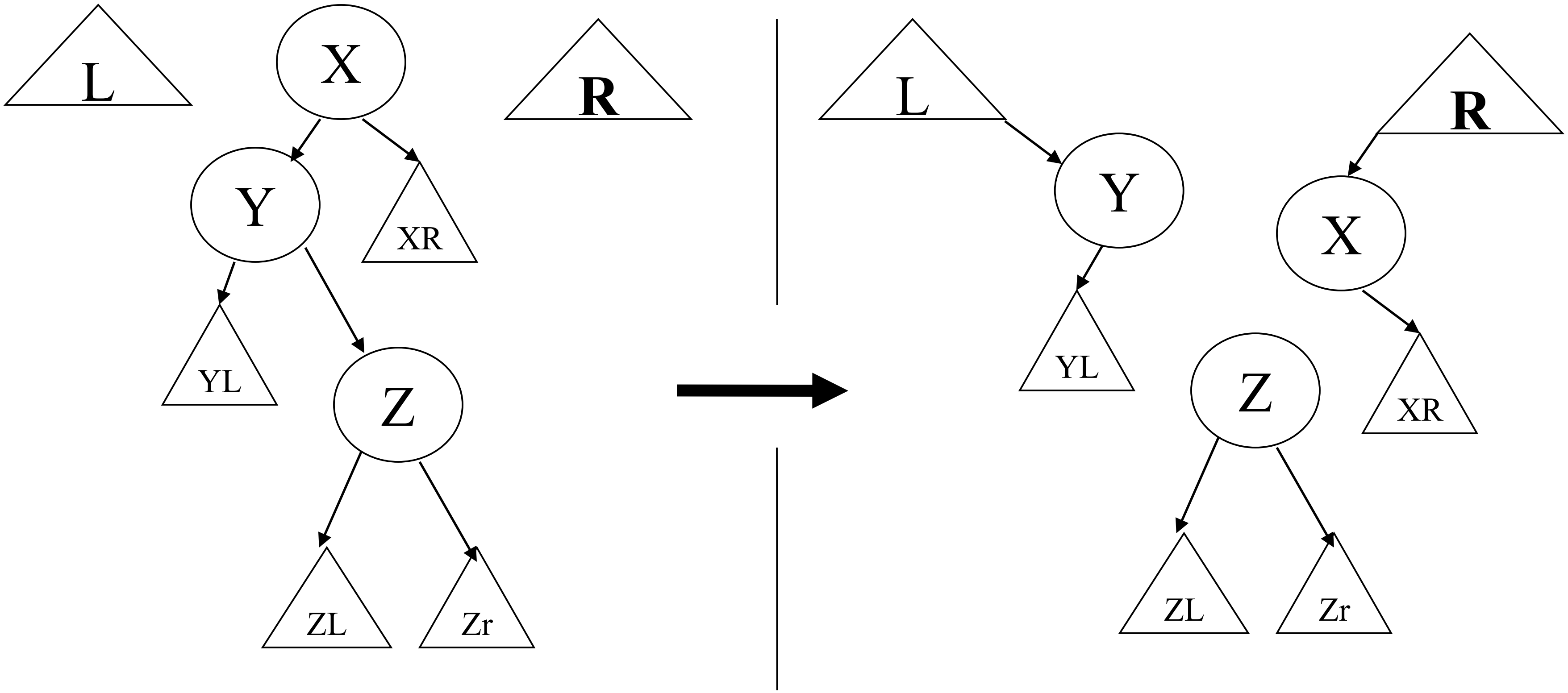
- Y doit devenir la racine
- X et son sous-arbre droit sont devenus fils gauche du plus petit élément de R
- Y est devenu la racine de l’arbre du centre.

Cas 2: Zig-Zig (TD)



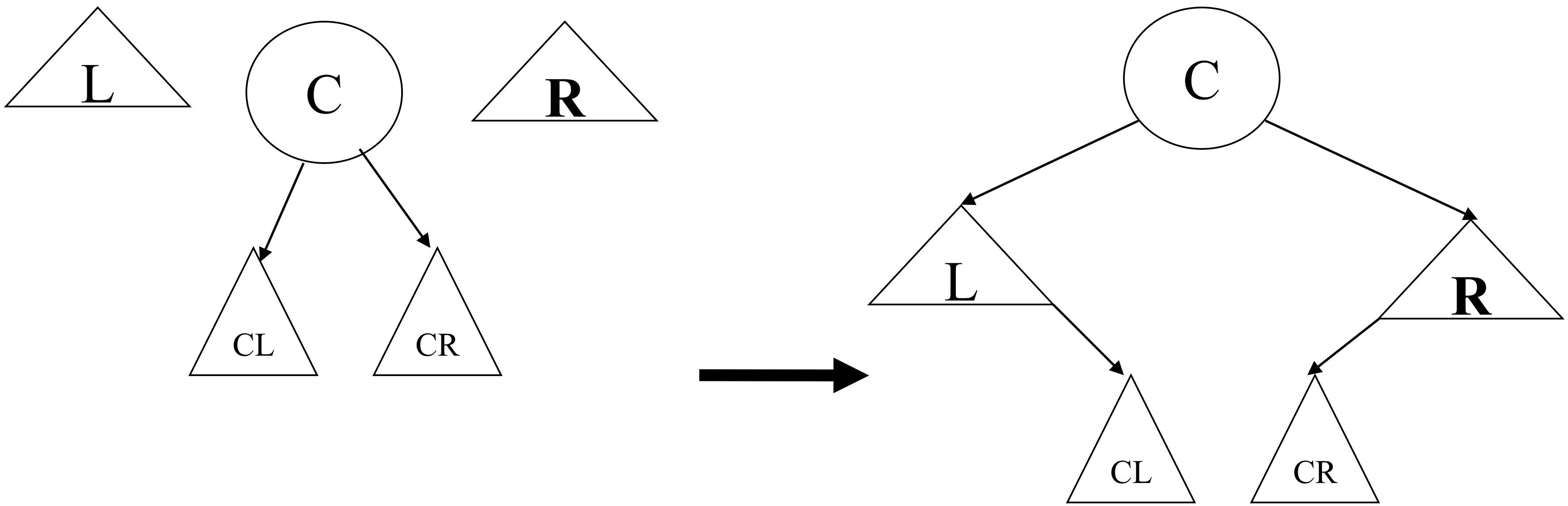
- La valeur sur laquelle le Splay doit s'effectuer est dans l'arbre Z.
- Effectuer une rotation à droite de Y et l'attacher comme fils gauche de l'élément le plus petit dans R.

Cas 3: Zig-Zag (TD)



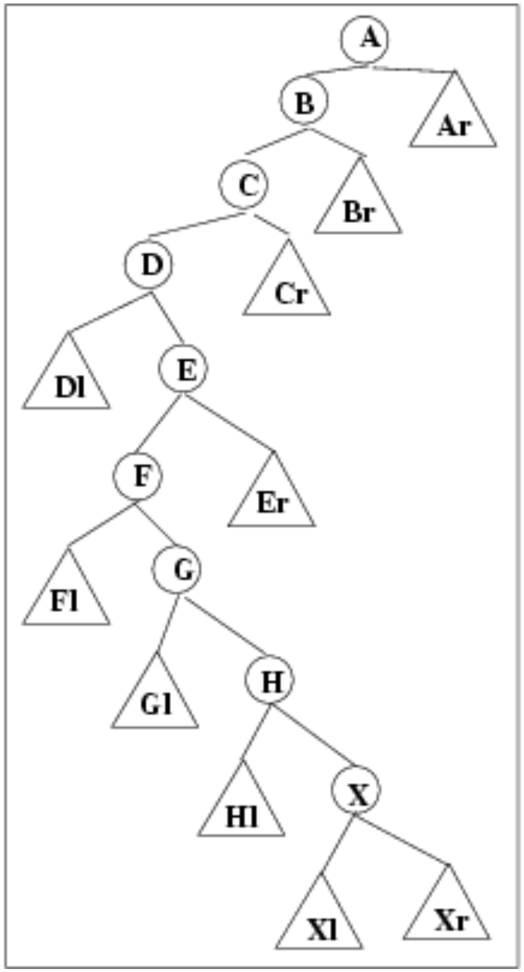
- L'élément sur lequel le Splay doit s'effectuer se trouve dans l'arbre dont la racine est Z.
- La rotation Zig-Zag est réduit en un simple Zig.
- Ceci génère plus d'itérations dans le processus Splay.

Rassembler l’arbre Splay

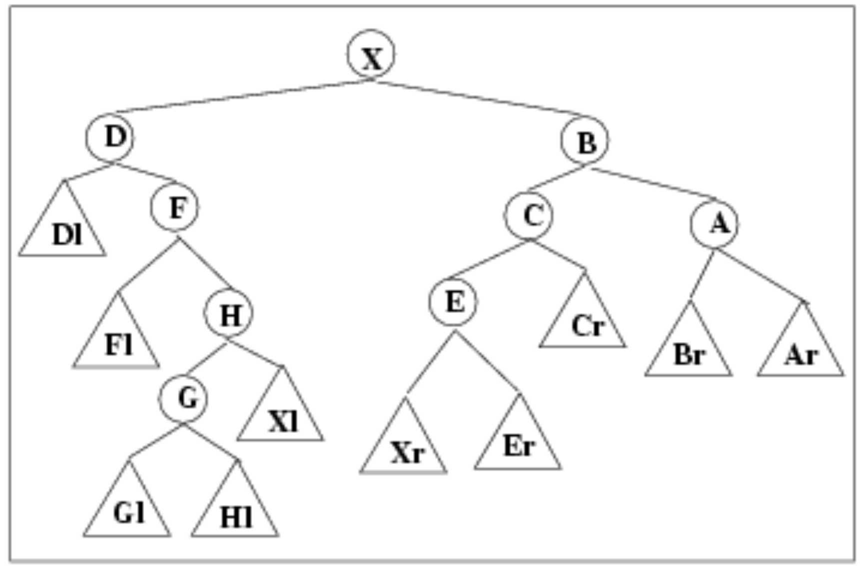


- Lorsque l’élément concerné par le Splay est devenu la racine de l’arbre du centre on atteint le point où l’arbre Splay doit être rassemblé.
- Il faut mettre CL comme fils droit du plus grand élément dans L.
- Il faut mettre CR comme fils gauche du plus petit élément dans R.
- Il faut mettre L et R comme fils gauche et fils droit de C respectivement.

Exemple: (en appliquant le Splay bottom-up)

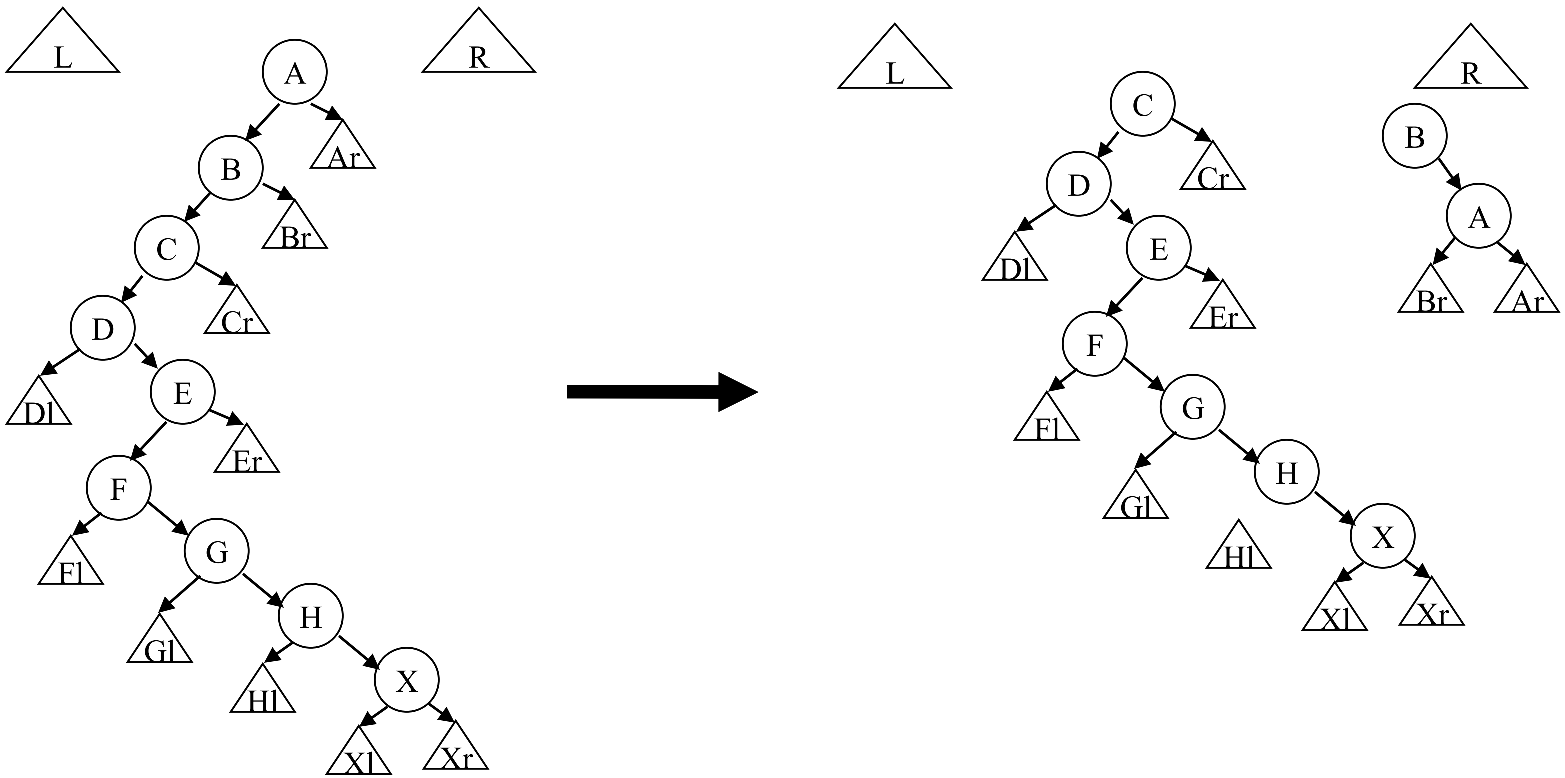


Arbre original

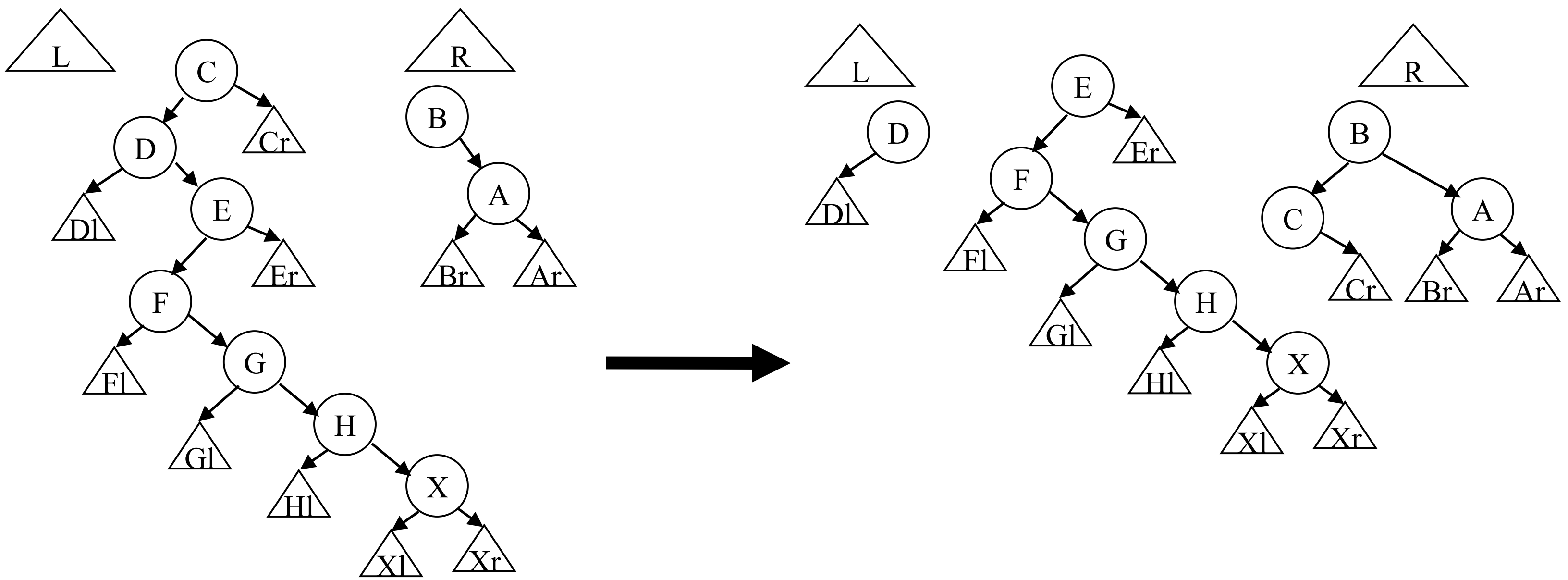


Arbre après le Splay du nœud X

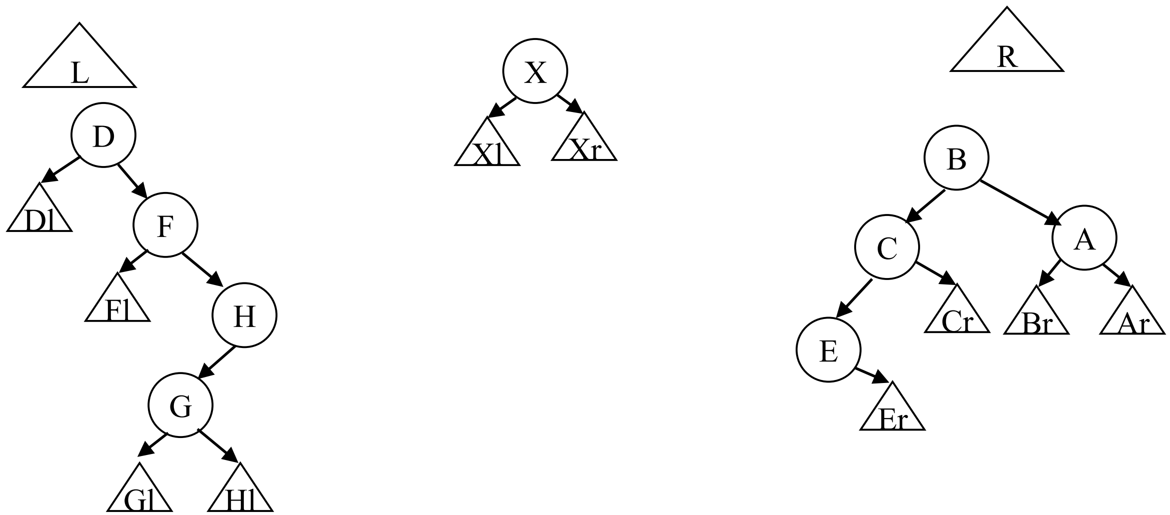
Operation 1: Zig-Zig



Operation 2: Zig-Zag



Lorsque X atteint la racine:

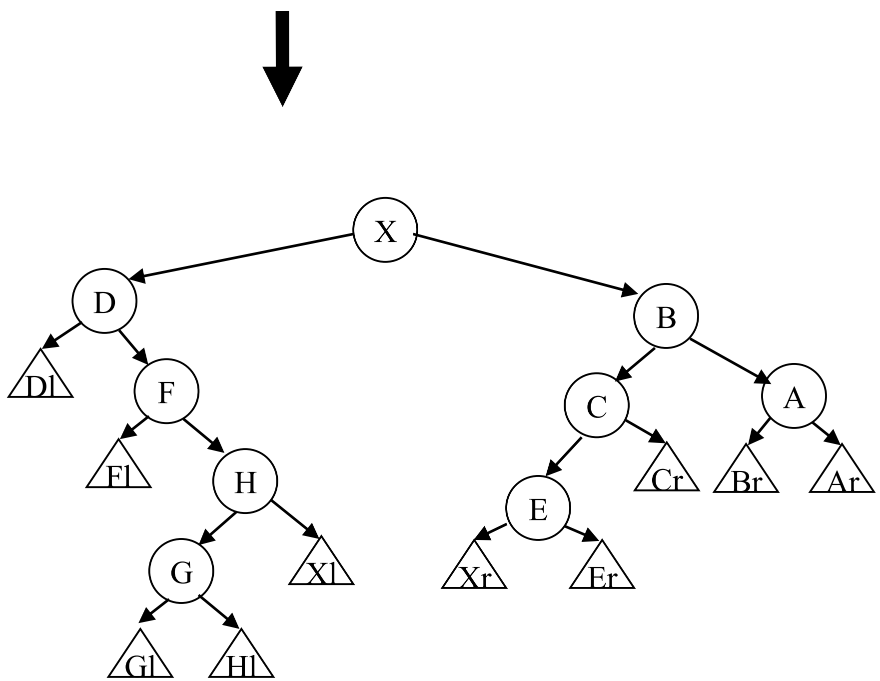


Xl devient le fils droit de H

Xr devient le fils gauche de E

L et R deviennent les fils gauche

et droit de X respectivement.



Arbres Splay - Avantages

- ***Amélioration du temps d'accès global d'une*** séquences d'opérations.
- ***Amélioration du temps d'accès*** pour les éléments fréquemment appelés car il seront dans la racine ou très proches d'elle.
- On n'a pas besoin de maintenir les hauteurs des sous-arbres ni l'information relative à l'équilibrage de l'arbre **=>** gain en espace et simplification du code.