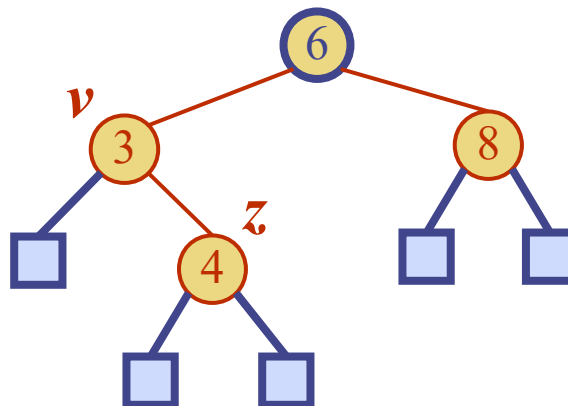


ALGORITMOS Y ESTRUCTURA DE DATOS III

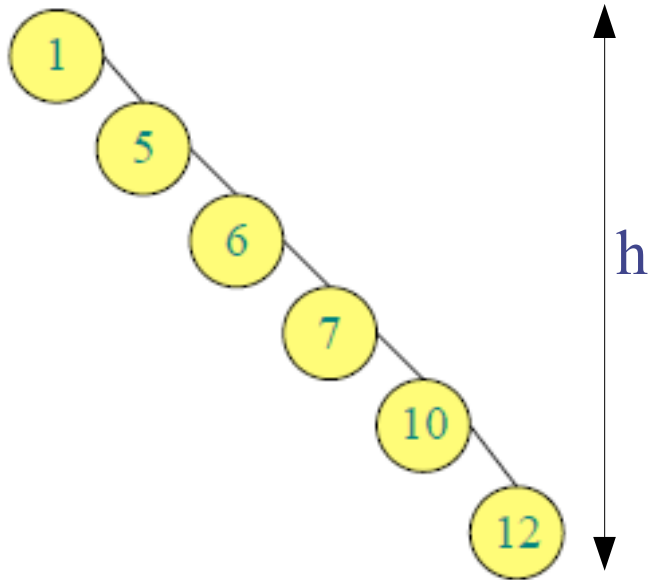
Sección TQ - 1er Semestre/2018

Prof. Cristian Cappo

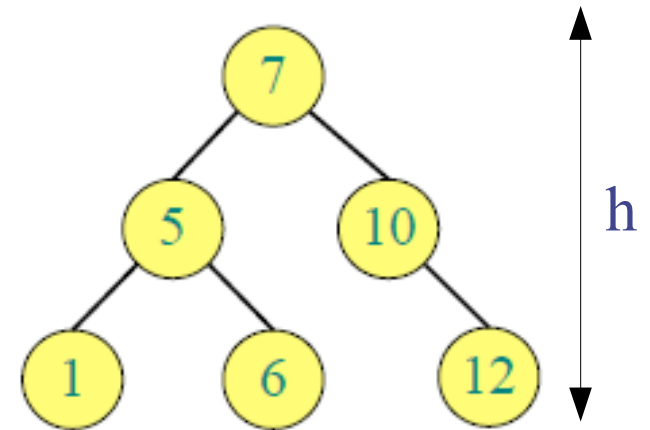
Balanceo de Árboles Binarios de Búsqueda Ejemplo: Árboles AVL



La importancia de tener un árbol balanceado



$$h = O(n)$$



$$h = O(\log n)$$

Estrategia para mantener balanceo (una de ellas)

- Agregar a cada nodo algún dato relevante.
- Definir un *invariante* local en los datos.
- Probar que la *invariante* garantiza una altura $O(\log n)$
- Diseñar los algoritmos para mantener los datos y la invariante.

Ejemplos:

AVL (*veremos*)

Rojinegros (Red-Black Tree)

Treap (*veremos*)

Existen otras formas en las que no se mantiene información adicional:

- *splayTree*

- *scapegoatTree*

Árbol AVL

■ Propuesto por **Adelson-Velskii,** **G.; E. M. Landis** (de ahí el nombre):

- ❖ Artículo: "*An algorithm for the organization of information*".
- ❖ Presentado en la "USSR Academy of Sciences", en 1962

AN ALGORITHM FOR THE ORGANIZATION OF INFORMATION

G. M. ADEL'SON-VEL'SKII AND E. M. LANDIS

In the present article we discuss the organization of information contained in the cells of an automatic calculating machine. A three-address machine will be used for this study.

Statement of the problem. The information enters a machine in sequence from a certain reserve. The information element is contained in a group of cells which are arranged one after the other. A certain number (the information estimate), which is different for different elements, is contained in the information element. The information must be organized in the memory of the machine in such a way that at any moment a very large number of operations is not required to scan the information with the given evaluation and to record the new information element.

An algorithm is proposed in which both the search and the recording are carried out in $O(\lg N)$ operations, where N is the number of information elements which have entered at a given moment.

A part of the memory of the machine is set aside to store the incoming information. The information elements are arranged there in their order of entry. Moreover, in another part of the memory a "reference board" [1] is formed, each cell of which corresponds to one of the information elements.

The reference board is a dyadic tree (Figure 1a): each of its cells has no more than one left cell, and no more than one right cell subordinated to it. Direct subordination induces subordination (partial ordering). In addition, for each cell of the tree, all the cells which are subordinate to a left (right) directly subordinate cell, will be arranged further to the left (right) than the given cell. Moreover, we assume that there is a cell (the head) to which all the others are subordinate. By transitivity, the conception "further to the left" and "further to the right" extends to the aggregate of all the cell pairs, and this aggregate becomes ordered. Thus, a given order of cells in a reference board should coincide with the order of arrangement of the estimates of the corresponding information elements (to be specific, we shall consider the estimates as increasing from left to right).

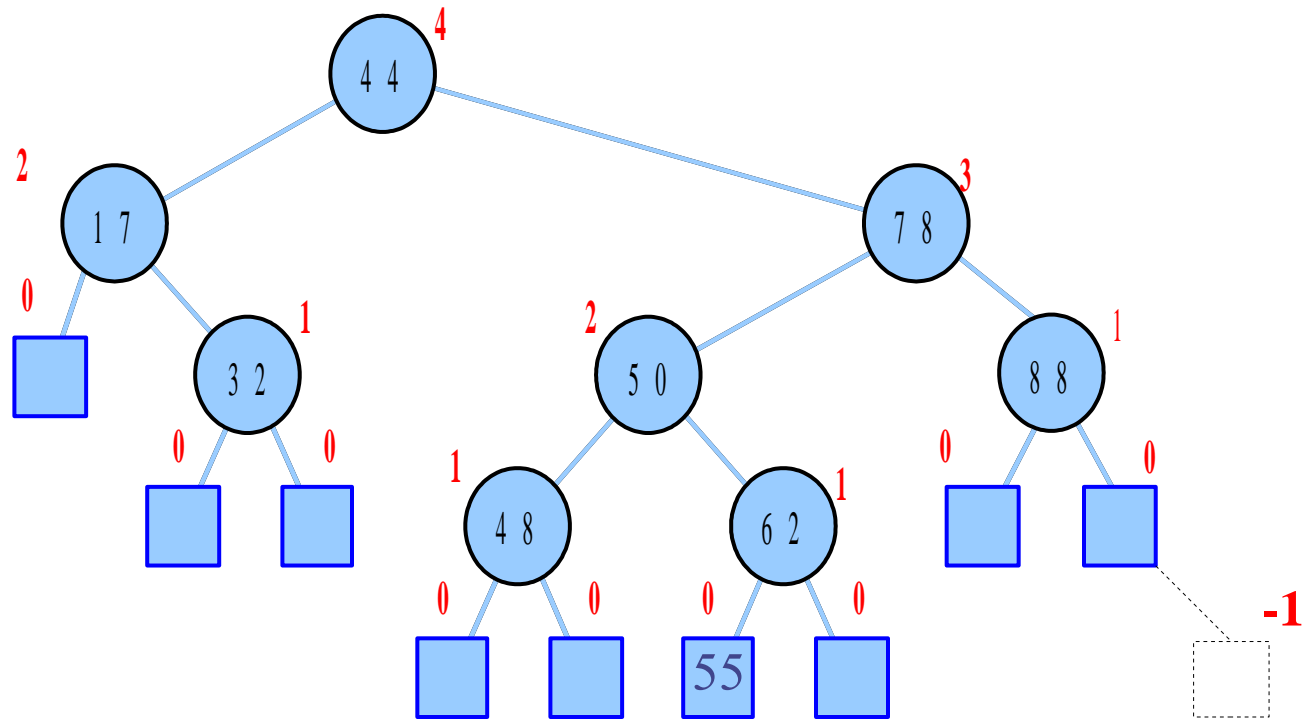
In the first address of each cell of the reference board, a place is indicated where the corresponding information element is located. The addresses of the cells of the reference board, which are directly subordinate on the left and right respectively to the given cell, are located in the second and third addresses. If a cell has no directly subordinate cells on either side, then there is zero in the corresponding address. The head address is stored in a certain fixed cell L .

Let us call the sequence of the cells of the tree a *chain* in which each previous cell is directly

Árbol AVL

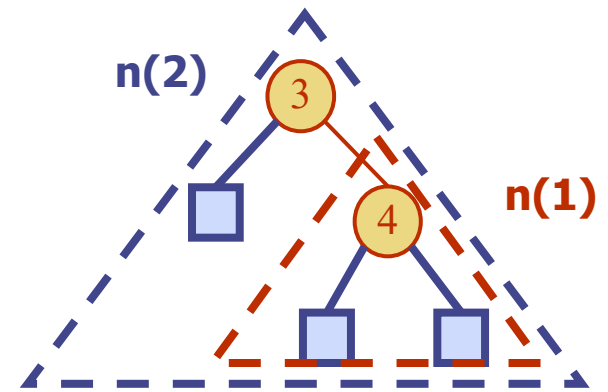
- *Es un Árbol Binario de Búsqueda (BST)*
- *Dato adicional* : altura del nodo
 - ❖ Hojas llenas tienen altura 0
 - ❖ Hojas vacías (NULL o NIL) tiene altura -1
- *Invariante* : en cada nodo **X**, las alturas de su hijo derecho e izquierdo pueden diferir en hasta 1

Ejemplo de un árbol AVL



Un ejemplo de un árbol AVL donde las alturas se muestran al lado de cada nodo

Altura de un árbol AVL



- **Hecho:** La altura de un árbol AVL de n nodos es $O(\log n)$.
- **Prueba:** Dado n_h : el número mínimo de nodos internos de un árbol AVL de altura h .
- Podemos ver que $n_1 = 1$ y $n_2 = 2$
- Para $n > 2$, un árbol AVL de altura h contiene la raíz, un subárbol AVL de altura $h-1$ y otro de altura $h-2$.
- Esto es, $n_h = 1 + n_{h-1} + n_{h-2}$
- Sabiendo que $n_{h-1} > n_{h-2}$, podemos decir $n_h > 2n_{h-2}$. Así
 - ❖ $n_h > 2n_{h-2}$, $n_h > 4n_{h-4} > 8n_{h-6}$, .. por inducción
 - ❖ $n_h > 2^i n_{h-2i}$
- Ahora resolvemos esta ecuación en la sigte. lámina

Altura de un árbol AVL (cont.)

$n_h > 2^i n_{h-2i}$ (*) para i tal que $h-2i \geq 1$

conocemos que $n_1=1$ $n_2=2$

podemos hacer $h-2i=2$ para $n_2=2$

$i=h/2-1$ reemplazando en (*)

$$n_h > 2^{h/2-1} n_{h-2(h/2-1)}$$

$$n_h > 2^{h/2-1} \cdot 2 \text{ ya que } n_2=2$$

$$\text{Tenemos } n_h > 2^{h/2}$$

$$\text{Haciendo } \log(n_h) > \frac{h}{2\log(2)}$$

$$\text{Así } h < 2\log(n_h) \text{ es decir } h = O(\log(n))$$

(en Weiss es $h < 1,44 \log(n+2) - 1,328$)

- Esto es, la altura de un árbol AVL está en $O(\log n)$

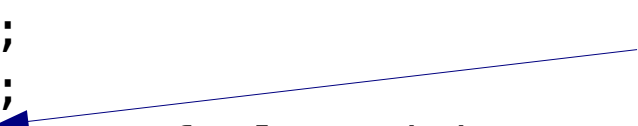
Estructura de datos AVL (Nodo)

```
public class AvlNode /* @author Mark Allen Weiss */
{
    AvlNode( Comparable theElement ) {
        this( theElement, null, null );
    }

    AvlNode( Comparable theElement, AvlNode lt, AvlNode rt ){
        element    = theElement;
        left       = lt;
        right      = rt;
        height     = 0; // Default Height */
    }

    Comparable element;           // The data in the node
    AvlNode    left;              // Left child
    AvlNode    right;             // Right child
    int        height;            // Height
}
```

¿Porqué 0?



Estructura de datos AVL (contenedor)

```
public class AvlTree /* @author Mark Allen Weiss */
{
    /** The tree root. */
    private AvlNode root;

    /**
     * Construct the tree.
     */
    public AvlTree( )
    {
        root = null;
    }

    public void insert( Comparable x )
    {
        root = insert( x, root );
    }
    ...
}
```

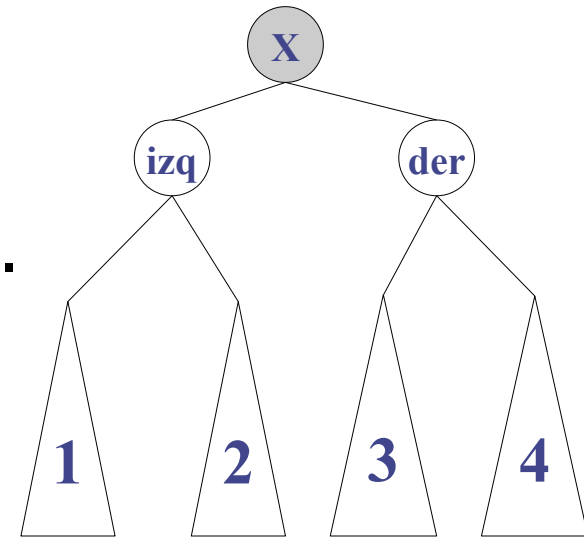
Inserción en un AVL

- Se requiere
 - ❖ Verificar la condición de equilibrio
 - ❖ Si hay desequilibrio:
 - ◆ Modificar el árbol
 - ◆ Actualizar la altura de los nodos que se encuentran en el camino de la actualización

Esto se conoce como ROTACIÓN
(debemos asegurar la ordenación en el árbol, esto es que continúe siendo BST)

Inserción en un árbol AVL

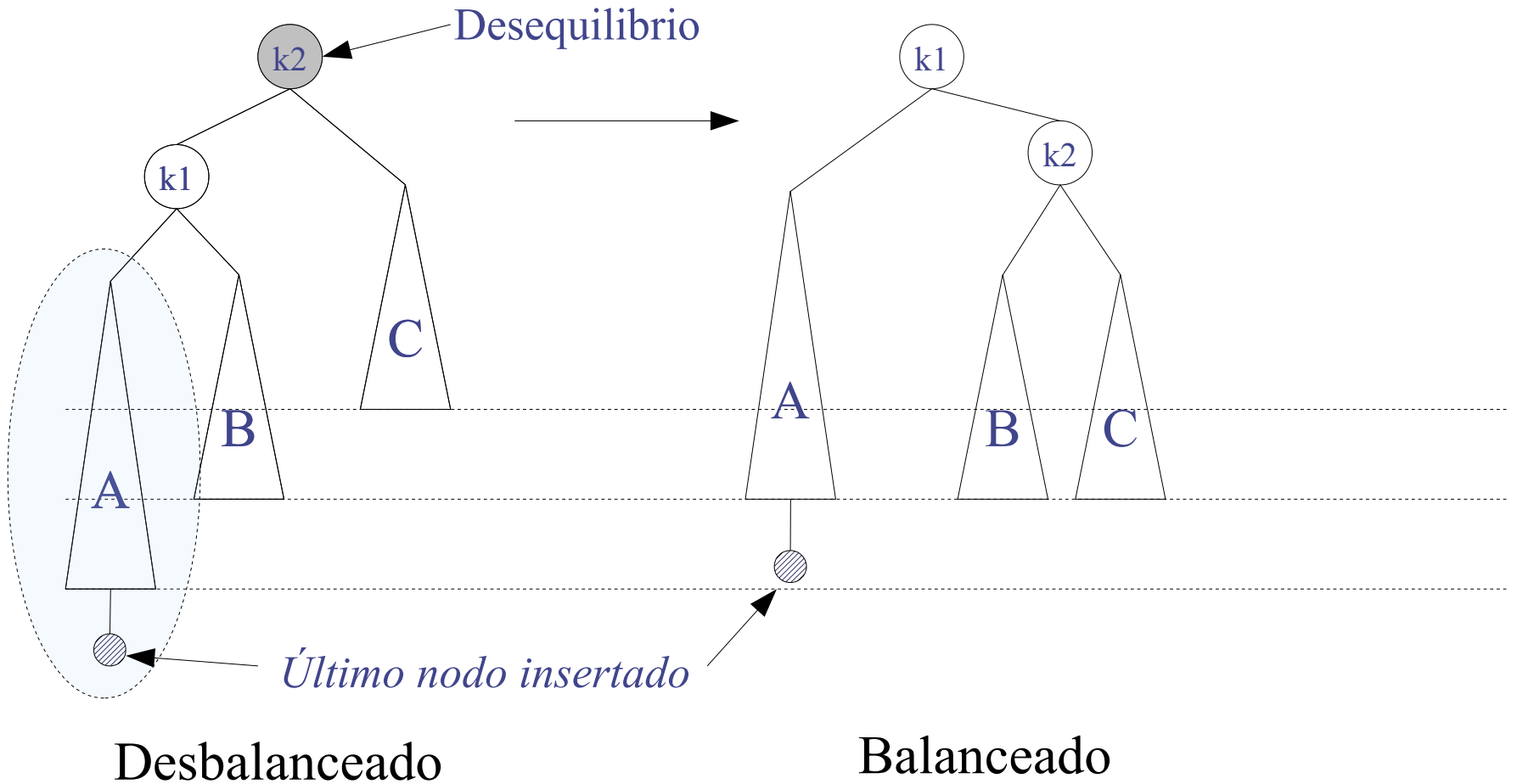
- 1) En el subárbol izq. del izquierdo de X.
- 2) En el subárbol der. del izquierdo de X.
- 3) En el subárbol izq. del derecho de X.
- 4) En el subárbol der. del derecho de X.



- ❖ Los casos 1 y 4 son simétricos respecto a X
 - ♦ Se produce en las márgenes, *rotación simple*
- ❖ Los casos 2 y 3 son simétricos respecto a X
 - ♦ Se produce hacia adentro, *rotación doble*

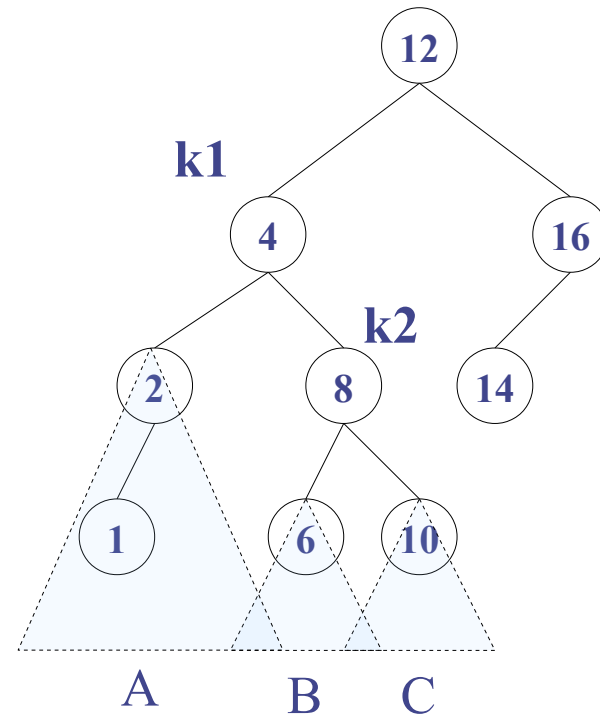
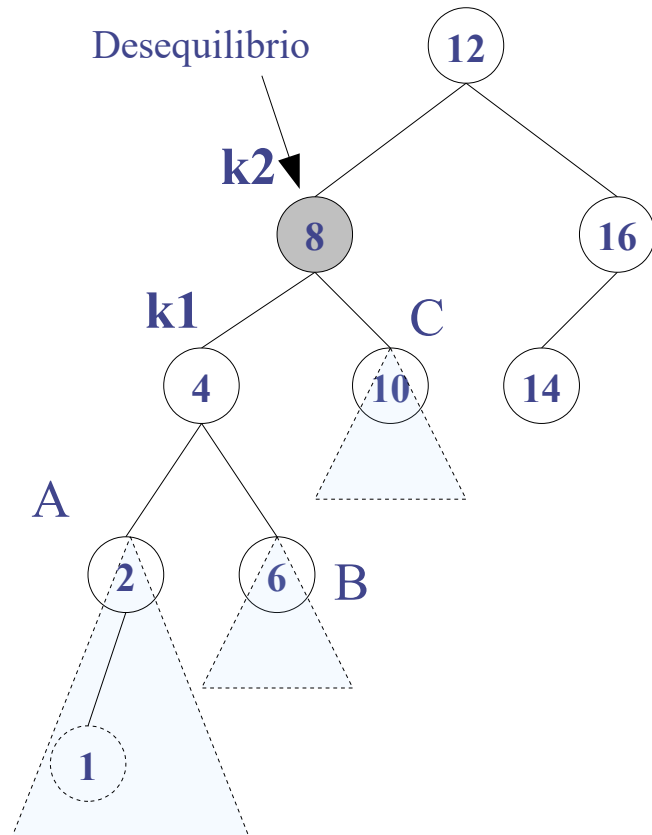
El nodo **X** es donde se produce el desequilibrio

Rotación simple a la derecha (caso 1)



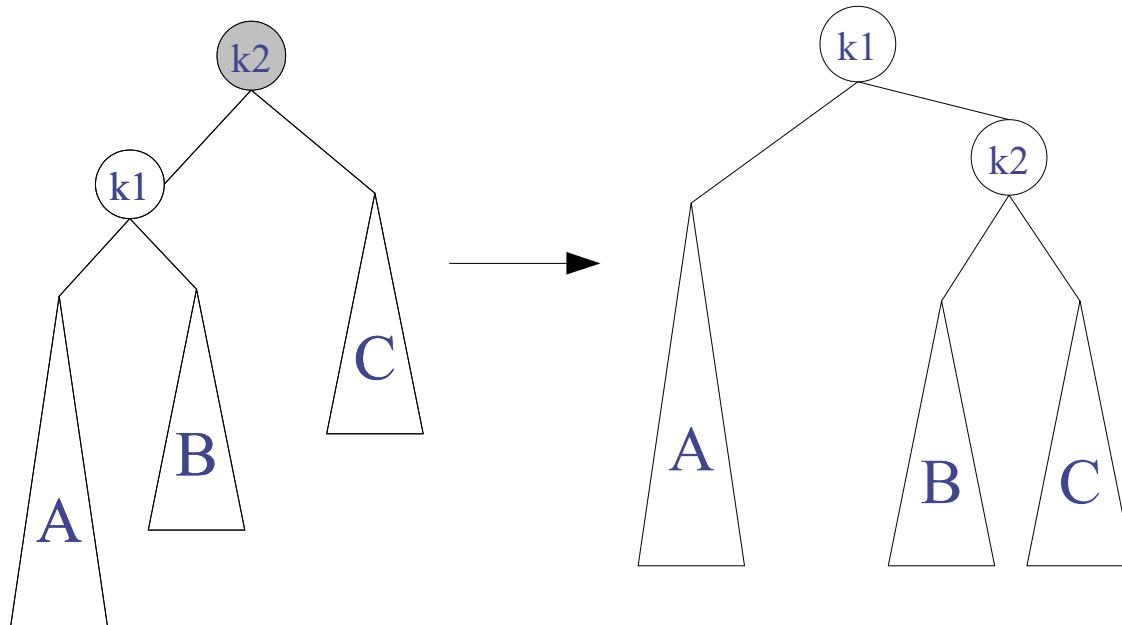
Rotación simple a la derecha

Ejemplo (insertar el 1)

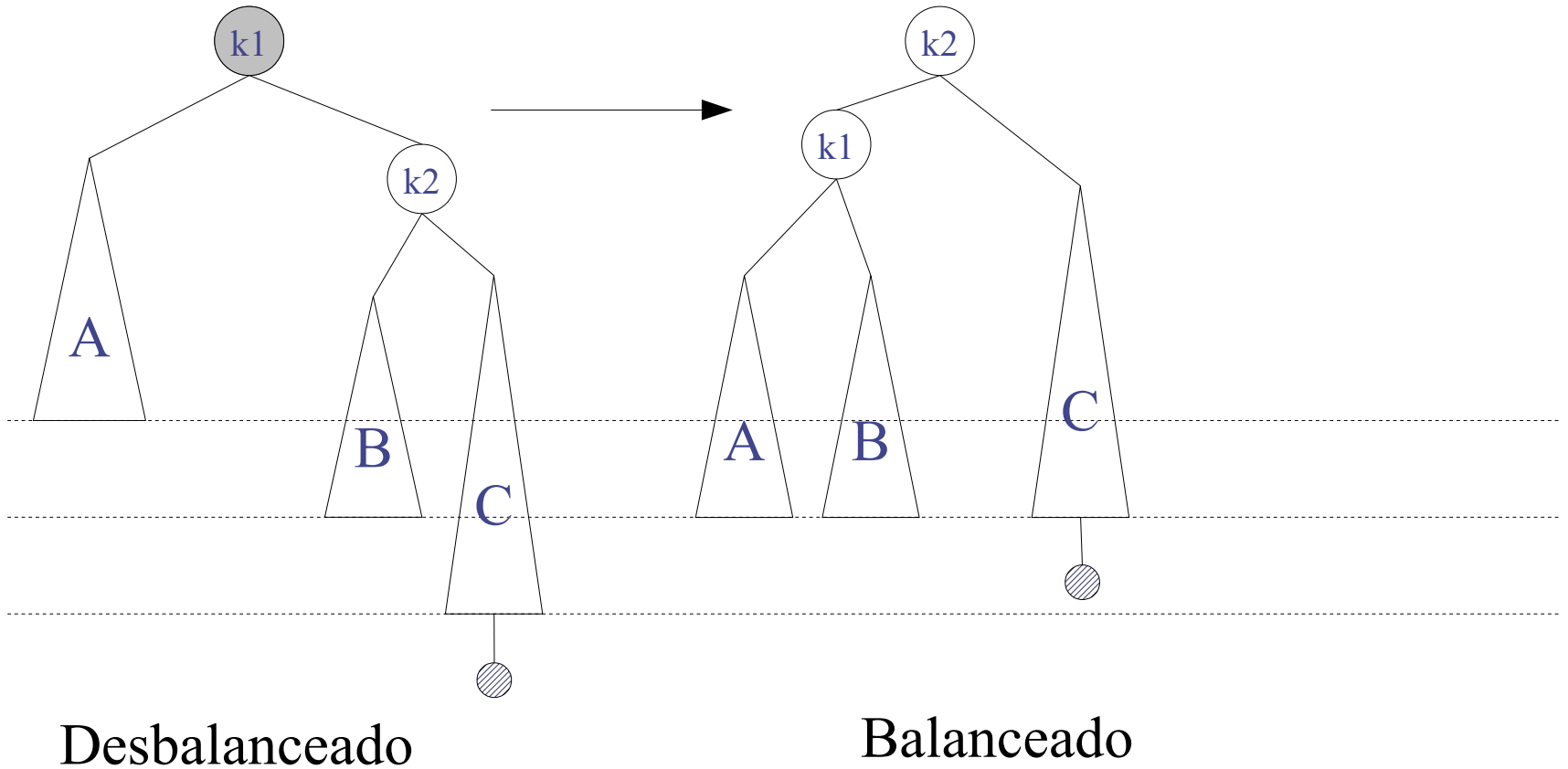


Rotación simple a la derecha (código)

```
private static AvlNode rotateWithLeftChild( AvlNode k2 )
{
    AvlNode k1 = k2.left;
    k2.left    = k1.right;
    k1.right   = k2;
    k2.height  = max( height(k2.left), height(k2.right)) + 1;
    k1.height  = max( height( k1.left ), k2.height ) + 1;
    return k1;
}
```

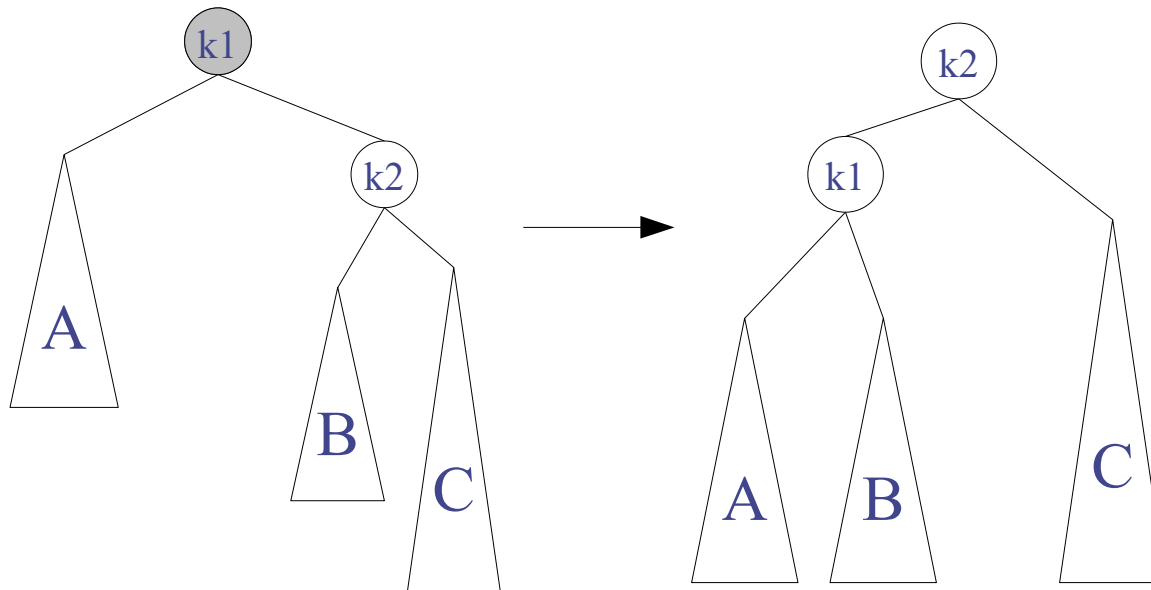


Rotación simple a la izquierda (caso 4)



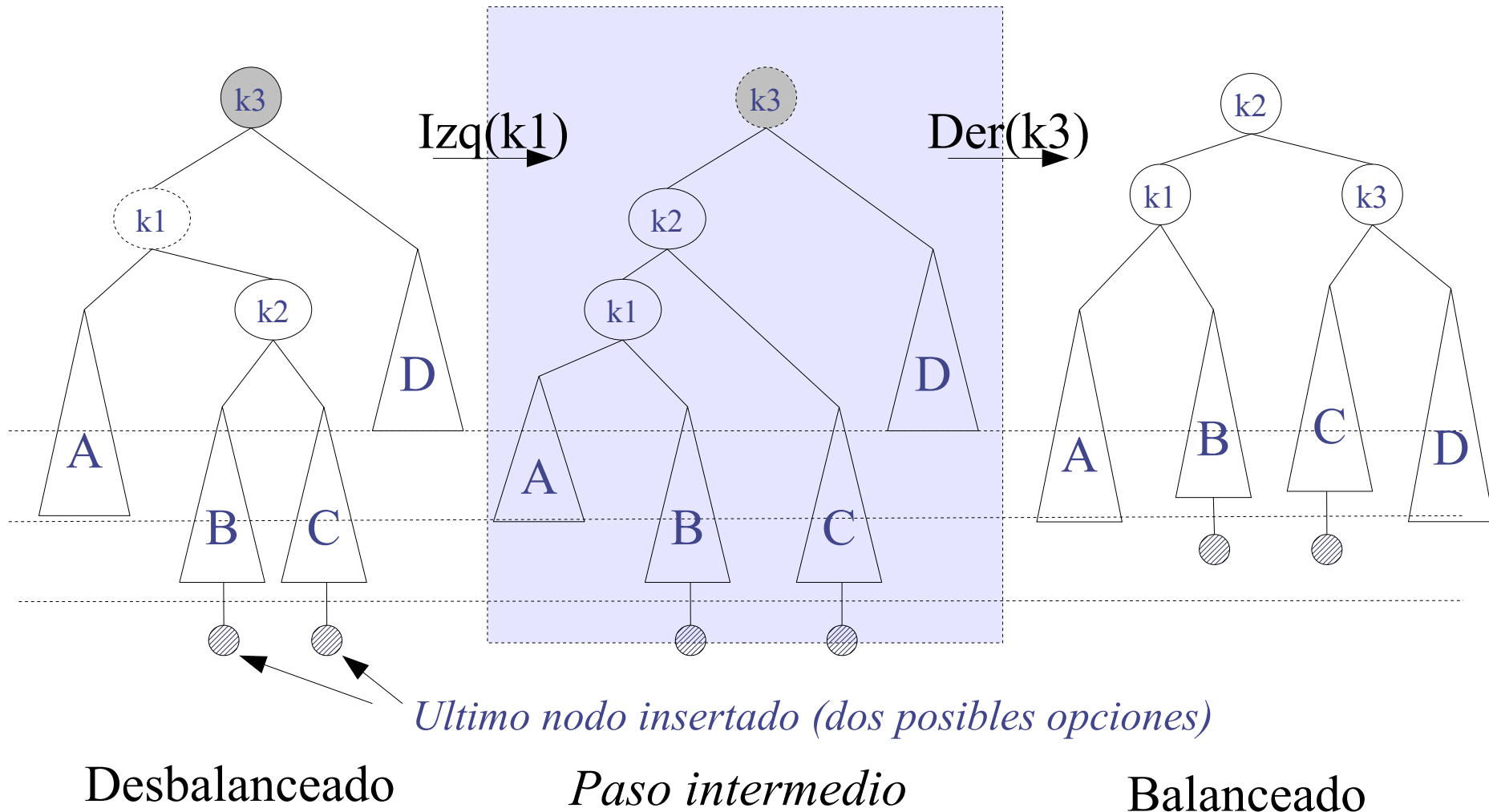
Rotación simple a la izquierda caso 4

```
private static AvlNode rotateWithRightChild( AvlNode k1 )  
{  
    AvlNode k2 = k1.right;  
    k1.right    = k2.left;  
    k2.left     = k1;  
    k1.height   = max( height(k1.left), height(k1.right) ) + 1;  
    k2.height   = max( height( k2.right ), k1.height ) + 1;  
    return k2;  
}
```



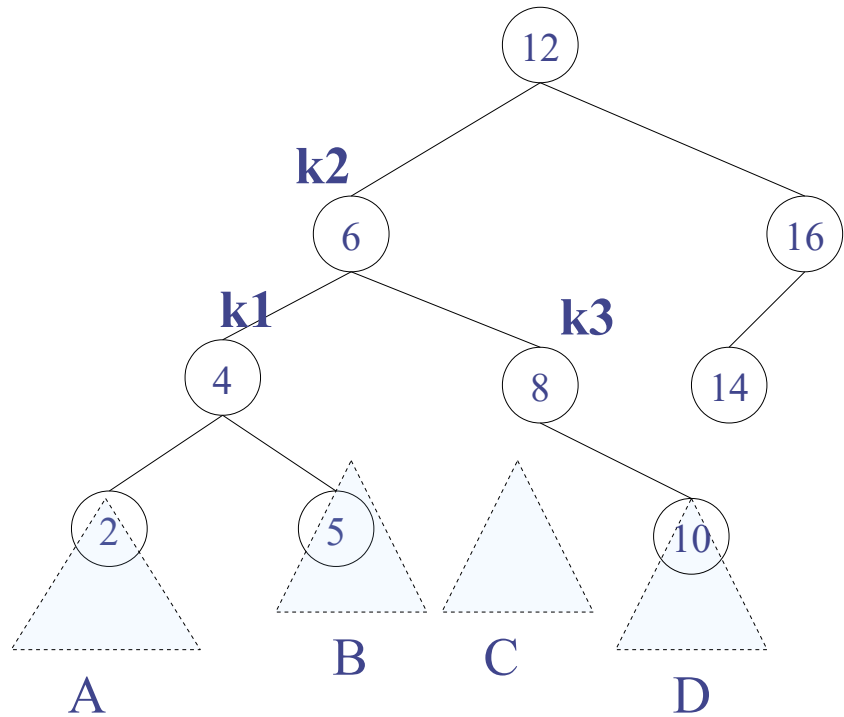
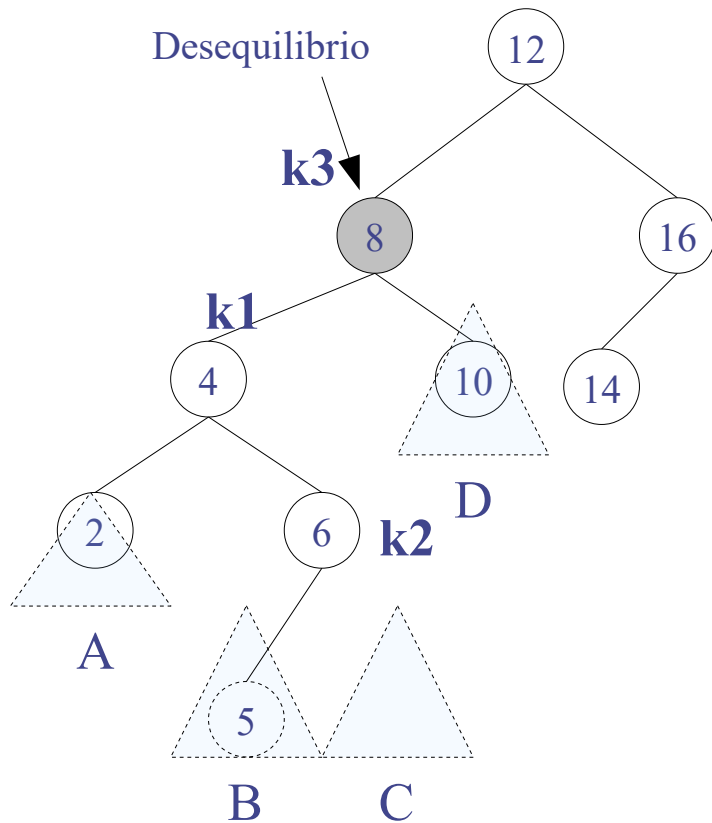
Rotación doble de izq. a derecha

(caso 2)



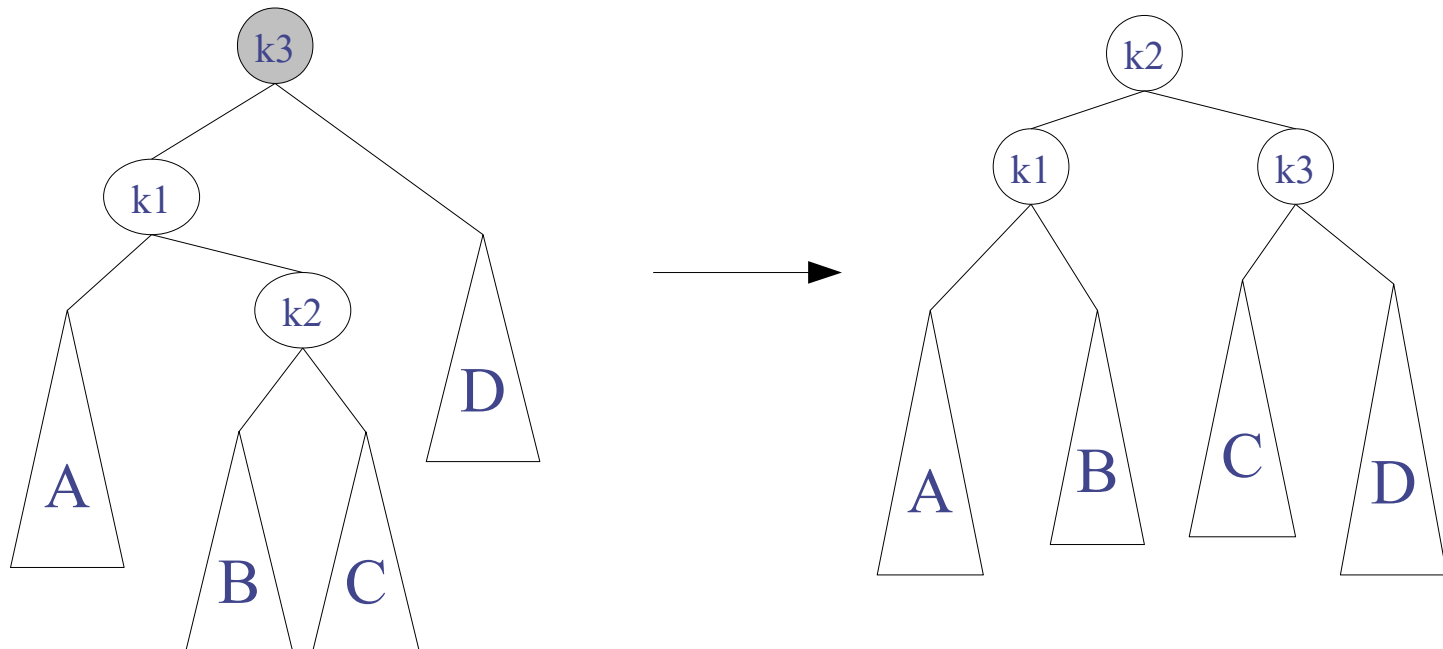
Rotación doble de izq a derecha

Ejemplo (insertar el 5)



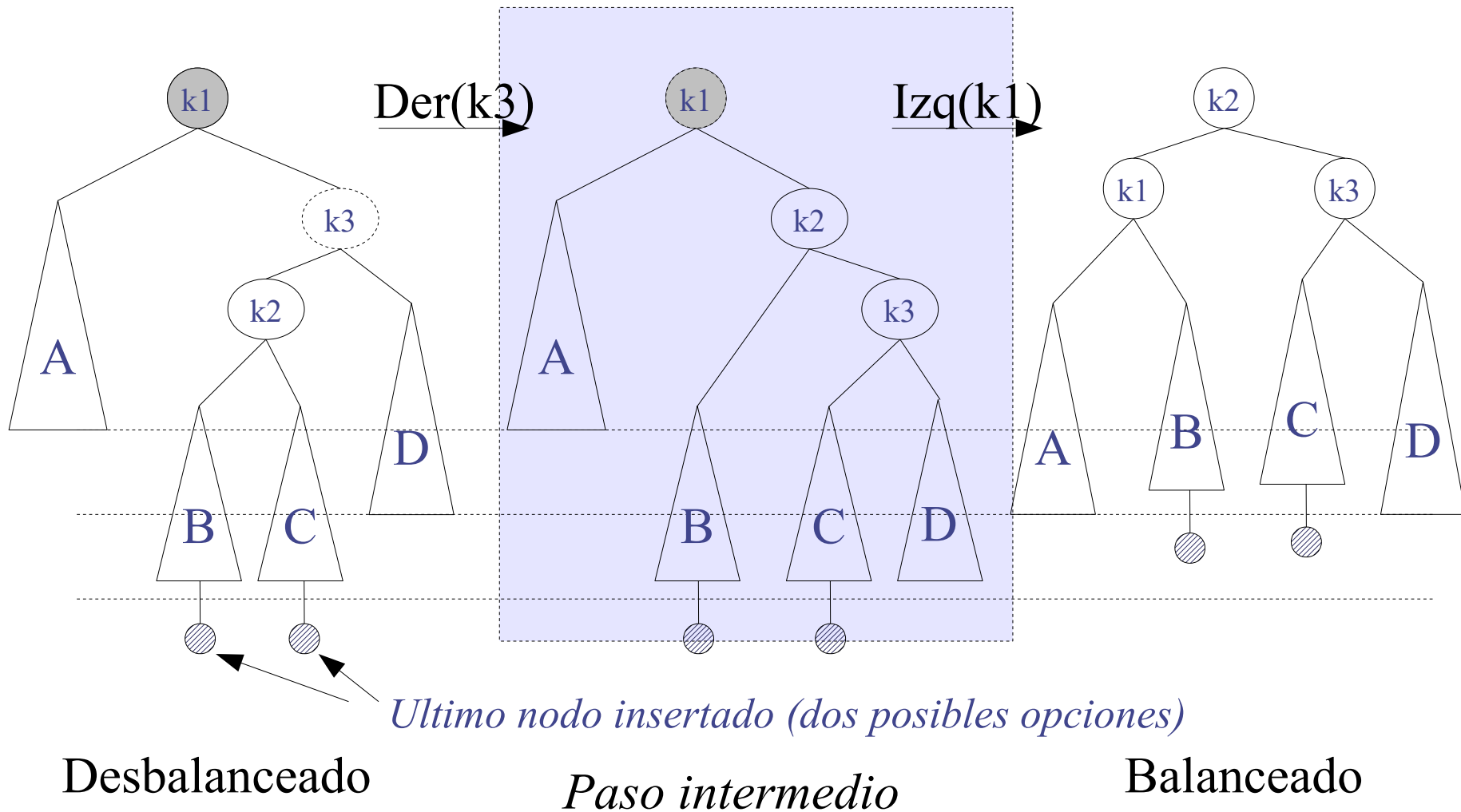
Rotación doble de izq a derecha

```
private static AvlNode doubleWithLeftChild( AvlNode k3 )  
{  
    k3.left = rotateWithRightChild( k3.left ); /* Rot izq */  
    return rotateWithLeftChild( k3 ); /* Rot. Der */  
}
```



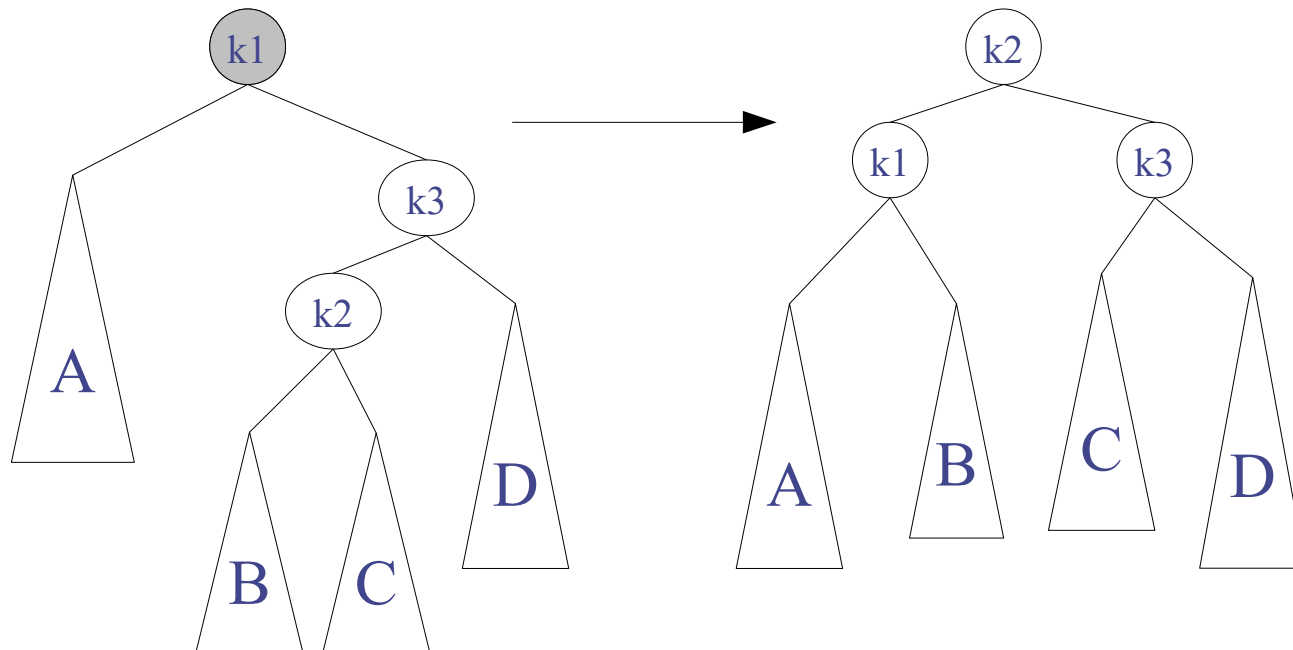
Rotación doble de derecha a izq.

(caso 3)



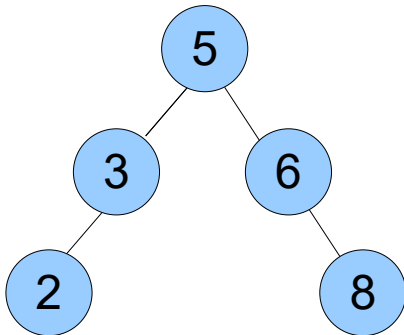
Rotación doble de derecha a izq.

```
private static AvlNode doubleWithRightChild( AvlNode k1 )  
{  
    k1.right = rotateWithLeftChild( k1.right ); /* Rot. Der. */  
    return rotateWithRightChild( k1 ); /* Rot. Izq */  
}
```

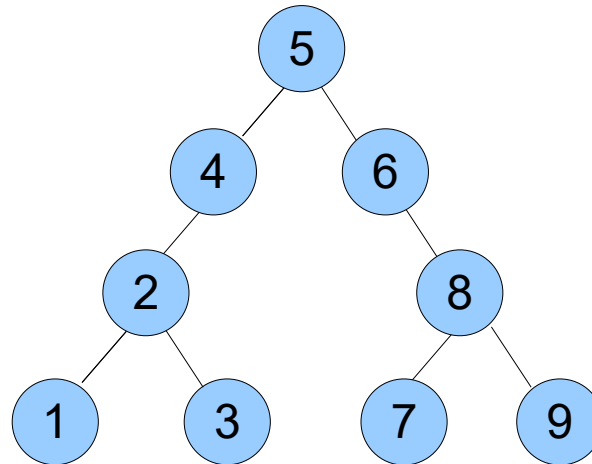


Ejercicio #1 para clase

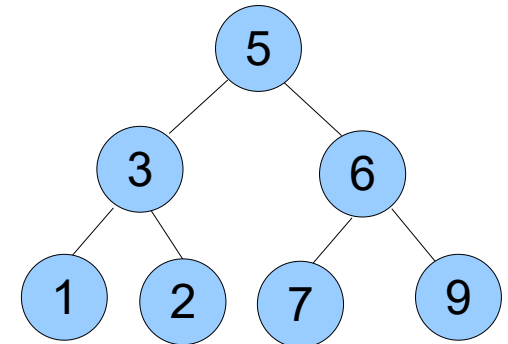
- ¿Cuál de estos árboles binarios es AVL?



a)



b)



c)

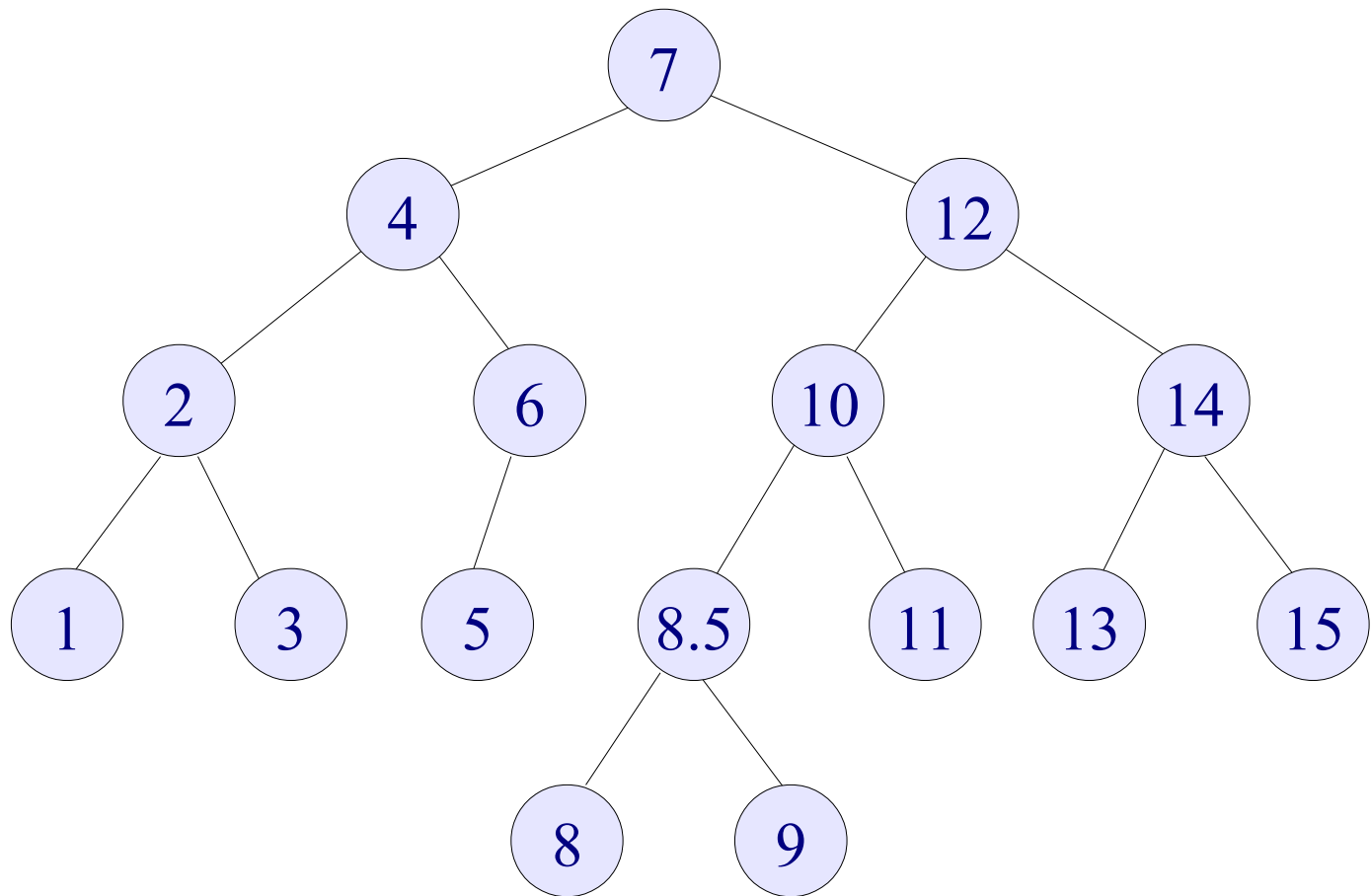
Ejercicio #2 para clase

- ¿Cuál es la máxima y mínima altura para un AVL de 3, 5 y 7 nodos?
- Si un AVL tiene altura 4, ¿Cuál es el máximo número de nodos y el mínimo número de nodos que puede tener?

Ejercicio #3 para clase

- Insertar consecutivamente en un AVL los valores de 1 al 7 (1, 2, 3, 4, 5, 6, 7)
- Luego insertar de 8 al 15 en forma inversa (15, 14, 13, 12, 11, 10, 9, 8)
- Al final insertar el valor 8.5
- Indicar el tipo de rotación que fue requerido (si es el caso)
- Mostrar el AVL resultante.

Árbol final del ejercicio #3



Código de inserción en un AVL

```
private AvlNode insert( Comparable x, AvlNode t )
{
    if( t == null )
        t = new AvlNode( x, null, null );
    else if( x.compareTo( t.dato ) < 0 ) {
        t.left = insert( x, t.left );
        if( height( t.left ) - height( t.right ) == 2 )
            if( x.compareTo( t.left.dato ) < 0 )
                t = rotateWithLeftChild( t ); /* Caso 1 - Rot. S. derecha */
            else
                t = doubleWithLeftChild( t ); /* Caso 2 - Rot. D. izq-der */
    }
    else if( x.compareTo( t.dato ) > 0 ) {
        t.right = insert( x, t.right );
        if( height( t.right ) - height( t.left ) == 2 )
            if( x.compareTo( t.right.dato ) > 0 )
                t = rotateWithRightChild( t ); /* Caso 4 - Rot. S. izq */
            else
                t = doubleWithRightChild( t ); /* Caso 3 - Rot. D. der-izq */
    }
    else ; // Duplicado; no hago nada
    t.height = max( height( t.left ), height( t.right ) ) + 1;
    return t;
}
```

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

Operación de Borrado en AVL

- Es lo mismo que en un BST, sin embargo puede quedar desbalanceado.
- Similar a la inserción, comenzamos por el nodo removido chequeamos todos los nodos en el camino hacia la raíz para detectar algún nodo desbalanceado.
- Usar rotación simple o doble si es necesaria para balancear el árbol.
- Se necesita continuar la búsqueda por nodos no balanceados en todo el camino hacia la raíz.

Borrar X en un árbol AVL

- Borrado:

- **Caso 1:**

- Si X es una hoja, borrar X

- **Caso 2:**

- Si X tiene 1 hijo, usarlo para reemplazar X

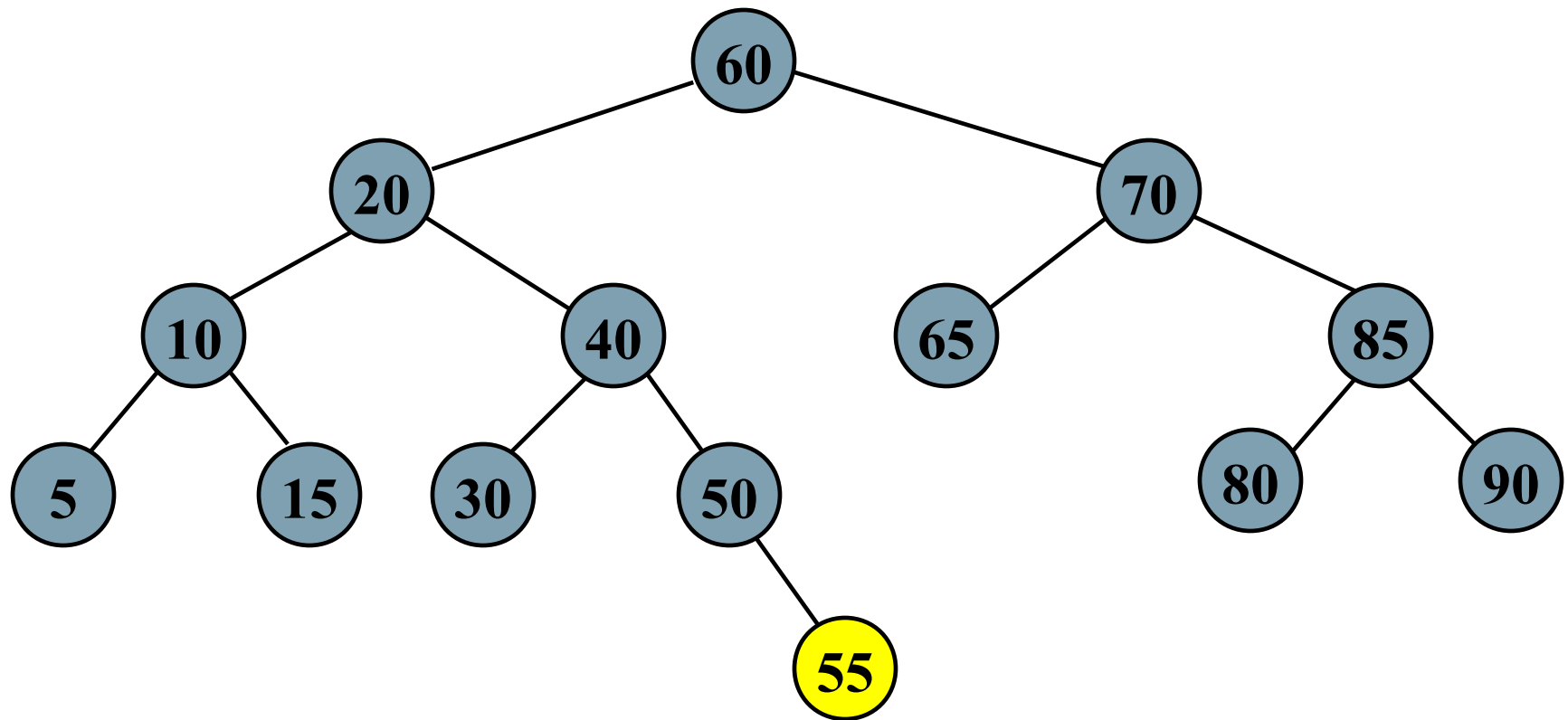
- **Caso 3:**

- Si X tiene 2 hijos, reemplazar X con su predecesor inorden (el mayor del árbol izquierdo) (y borrarlo recursivamente). A partir de allí actualizar las alturas y verificar la condición de AVL.

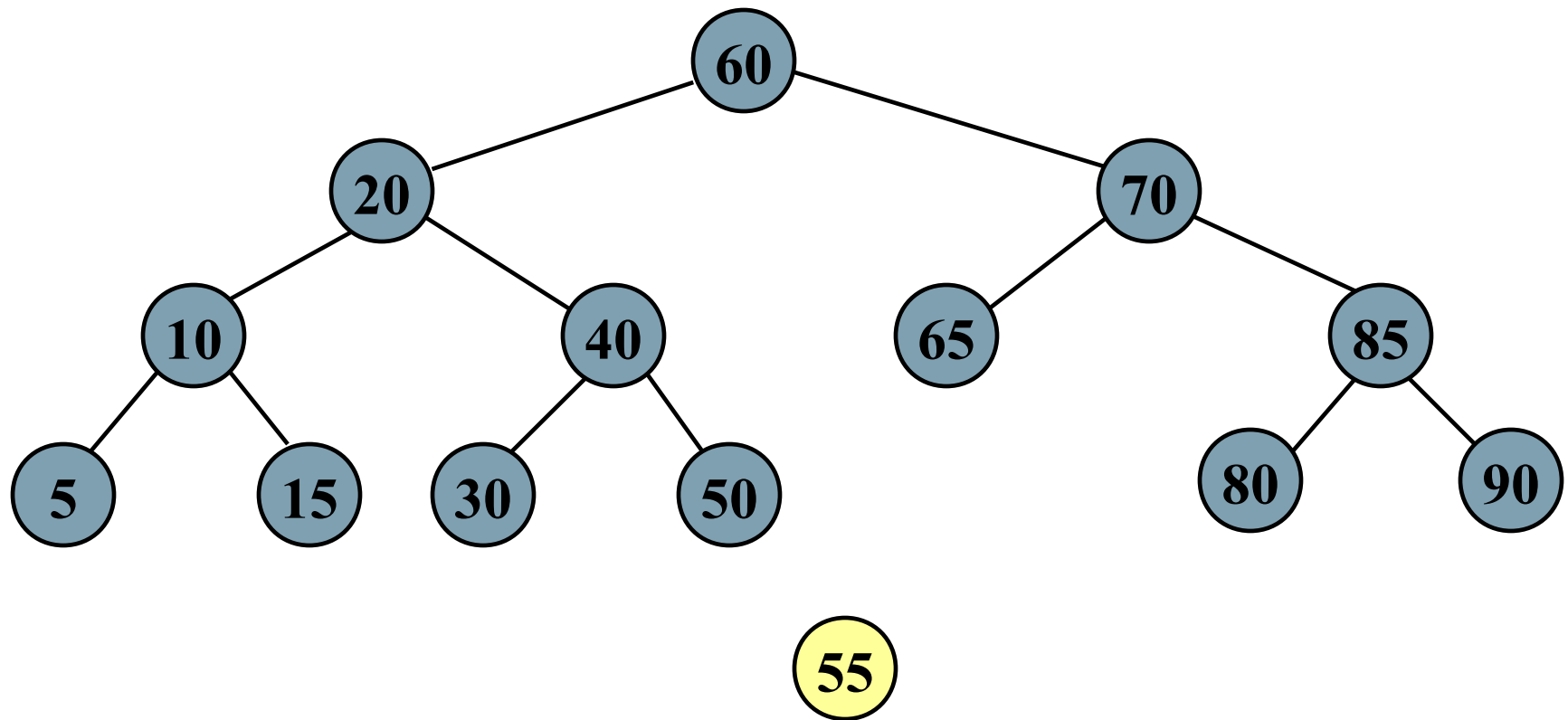
- Rebalancear

Ejemplo de borrado

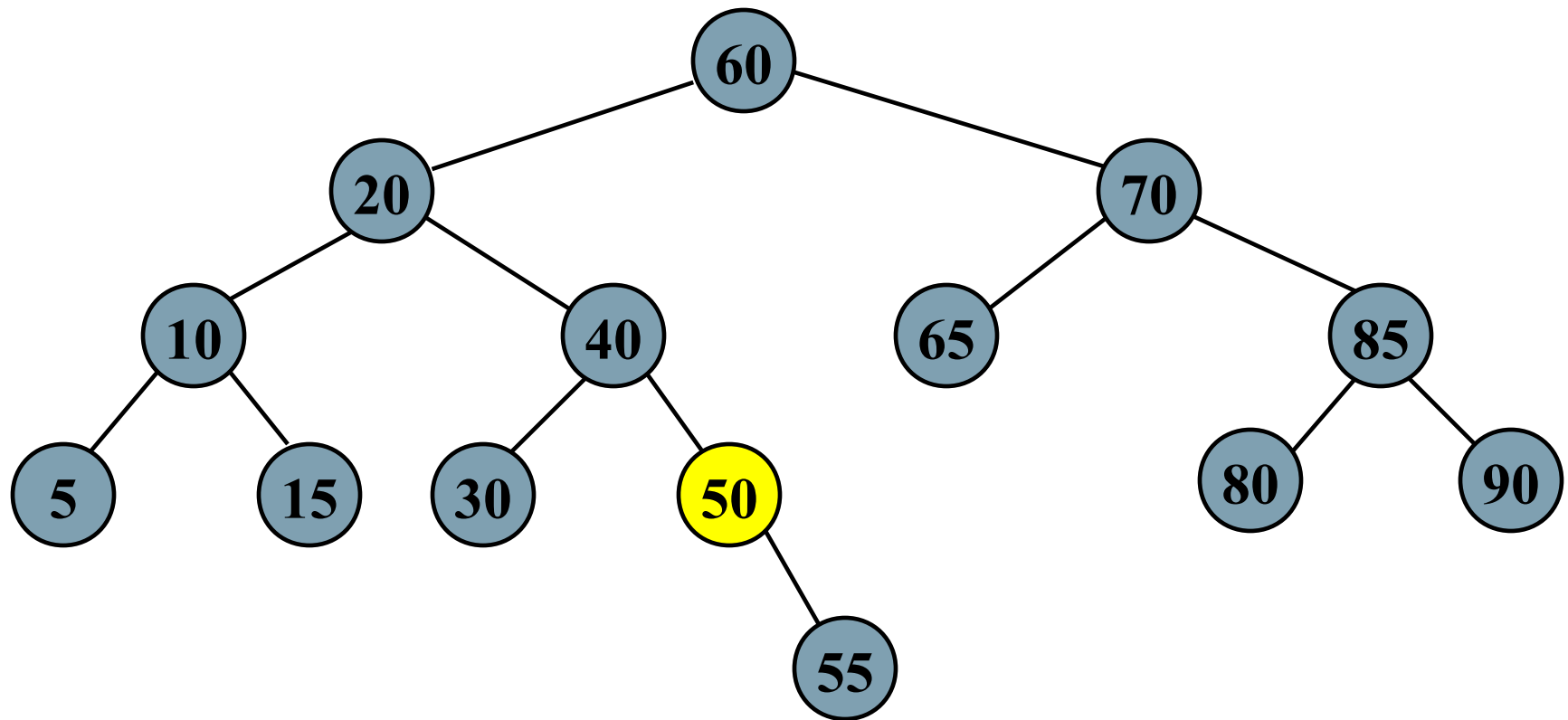
Borrar 55 (**caso 1**)



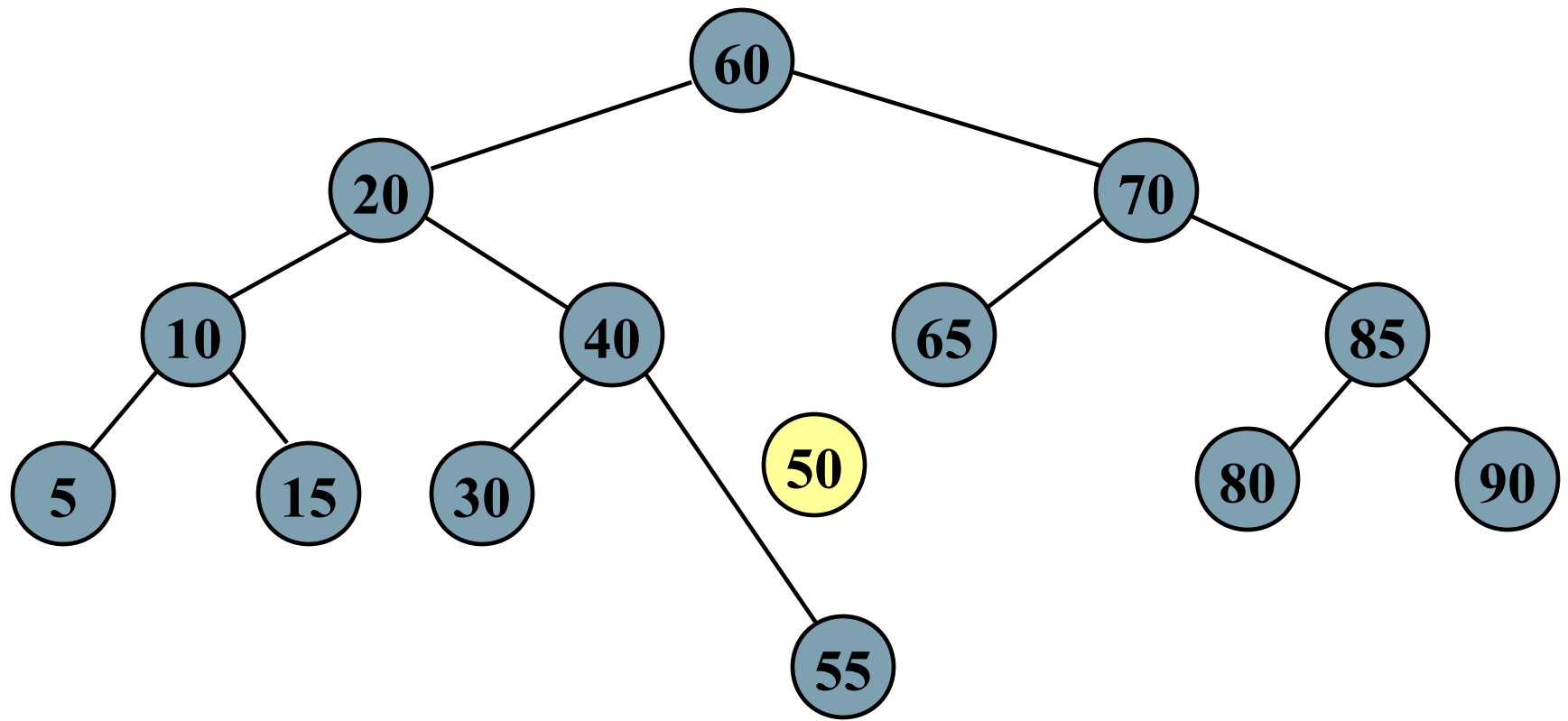
Borrar 55 (caso 1)



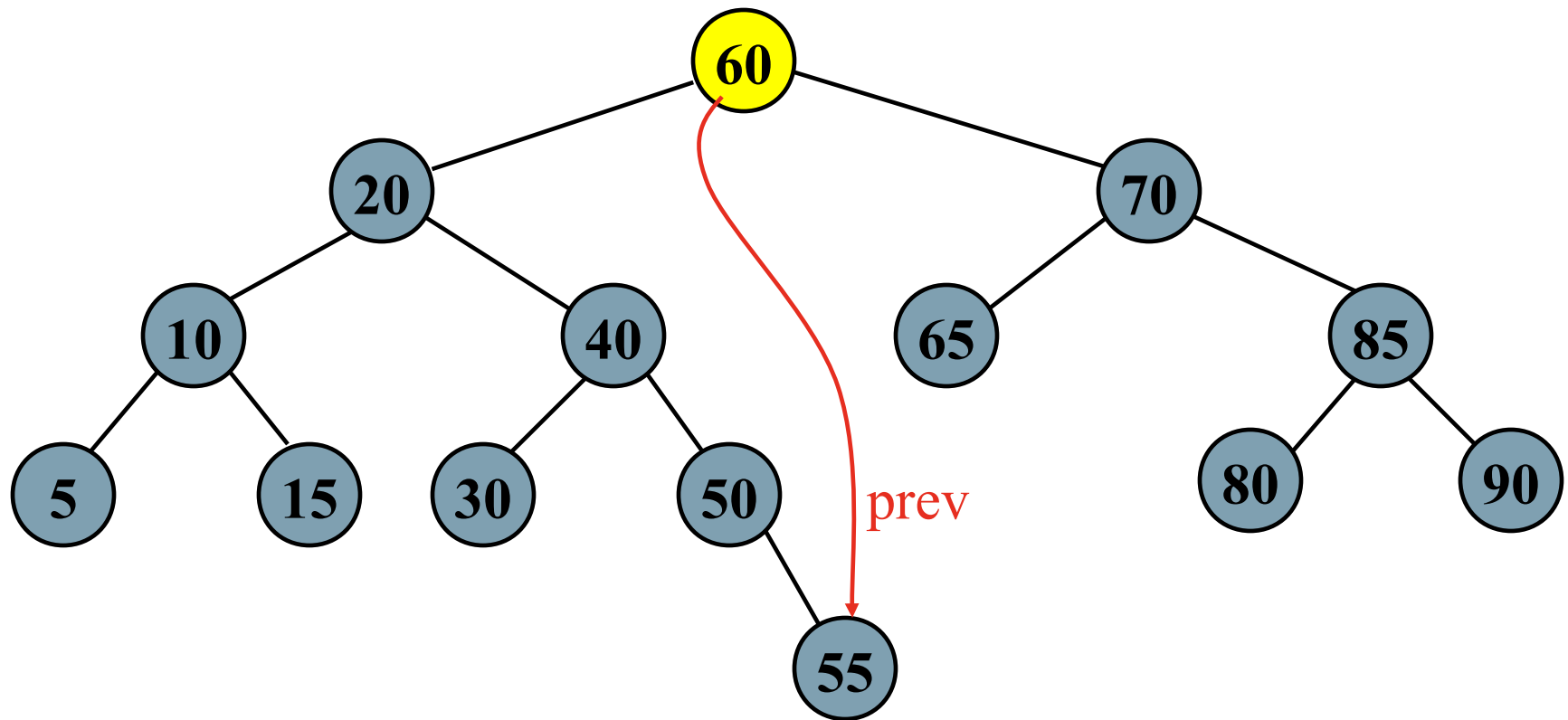
Borrar 50 (caso 2)



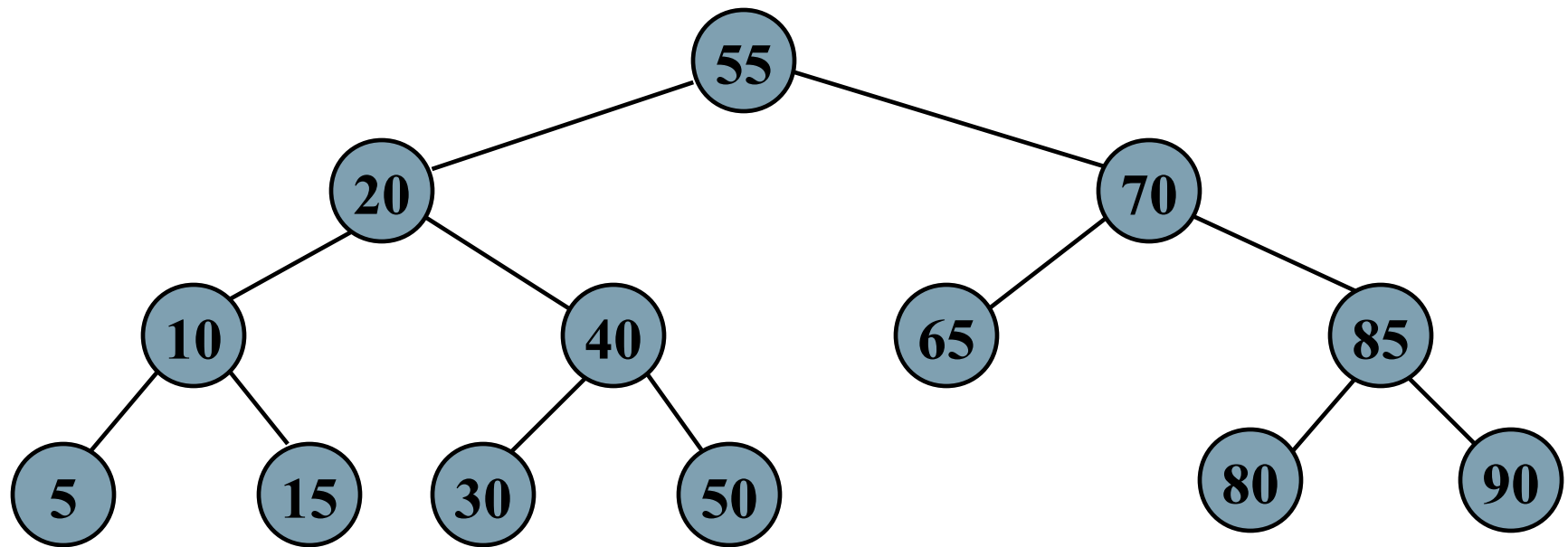
Borrar 50 (caso 2)



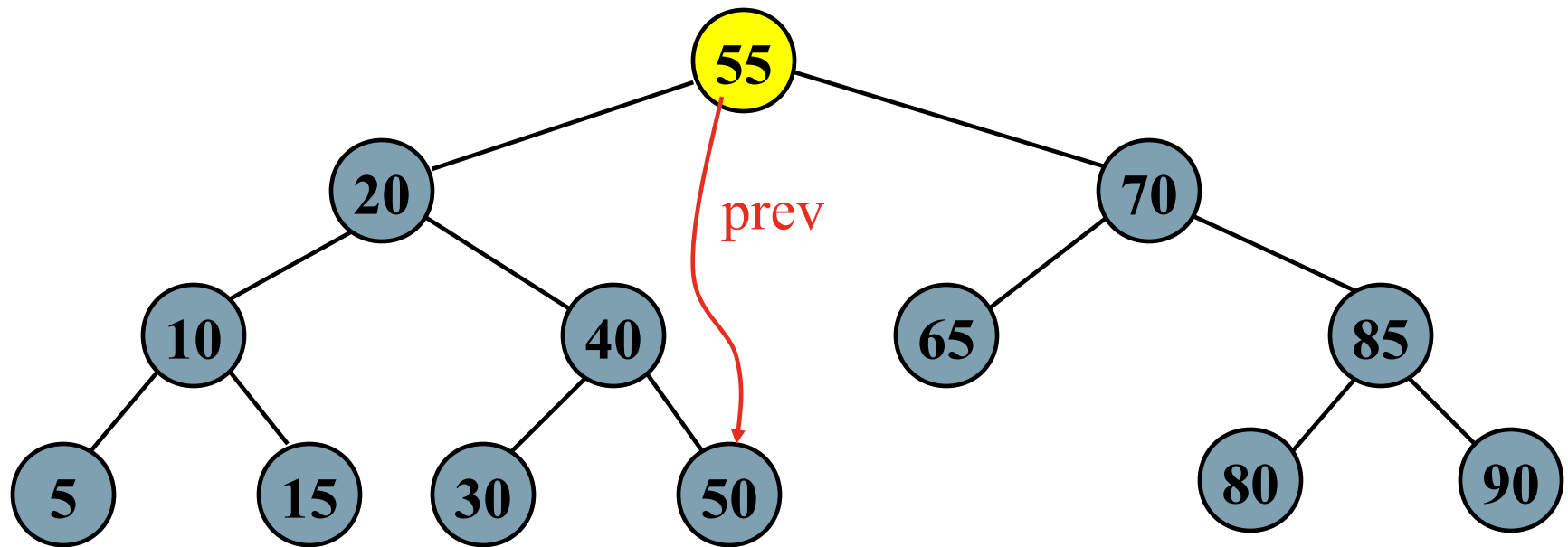
Borrar 60 (**caso 3**)



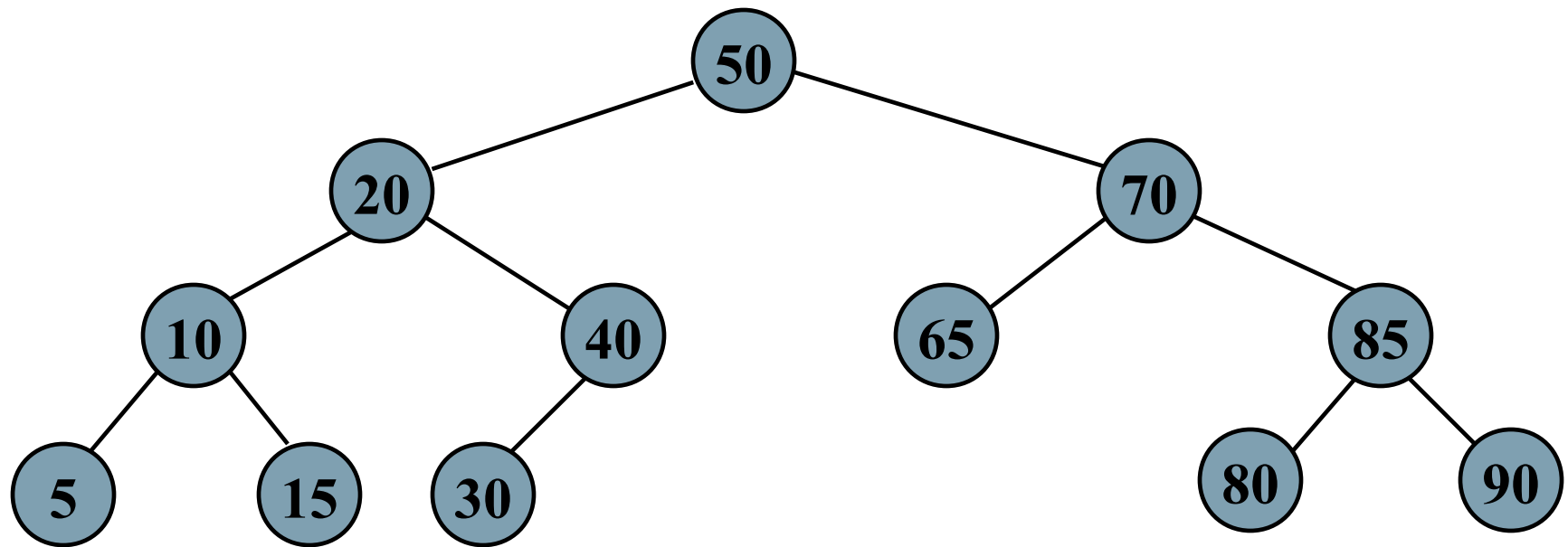
Borrar 60 (**caso 3**)



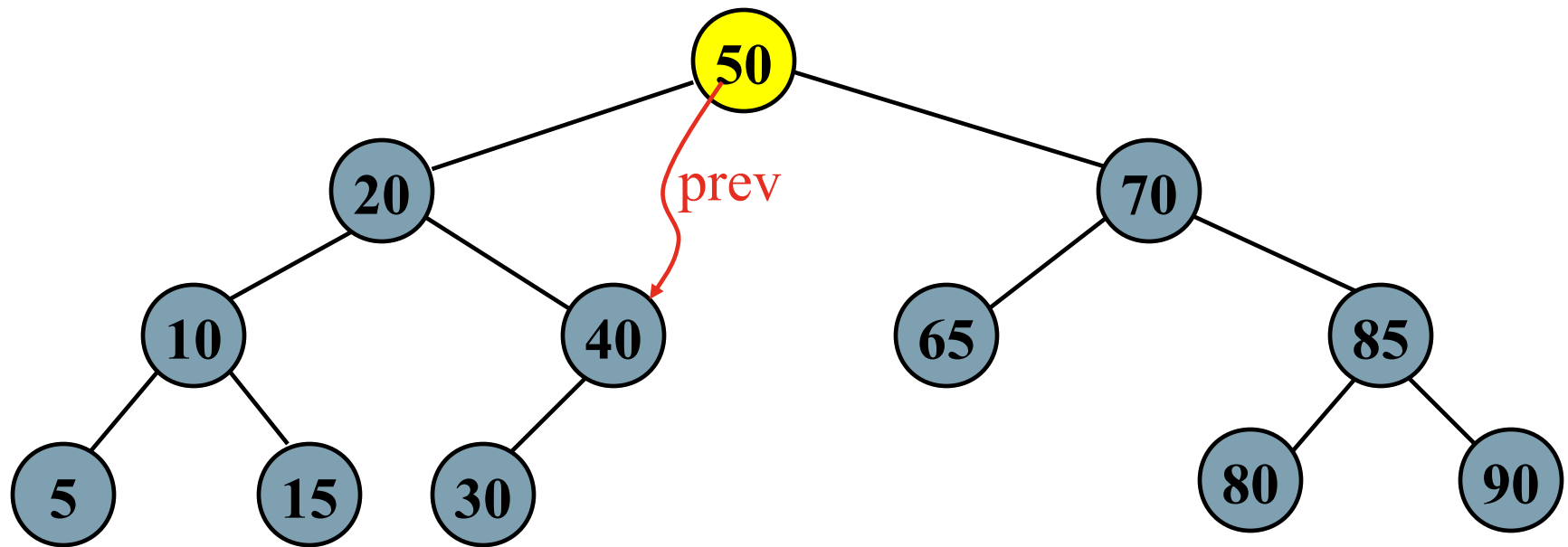
Borrar 55 (caso 3)



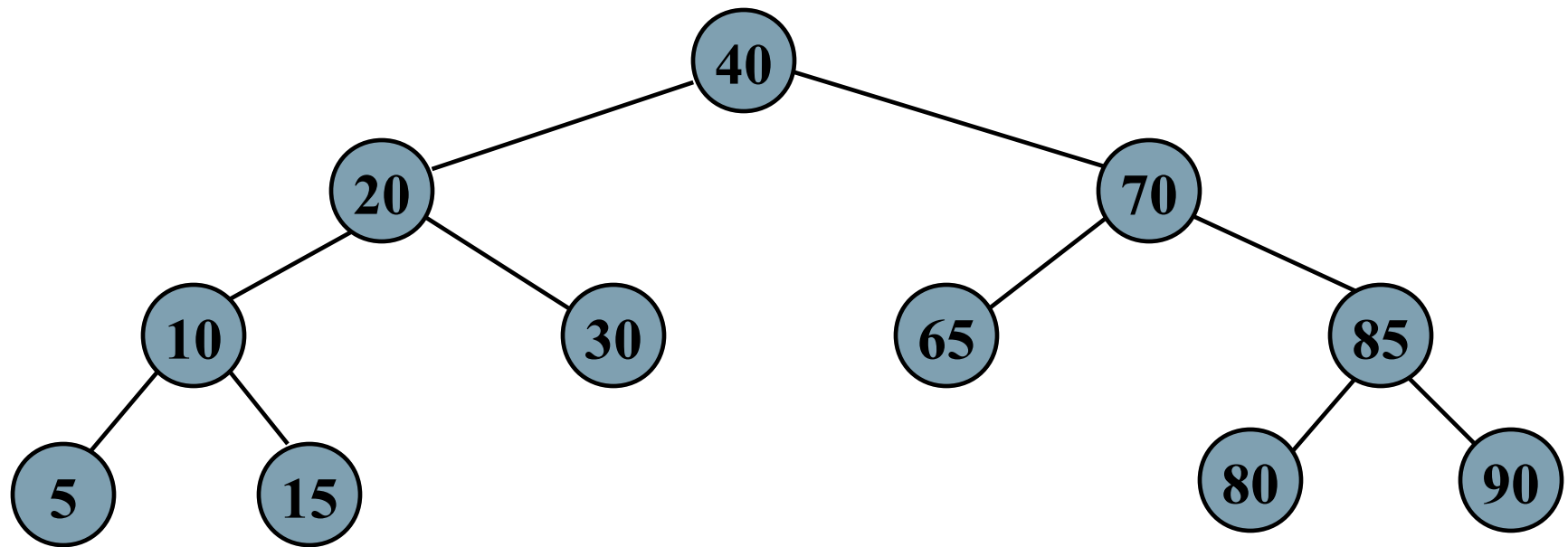
Borrar 55 (caso 3)



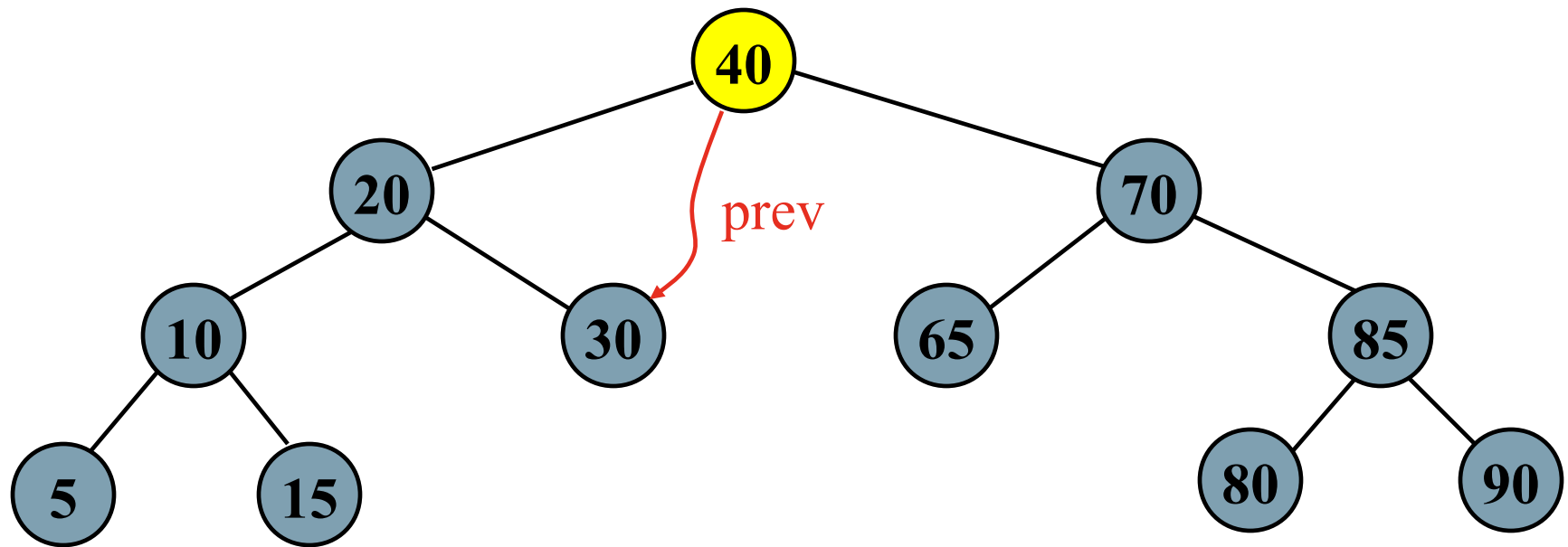
Borrar 50 (**caso 3**)



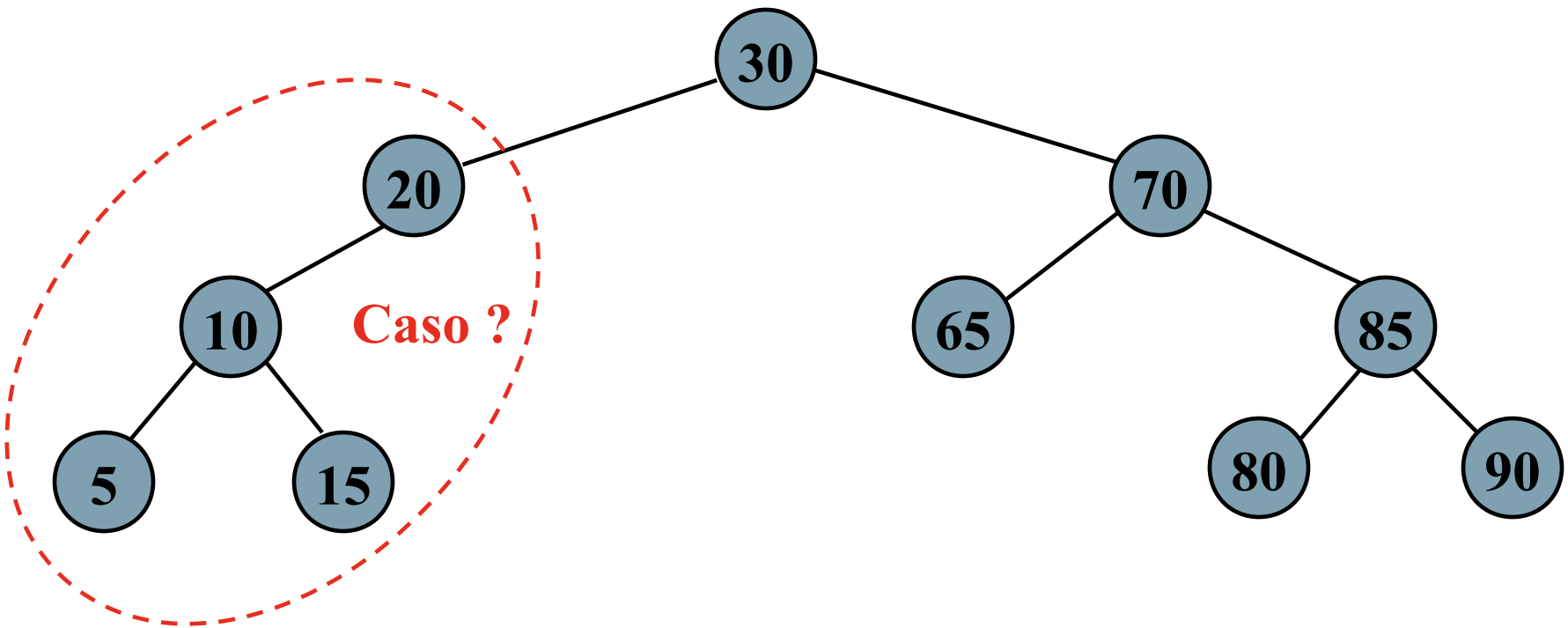
Borrar 50 (**caso 3**)



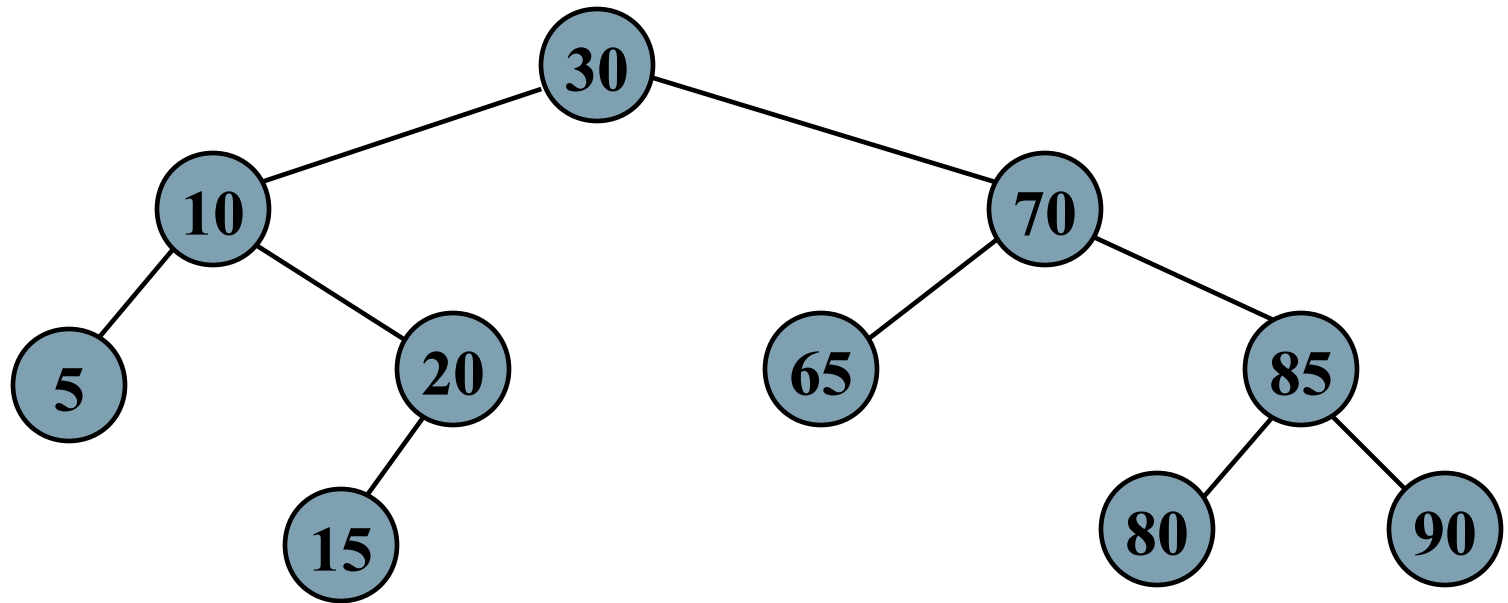
Borrar 40 (**caso 3**)



Borrar 40 : **Rebalancear**



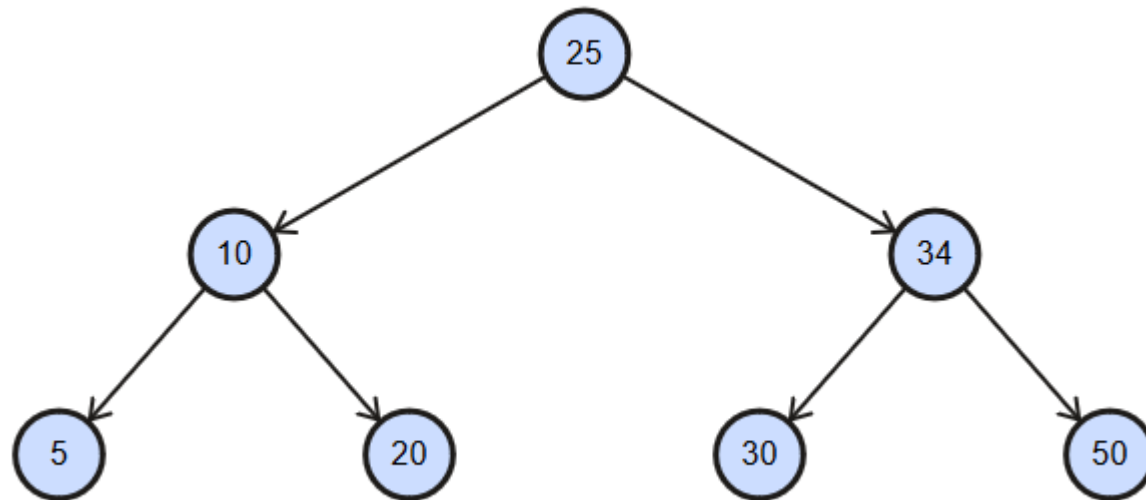
Borrar 40: luego de rebalancear



Rotación simple es preferida

Ejercicio #4

Dado el siguiente árbol AVL, borrar los elementos 34, 30, 50 y 5. Indicar el tipo de rotación requerida por cada borrado, si fuera el caso. Mostrar el árbol luego de cada paso.



Puede verificar su solución en:

<http://www.cs.armstrong.edu/liang/animation/web/AVLTree.html>

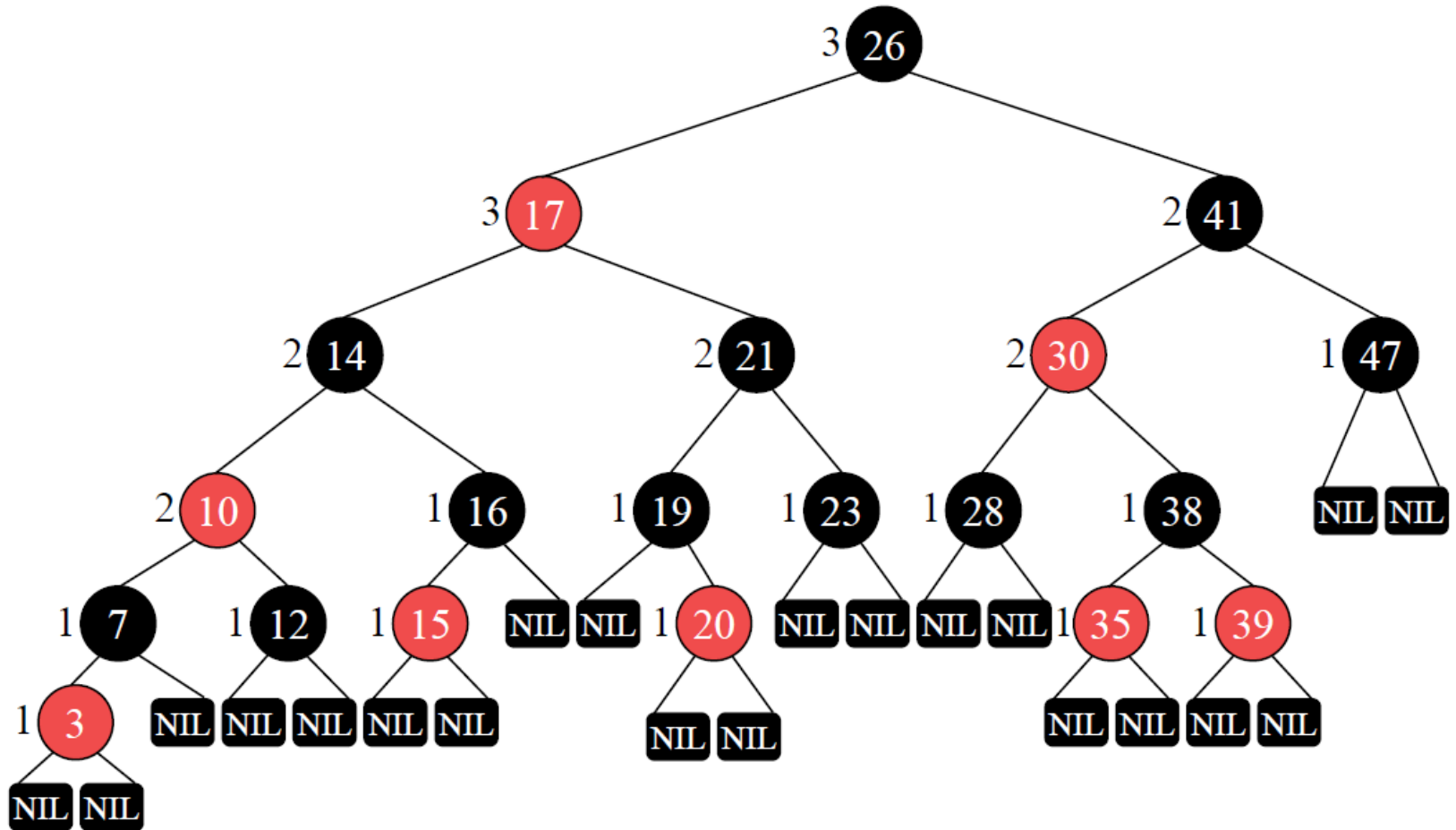
Tiempo de ejecución en AVL

- Reestructuración simple es $O(1)$
 - Intercambio de referencias en la lista enlazada
- Buscar está en $O(\log n)$
 - La altura del árbol está en $O(\log n)$, no se necesita reestructurar
- Insertar está en $O(\log n)$
 - Búsqueda inicial está en $O(\log n)$
 - Reestructurar el árbol en forma ascendente, manteniendo las alturas esta en $O(\log n)$
- Eliminar está $O(\log n)$
 - ❖ Búsqueda inicial está en $O(\log n)$
 - ❖ Reestructurar el árbol en forma ascendente, manteniendo las alturas esta en $O(\log n)$

Árbol Rojinegro (*Red-Black Tree*)

- Es un BST
- Dato adicional: color (Rojo o Negro)
- Un BST es un árbol rojinegro si:
 - ❖ R1: Cada nodo es rojo o negro
 - ❖ R2: La raíz es negra
 - ❖ R3: Cada hoja vacía (NIL) es negra
 - ❖ R4: Si un nodo es rojo, entonces ambos hijos son negros.
 - ❖ R5: Por cada nodo, todos los caminos del nodo a sus hojas descendientes contiene el mismo número de nodos negros.

Árbol rojinegro (ejemplo)



Altura de un árbol rojinegro

Ver en CLRS – Lema 13.1

■ Definiciones:

- ❖ Altura negra de un nodo x , ***an(x)***:
 - ♦ Es el número de nodos negros desde x (sin incluir x) hasta cualquier hoja descendente de x .
- ❖ Altura negra de un árbol: la altura negra de su raíz.

- **Lema:** Todo árbol rojinegro con n nodos internos tiene una altura menor o igual a **$2 \log(n+1)$** .

Demostración:

El subárbol con raíz x contiene al menos $2^{an(x)} - 1$ nodos internos. Por inducción en la altura de x .

- ❖ Caso $altura(x) = 0$. x debe ser una hoja(NIL) y el subarbol x contiene en efecto $2^{an(x)} - 1 = 2^0 - 1 = 0$ nodos internos.
- ❖ Inducción: considerar un nodo interno x con dos hijos. Sus hijos tienen altura negra $an(x)$ ó $an(x)-1$ dependiendo si es rojo o negro, respectivamente.

Los dos hijos de x tienen al menos $2^{an(x)-1} - 1$ nodos internos.

Por tanto el subárbol con raíz x tiene al menos:

$$(2^{an(x)-1} - 1) + (2^{an(x)-1} - 1) + 1 = \mathbf{2^{an(x)} - 1 \text{ nodos internos}}$$

- Sea h la altura del árbol

Por definición, al menos la mitad de los nodos de cualquier camino de la raíz a una hoja (sin incluir la raíz) deben ser negros.

Por tanto, la altura negra de la raíz debe ser al menos $h/2$, por tanto (reemplazando por el lema) :

$$n \geq 2^{h/2} - 1 \Rightarrow \log(n+1) \geq h/2 \Rightarrow \mathbf{h \leq 2\log(n+1)}$$

- **Por tanto:** las operaciones de búsqueda pueden implementarse en tiempo $O(\log n)$ para los árboles rojinegros con n nodos.

Árbol rojinegro

- Principal ventaja sobre AVL
 - ❖ Las rutinas de inserción y eliminación pueden efectuarse con un único recorrido descendente.
- Su desventaja ante el AVL es que las rutinas requieren más casos especiales y cuidados en la implementación

Fuentes y Bibliografía

- Mark A. Weiss. *Estructura de datos en Java. Compatible con Java 2*. Pearson. AW. 2000.
- M. Goodrich and R. Tamassia. *Data Structures & Algorithms in Java*. Fourth Edition. J. Wiley. 2005.
- Levitin, Anany. *Introduction to the Design and Analysis of Algorithms*. 2nd Edition. Pearson. AW. 2007.
- Cormen T., Leiserson C., Rivest R and Stein C. *Introduction to algorithms*. 3rd. Edition. 2009. Capitulo 13 (Red-Black Trees).
- Curso "6.006 Introduction to algorithms" MIT.
<http://courses.csail.mit.edu/6.006/spring11/>