Tema 5. ÁRBOLES BINARIOS DE BÚSQUEDA

ESTRUCTURAS DE DATOS Y ALGORITMOS II

Grado en Ingeniería Informática

María José Polo Martín

mjpolo@usal.es

curso 2018-2019

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

DEPARTAMENTO DE iNFORMÁTiCA Y AVTOMÁTiCA

Tema 5. Árboles Binarios de Búsqueda

1 Nivel abstracto o de definición

2 Nivel de representación

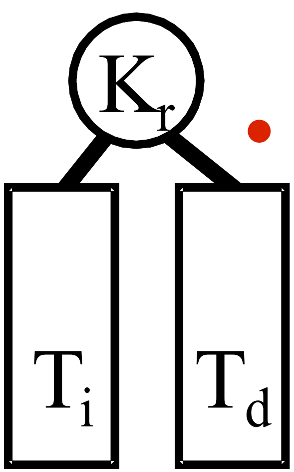
- Búsqueda
- Inserción
- Eliminación
- Análisis del caso medio

3 Árboles Balanceados

- Inserción y equilibrio del árbol
- Rotaciones
- Eliminación y equilibrio del árbol

1 NIVEL ABSTRACTO O DE DEFINICIÓN

- Aplicación común de árboles binarios: RECUPERACIÓN DE INFORMACIÓN $\Rightarrow$ Árboles Binarios de BÚSQUEDA
- Requisito: los nodos del árbol deben estar ordenados según el valor de alguno de sus campos de información (clave o llave)
- **Definición formal:** árbol binario que o bien es nulo o cada nodo contiene una clave que satisface las siguientes condiciones:
  - Todas las claves, si las hay, en el subárbol izquierdo de la raíz preceden a la clave de la raíz
  - La clave de la raíz precede a todas las claves, si las hay, que contiene el subárbol derecho
  - Los subárboles izquierdo y derecho de la raíz son también árboles binarios de búsqueda



$$\mathbf{K_i < K_r \;\; \forall\, K_i \in T_i}$$

$$\mathbf{K_r < K_d \;\; \forall\, K_d \in T_d}$$

(Esta definición puede modificarse para admitir claves duplicadas)

Características

- Existen diferentes formas de ordenar un conjunto de claves para conseguir un árbol binario de búsqueda
- La estructura de un árbol binario de búsqueda particular está determinada por el orden en que se insertan los nodos en el árbol
- Una vez decidida la clave que se inserta en primer lugar las propiedades del árbol determinan donde deben insertarse las siguientes
- Un nuevo nodo siempre se inserta como nodo hoja a no ser que se permita reestructurar el árbol durante el proceso de inserción
- El recorrido en orden de un árbol binario de búsqueda da lugar a una clasificación ascendente de los nodos según el valor de su campo clave

Operaciones básicas sobre árboles binarios de búsqueda

- BÚSQUEDA(k, A, n) : busca un nodo con valor de clave k en el árbol binario de búsqueda A y devuelve la posición de ese nodo en el árbol si lo encuentra o nulo en otro caso

- INSERTAR(n, A): añade el nodo n al árbol binario de búsqueda A. Después de la inserción A continuará siendo un árbol binario de búsqueda

- ELIMINAR(k, A): suprime el nodo con valor k en su campo clave del árbol binario de búsqueda A si existe. Si no existe un nodo con ese valor k para la clave la operación no está definida. Después de la eliminación A continuará siendo un árbol binario de búsqueda

2 NIVEL DE REPRESENTACIÓN O IMPLEMENTACIÓN

Declaraciones básicas

tipos

    tipoNodo = **registro**

        clave: tipoClave

        información : tipoInformación

        izq, der : ↑tipoNodo

fin registro

tipoÁrbol = ↑tipoNodo

punteroNodo = ↑tipoNodo

Búsquedas en árboles binarios de búsqueda

- La definición de un a.b.b implica la existencia de un procedimiento para determinar si un nodo con un valor de clave dado se encuentra en el árbol y para encontrarlo cuando exista
- Procedimiento para encontrar un nodo con valor k para la clave en un a.b.b. con raíz R y clave de la raíz k_R
  - Si el árbol está vacío la búsqueda termina sin éxito
  - Si $k = k_R$ la búsqueda termina satisfactoriamente. El nodo buscado es la raíz del árbol
  - Si $k < k_R$ se sigue la búsqueda en el subárbol izquierdo de la raíz
  - Si $k > k_R$ se sigue la búsqueda en el subárbol derecho de la raíz
- Procedimiento que puede implementarse de forma **recursiva**, donde inicialmente R apunta a la raíz del árbol

Algoritmo de búsqueda en a.b.b.

procedimiento búsqueda(k: tipoClave, **raíz: tipoÁrbol**,
ref n : punteroNodo)

1. **si** raíz = NULO **entonces**
2. n ← NULO
3. **si no, si** (k=raíz↑.clave) **entonces**
4. n ← raíz
5. **si no, si** (k < raíz↑.clave) **entonces**
6. **búsqueda**(k, **raíz↑.izq**, n)
7. **si no [, si** (k > raíz↑.clave) **entonces]**
8. **búsqueda**(k, **raíz↑.der**, n)
9. **fin si**

Comportamiento del algoritmo de búsqueda

- Se analizan el número de comparaciones realizadas antes de terminar la búsqueda
- El comportamiento del algoritmo depende de la profundidad del nodo que contiene la clave buscada en el árbol. Cuanto más lejos se encuentre de la raíz peor será
- ¿Cómo se puede mejorar el comportamiento?
  - Organizando el árbol de forma que las claves buscadas con mayor frecuencia estén situadas tan cerca como sea posible de la raíz. Esto se consigue insertándolas en el árbol en el orden apropiado
    - Problemas:
      - deben conocerse las probabilidades de acceso
      - Se debe tener en cuenta además que, la trayectoria esperada cambiará a medida que se van insertando nuevos nodos en el árbol
      - Se deben tener en cuenta los efectos de una búsqueda sin éxito.
  - En general el número de comparaciones se reduce cuando la altura del árbol es mínima ⇒ **árboles BALANCEADOS**

Inserción en árboles binarios de búsqueda

- Procedimiento para insertar un nodo con valor k para la clave en un a.b.b. con raíz R y clave de la raíz k_R
  - Si el árbol está vacío el nodo con clave k será la raíz
  - Si $k = k_R$ la inserción no puede hacerse (ya existe un nodo con clave k)
  - Si $k < k_R$ se recorre el subárbol izquierdo de la raíz hasta encontrar la posición adecuada para insertar el nuevo nodo
  - Si $k > k_R$ se recorre el subárbol derecho de la raíz hasta encontrar la posición adecuada para insertar el nuevo nodo

- Procedimiento, similar al de búsqueda, que puede implementarse de forma **recursiva**, donde inicialmente R apunta a la raíz del árbol

Algoritmo de inserción en a.b.b.

procedimiento **insertar**(nuevo: punteroNodo, **ref raíz : punteroNodo**)

1. **si** raíz = NULO **entonces**
2.   raíz ← nuevo
3. **si no, si** (nuevo↑.clave = raíz↑.clave) **entonces**
4.   /*Clave duplicada: Implementar según especificación */
5. **si no, si** (nuevo↑.clave < raíz↑.clave) **entonces**
6.   **insertar**(nuevo, **raíz↑.izq**)
7. **si no [, si** (nuevo↑.clave > raíz↑.clave) **entonces]**
8.   **insertar**(nuevo, **raíz↑.der**)
9. **fin sin**

Observaciones

- Nuevo es un puntero al nodo que ha de ser insertado en el a.b.b
- Podría permitirse la inserción de claves duplicadas
- La condición que causa que el procedimiento de búsqueda termine con éxito es la misma que causa que el procedimiento de inserción termine sin éxito
- La inserción ordenada de claves en un a.b.b. produce un árbol largo sin ramificaciones (lista de nodos)
- El orden en que se insertan las claves influye en la altura del árbol y, por tanto, en el comportamiento del algoritmo de búsqueda
- Puede mejorarse este comportamiento reacomodando nodos en el proceso de inserción $\Rightarrow$ árboles BALANCEADOS

Eliminación en árboles binarios de búsqueda

- Eliminación de un nodo del árbol de forma que no viole los principios que lo definen: **el árbol resultante después de la eliminación será un a.b.b**
- Procedimiento para eliminar el nodo con valor k para la clave en un a.b.b con raíz R y clave de la raíz k_R
  - Localizar el nodo que se desea eliminar siguiendo el mismo método que en el procedimiento de búsqueda
  - Distinguir los siguiente casos
    - Si el nodo a eliminar es hoja o terminal, simplemente se suprime
    - Si el nodo a eliminar tiene un solo descendiente, se sustituye por ese descendiente y se suprime
    - Si el nodo a eliminar tiene los dos descendientes, entonces se sustituye por el **nodo más a la izquierda del subárbol derecho** o por el **nodo más a la derecha del subárbol izquierdo**, eliminándose el nodo sustituido

Algoritmo de eliminación en a.b.b.

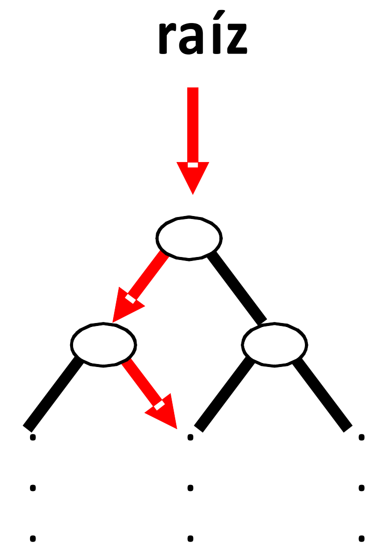
procedimiento eliminar(x: tipoClave, **ref raíz : tipoÁrbol**)

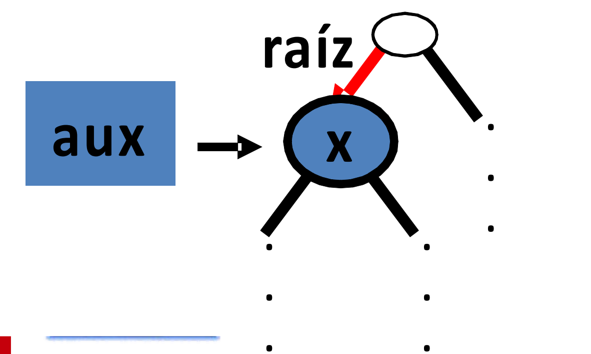
1. aux, ant : punteroNodo
2. **si** raíz = NULO entonces
3. /*no existe nodo con clave x: implementar según especificación*/
4. **si no, si** x < raíz↑.clave **entonces**
5. **eliminar**(x, **raíz↑.izq**)
6. **si no, si** x > raíz↑.clave **entonces**
7. **eliminar**(x, **raíz↑.der**)
8. **si no [, si** (x = raíz↑.clave) **entonces]**
9. aux ← raíz

……………

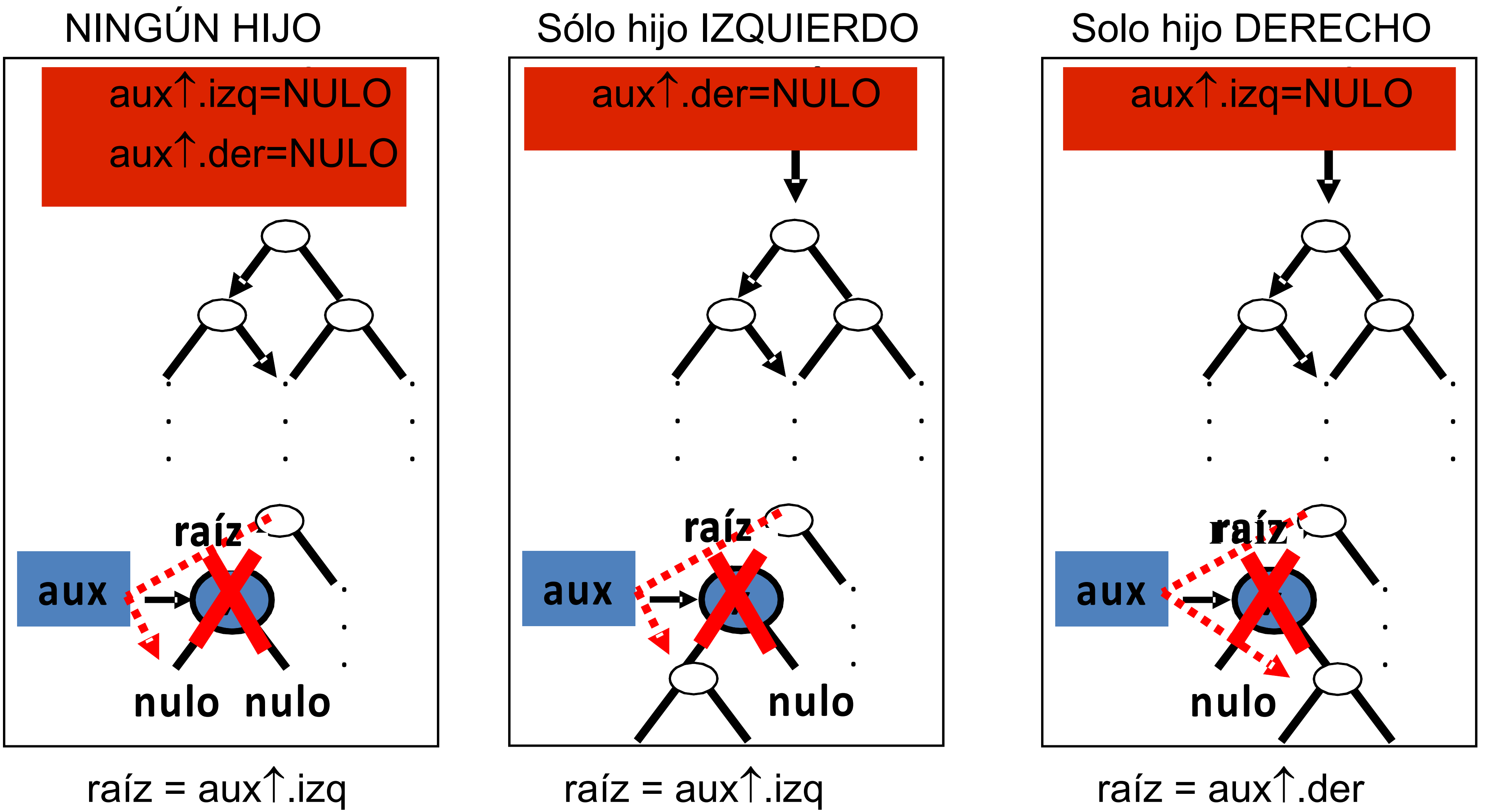
28. eliminar_nodo(aux)
29. **fin si**

Distinguir casos y eliminar nodo





Distinción de casos: un descendiente o ninguno



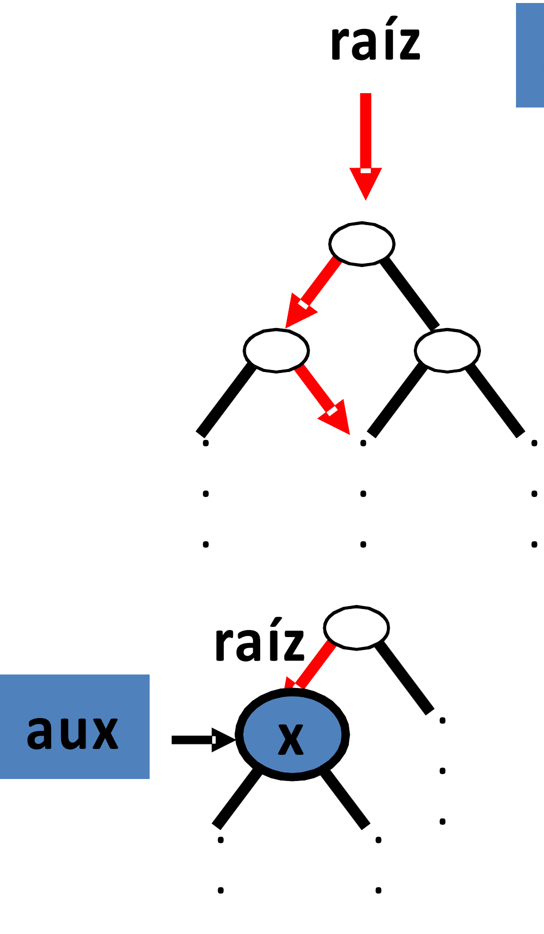
Distinción de casos

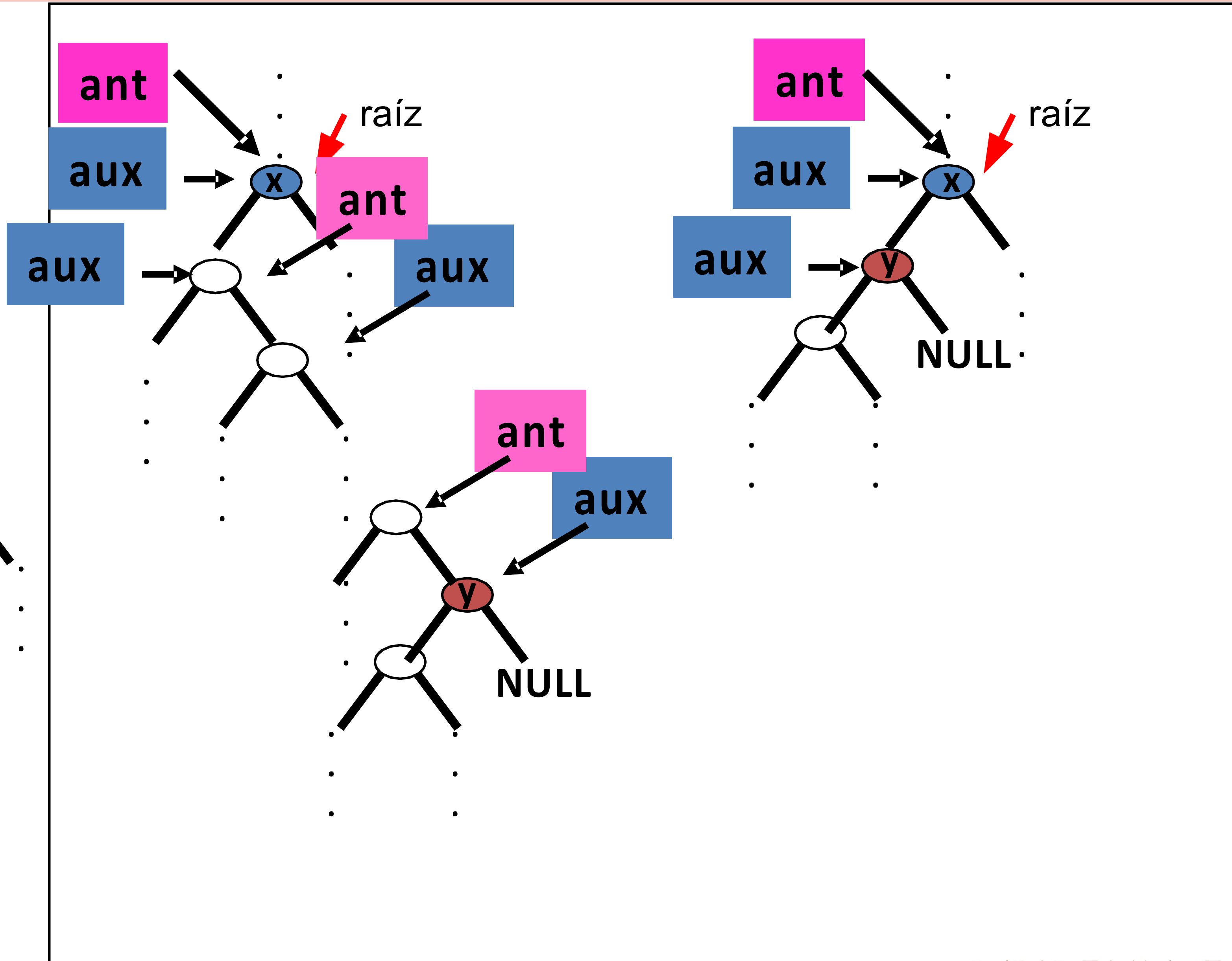
```
aux ← raíz
si aux↑.der = NULO entonces /* sólo hijo izquierdo o ningún descendiente*/
    raíz ← aux ↑.izq
si no, si aux↑.izq = NULO entonces /* sólo hijo derecho*/
    raíz ← aux ↑.der
si no
…..........................
 fin si
eliminar_nodo(aux)
fin si
```

DOS HIJOS

Dos descendientes:
buscar nodo mas derecha de subárbol izquierdo de x

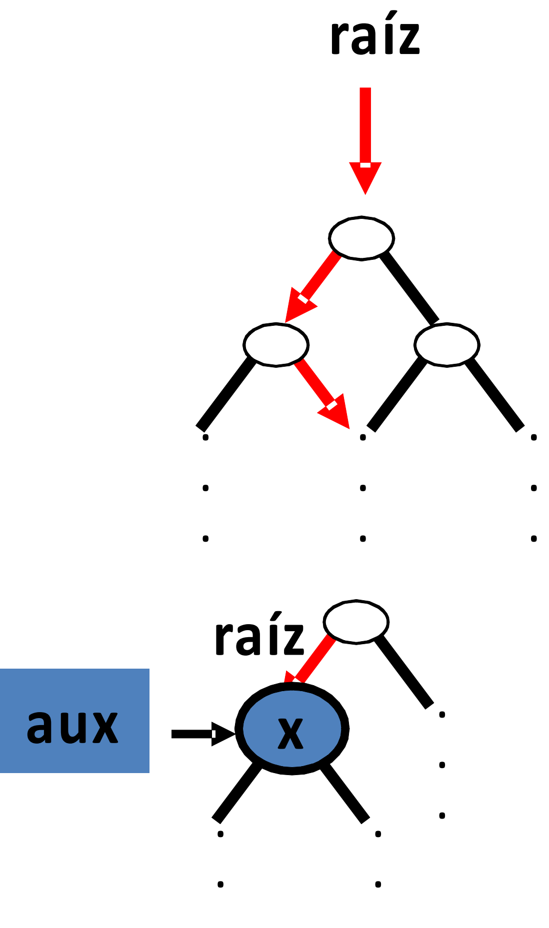
aux↑.izq≠NULO
aux↑.der≠NULO

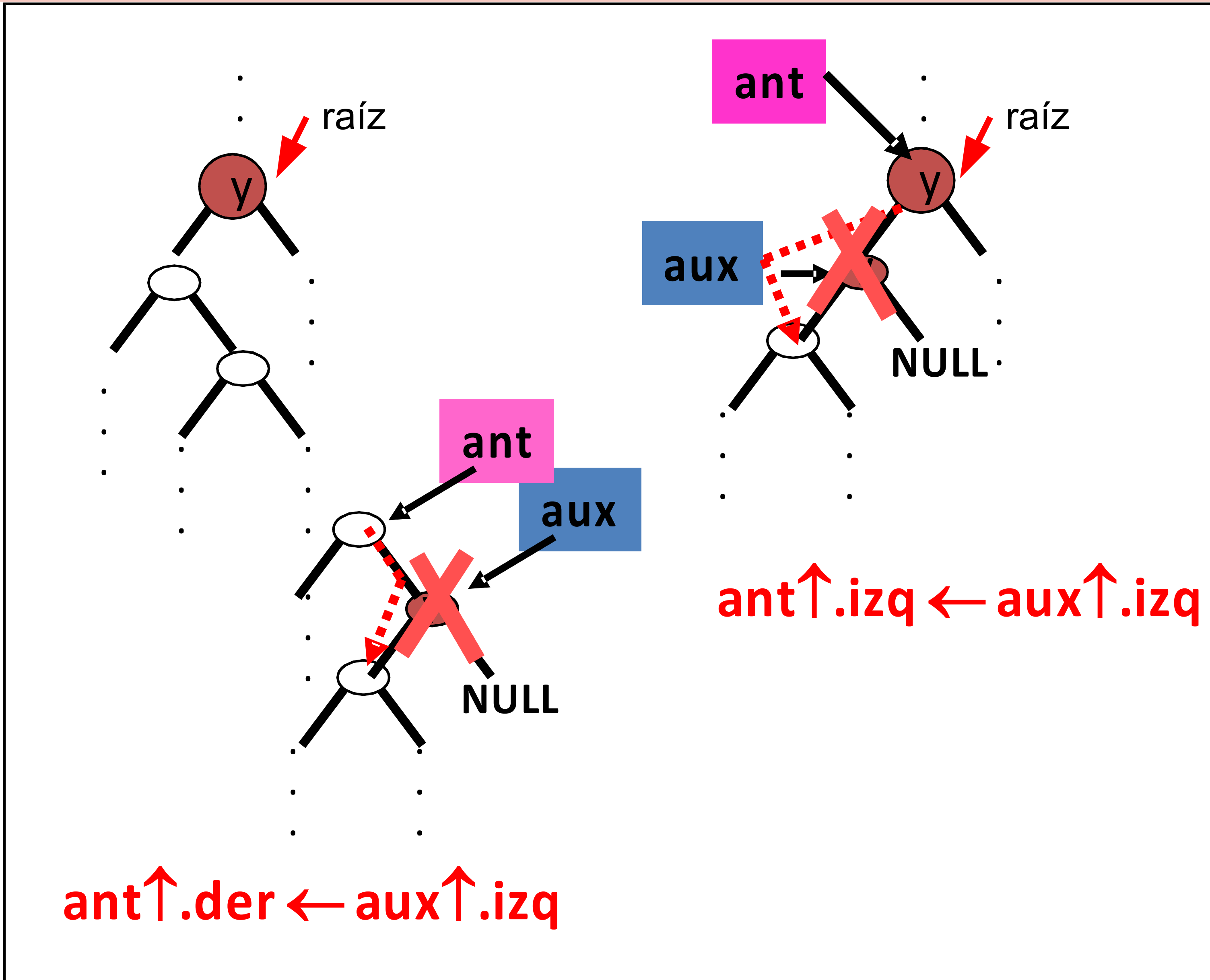




Dos descendientes:
sustituir x por nodo mas derecha de subárbol izquierdo de x

aux↑.izq≠NULO
aux↑.der≠NULO





Distinción de casos

```
…
si no                                   /* dos hijos*/
    ant ← aux
    aux ←  aux↑.izq
    mientras aux↑.der ≠ NULO hacer
        ant  ← aux
        aux ← aux↑.der
    fin mientras
    raíz↑.clave ← aux↑.clave
    raíz↑.información ← aux↑.información
    si ant = raíz entonces
        ant↑.izq ← aux↑.izq
    si no
        ant↑.der ← aux↑.izq
   fin si
eliminar_nodo(aux)
fin si
```

Bucar nodo más derecha subárbol izquierdo

Sustituir información

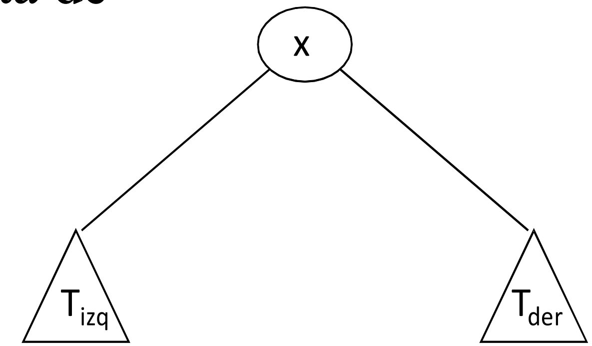
Enlazar subárboles de aux ante de eliminar aux

Análisis del caso promedio

- Puesto que el tiempo en descender un nivel en el árbol es constante, el tiempo de ejecución de las tres operaciones vistas será O(d), siendo d la profundidad del nodo que contiene la clave de búsqueda
- Para cada nodo del árbol, el numero de comparaciones viene dado por su profundidad o distancia desde la raíz a ese nodo
- La suma de estas distancias para todos los nodos se denomina longitud de camino interno del árbol
- Dividiendo la longitud de camino interno por el número de nodos se obtendrá el número medio de comparaciones para una búsqueda con éxito
- Si todos los árboles son igualmente probables, la profundidad media en todos los nodos de un árbol es O(log n) y por tanto también lo será el tiempo de ejecución de una operación de búsqueda

Longitud media de camino interno

- Un a.b.b. aleatorio de n nodos, para $0 \le i < n$, consta de
  - Una raíz
  - Un subárbol izquierdo de i nodos.
  - Un subárbol derecho de $n-i-1$ nodos



- Si $D(n)$ es la longitud de camino interno de un árbol de n nodos

$$D(n) = D(i) + D(n-i-1) + (n-1)$$

- El término $n-1$ tiene en cuenta el hecho de que la raíz contribuye con 1 a la longitud del camino para cada uno de los $n-1$ nodos restantes del árbol
- Si todos los tamaños de subárboles son igualmente probables $\Rightarrow$ el valor promedio de $D(i)$ y $D(n-i-1)$ es

$$\frac{1}{n}\sum_{j=0}^{n-1} D(j)$$

Profundidad esperada de un nodo cualquiera

- Se obtiene, por tanto, la longitud media de camino interno de un árbol de *n* nodos

$$D(n) = \frac{2}{n}\sum_{j=0}^{n-1} D(j) + n - 1$$

- Misma recurrencia que aparece en el análisis del algoritmo de ordenación rápida (quicksort), donde se obtuvo que

$$D(n) \in O(n \log n)$$

- Por tanto la profundidad esperada de cualquier nodo es O(log*n*)
- ¿Son todos los árboles igualmente probables?
  - La inserción ordenada de claves produce una lista de nodos
  - El algoritmo de eliminación favorece la creación de subárboles izquierdos
- Solución: añadir una condición estructural que evite profundidades excesivas en los nodos

3 ÁRBOLES BALANCEADOS

- Los árboles balanceados o árboles AVL (Adelson-Velskii, Laudis) tratan de mejorar el comportamiento del algoritmo de búsqueda en a.b.b. realizando "reacomodos" de nodos después de las inserciones y eliminaciones
- Evitan que el árbol pueda "crecer" o "decrecer" descontroladamente
- **Definición formal:** un árbol balanceado es un a.b.b. en el cual para todo nodo n_i se cumple la siguiente condición:

  "La altura del subárbol izquierdo de n_i y la altura del subárbol derecho de n_i difieren como máximo en una unidad"

- Para determinar si un árbol está o no balanceado se necesita información relativa al equilibrio de cada nodo del árbol $\Rightarrow$ factor de equilibrio.

Factor de equilibrio

- Factor de equilibrio de un nodo(FE): diferencia entre la altura del subárbol derecho y la altura del subárbol izquierdo

$$\boxed{FE_n = H_{RD} - H_{RI}}$$

- Valores posibles para factor de equilibrio en un árbol balanceado: **-1, 0, 1**
- Para evitar que el factor de equilibrio llegue a tomar valores -2 ó 2 el árbol deberá reestructurarse.

Declaraciones básicas y ejemplo

tipo

tipoNodo = **registro**

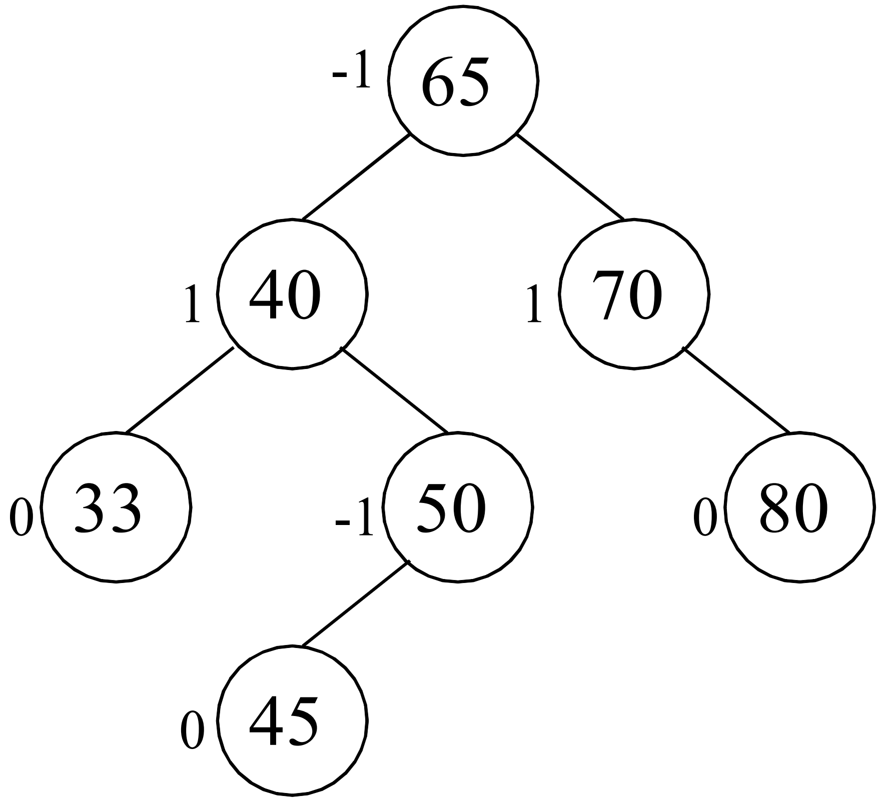
clave : tipoClave

información : tipoInformación

fe : -1 .. 1

izq, der : ↑punteroNodo

fin registro



Inserción y equilibrio del árbol

- Siguiendo el algoritmo de búsqueda en a.b.b, se insertará el nodo en el subárbol izquierdo o derecho, según corresponda
- Casos a tener en cuenta:
  - El nuevo nodo se inserta sin modificar la altura del subárbol en que se inserta $\Rightarrow$ ni la altura de la raíz ni el equilibrio del árbol se modifican
  - El nuevo nodo se inserta aumentando la altura del subárbol más corto

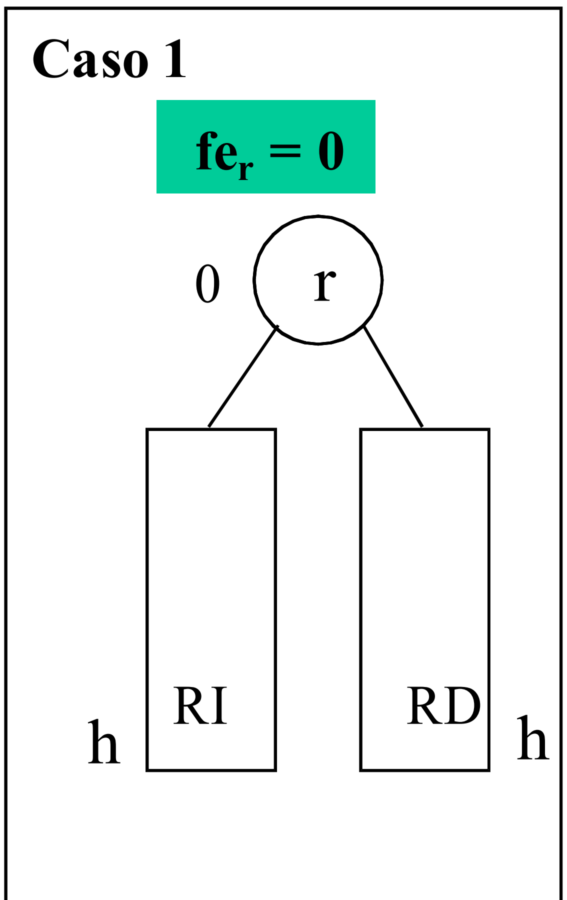
    $\Rightarrow$ tampoco se perderá el equilibrio del árbol
  - El nuevo nodo se inserta aumentando la altura del subárbol más largo

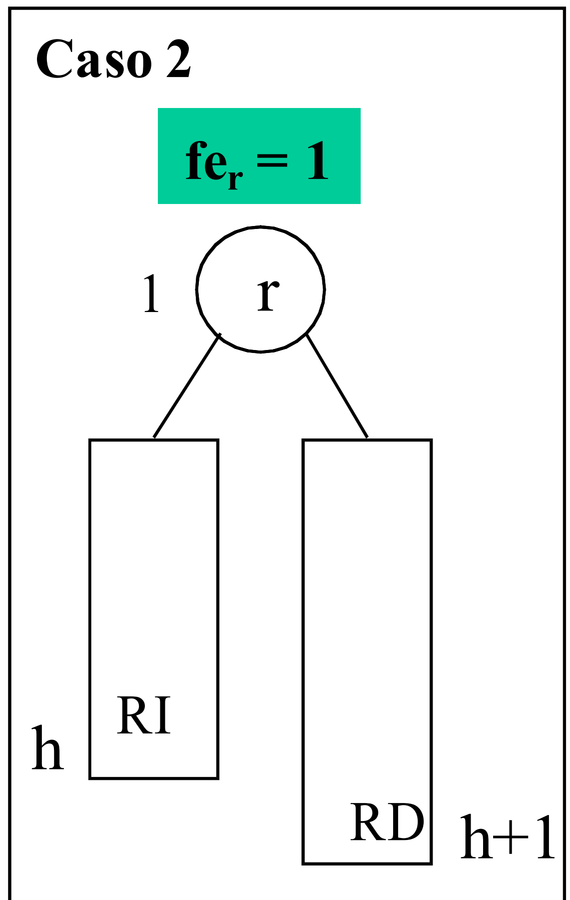
    $\Rightarrow$ el árbol perderá el equilibrio

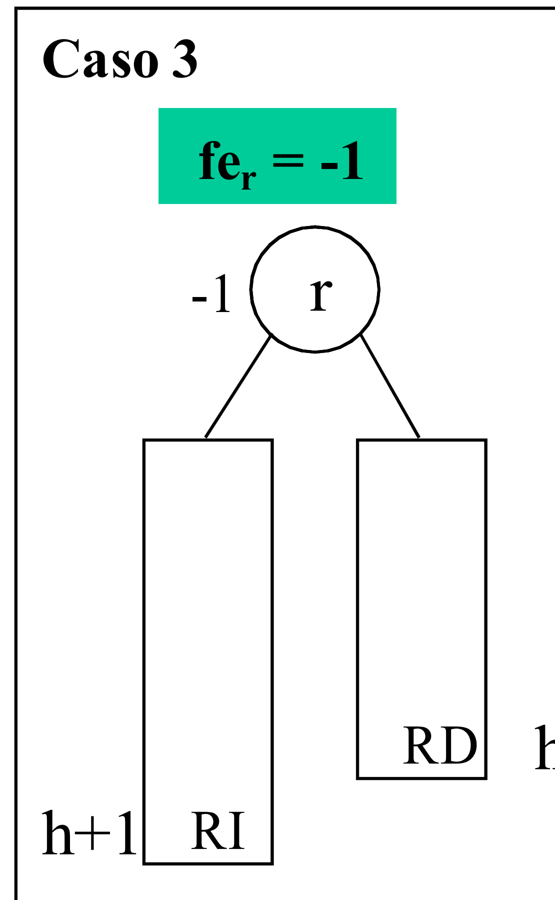
- Para distinguir estos casos, partiremos de las diferentes situaciones antes de la inserción y estudiaremos todas las posibilidades que pueden presentarse ante una inserción de un nuevo nodo

Distinción de casos antes de la inserción

- Las ramas izquierda y derecha tienen la misma altura
- La altura de la rama izquierda es menor que la altura de la rama derecha
- La altura de la rama izquierda es mayor que la altura de la rama derecha

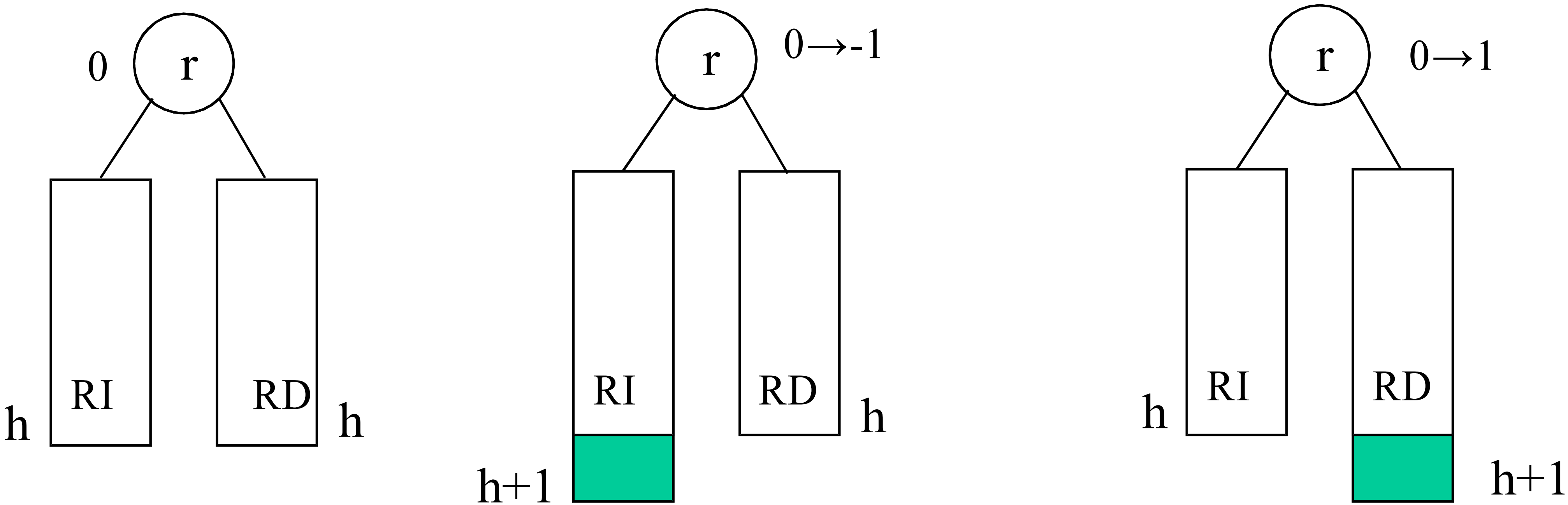






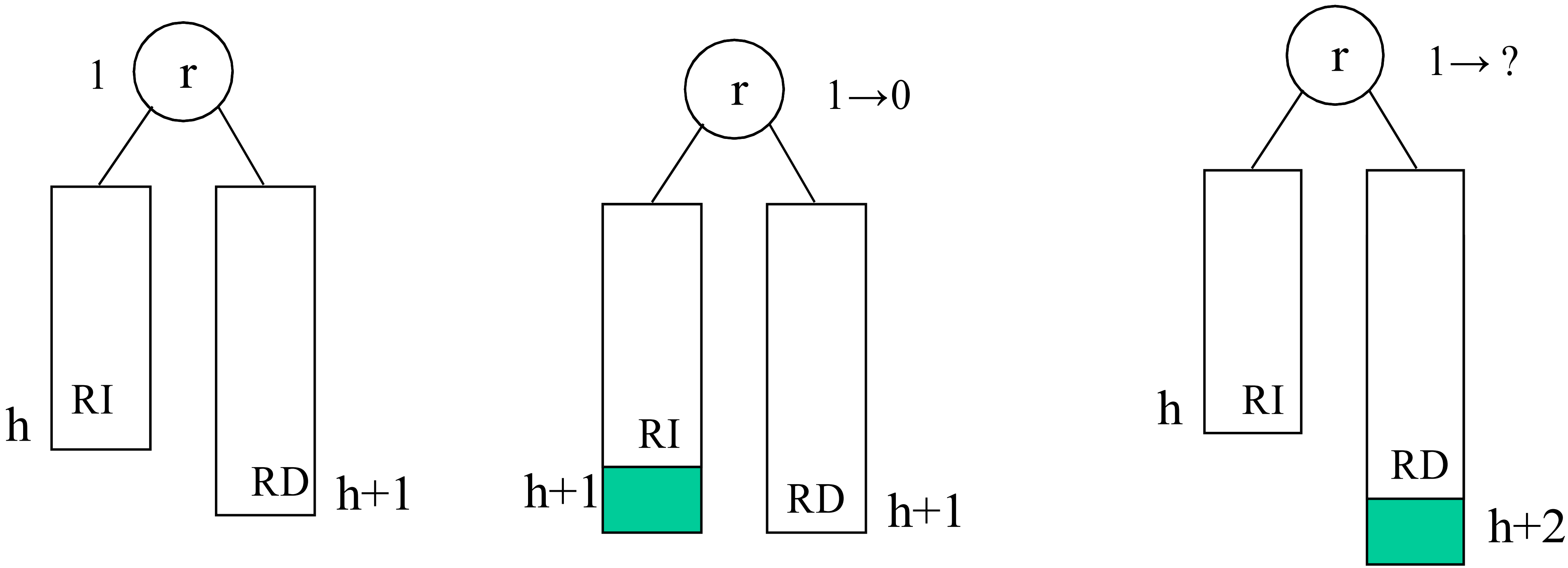
Inserción de un nodo en el caso 1 (FE = 0)

- Tanto si se inserta por la izquierda como por la derecha, el árbol sigue balanceado, pero su altura aumenta y cambia el factor de equilibrio



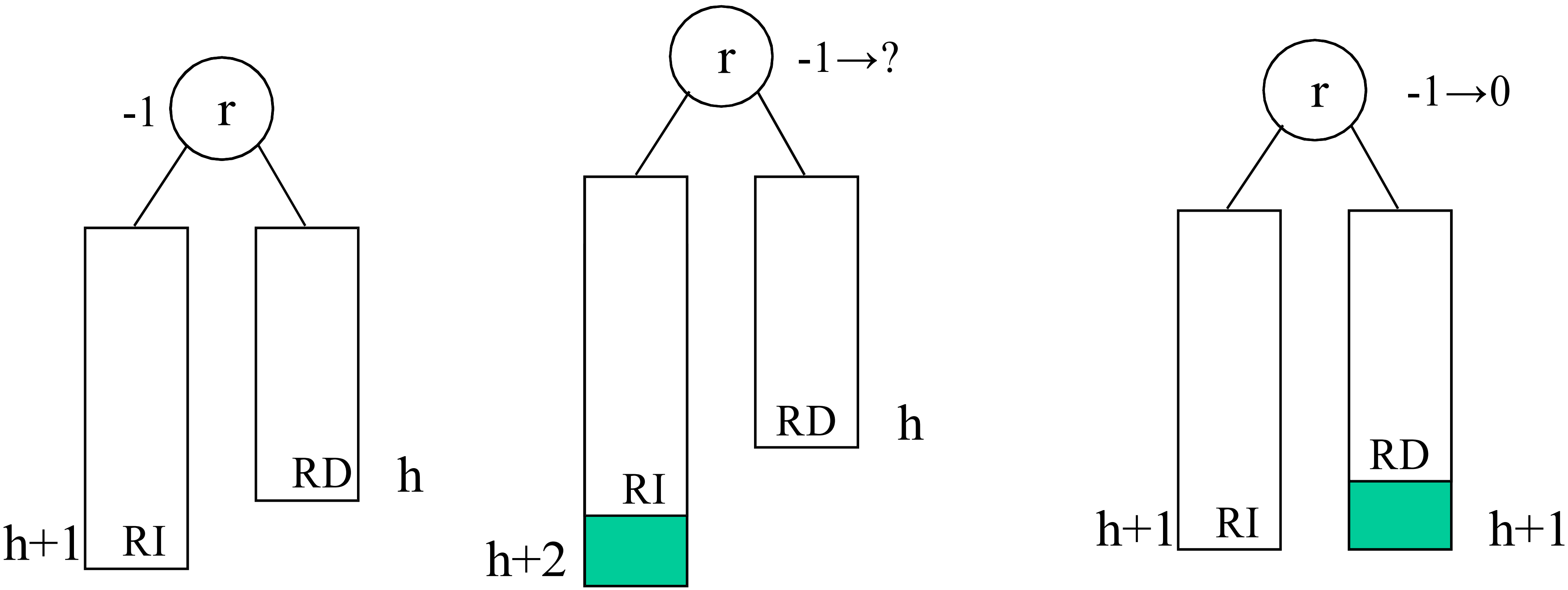
ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	1	h+2	verdadero

Inserción de un nodo en el caso 2 (FE = 1)



ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	**h+2**	**0**	**h+2**	**falso**	**? Reestruct. SUB.DERECHO**		

Inserción de un nodo en el caso 3 (FE = -1)



ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	?	?	?
-1	**h+2**	**? Reestruct. SUB.IZQUIERDO**			**0**	**h+2**	**falso**

Procedimiento de INSERCIÓN

- Localizar la posición del árbol donde debe insertarse el nuevo nodo utilizando el mismo método que en la inserción del a.b.b, insertar el nodo y calcular su factor de equilibrio (lógicamente será cero)
- **Regresar** por el camino de búsqueda **recalculando** el factor de equilibrio de todos los nodos siempre que la altura de alguno de sus subárboles haya cambiado. Reestructurar el árbol en aquellos casos en que sea necesario (antes de que el factor de equilibrio tome valores 2 ó -2)

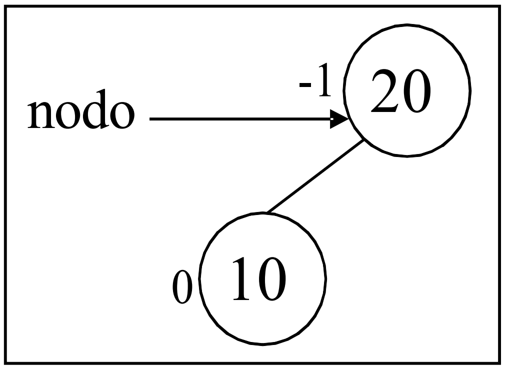
- Puede implementarse como un algoritmo **recursivo** que tiene como parámetros:
  - puntero al nuevo nodo
  - puntero que inicialmente señala a la raíz del árbol, que permitirá seguir el camino de búsqueda
  - un parámetro de tipo lógico que indicará si la altura del subárbol ha cambiado (aumentado) como consecuencia de la inserción

Algoritmo de inserción

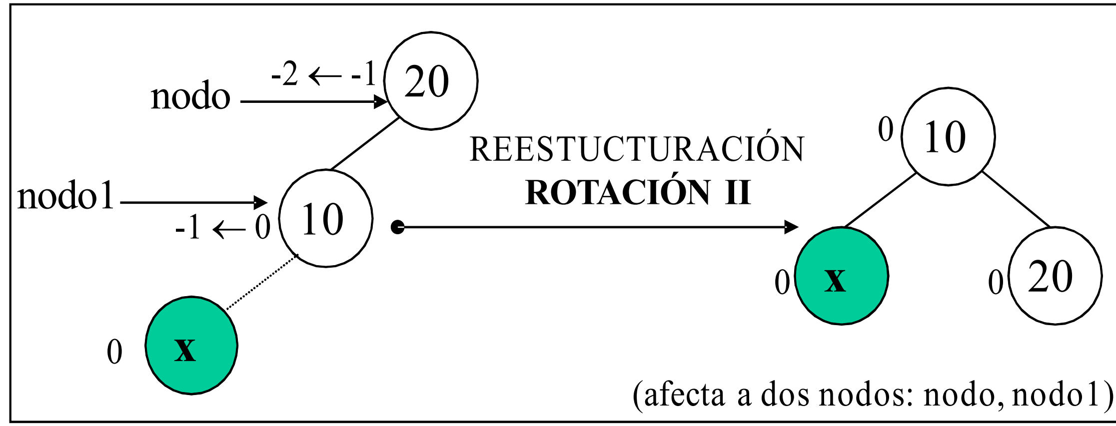
```
procedimiento insertar( nuevo: punteroNodo, ref cambiaH:lógico,
                ref nodo : punteroNodo)
    aux, nodo1, nodo2: punteroNodo
    si nodo = NULO entonces
        nodo ← nuevo
        cambiaH ← VERDADERO
    sino, si (nuevo↑.clave < nodo↑.clave) entonces
        insertar(nuevo, cambiaH, nodo↑.izq)
        si cambiaH entonces
            equilibrarIZQ(nodo,cambiaH)
        fin si
    sino, si (nuevo↑.clave > nodo↑.clave) entonces
        insertar(nuevo, cambiaH, nodo↑.der)
        si cambiaH entonces
            equilibrarDER(nodo,cambiaH)
        fin si
    sino
        /* clave duplicada: implementar según especificación*/
    fin si
```

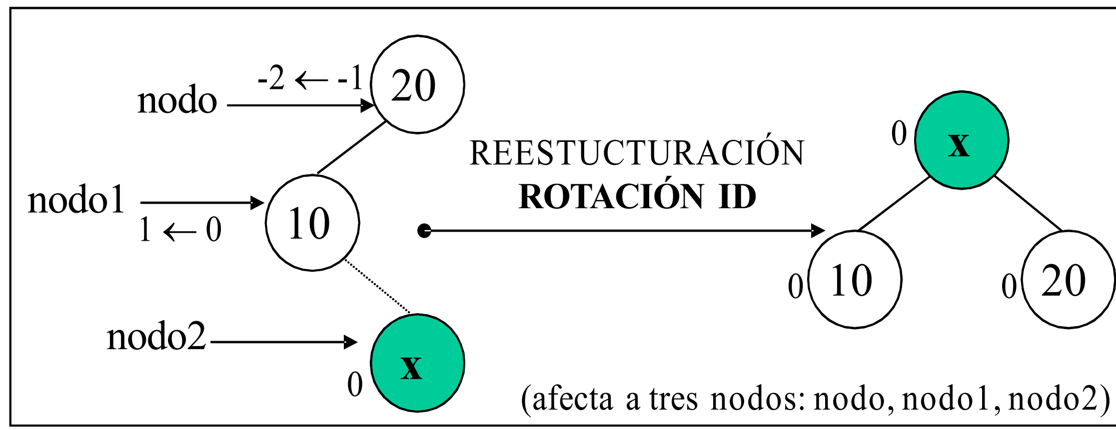
Ejemplo reestructuración subárbol IZQUIERDO (nodo↑.FE = −1)

Estado antes de la inserción:



Estado después de la inserción:





ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	?	?	?
-1	h+2	¿II o ID? Depende de FE nodo↑.izq			0	h+2	falso

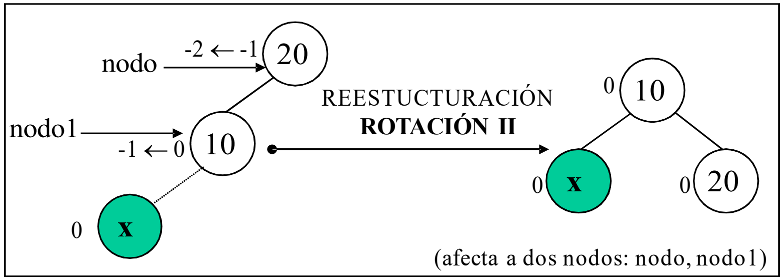
Equilibrar rama izquierda

```
procedimiento equilibrarIZQ( ref nodo: punteroNodo, ref cambiaH: lógico)
caso nodo↑.fe en
     0:  nodo↑. fe ← -1
     1:  nodo↑. fe ← 0
         cambiaH ← FALSO
    -1:  nodo1← nodo↑.izq
         si nodo1↑.fe= -1 entonces
             rotaciónII(nodo, nodo1)
         sino
             nodo2 ← nodo1↑.der
             rotaciónID(nodo, nodo1, nodo2)
         fin si
         camibaH ← FALSO
fin caso
```

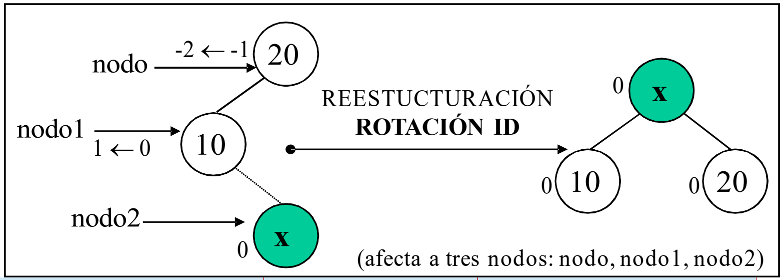
Reestructuración Subárbol IZQUIERDO

ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero
1	h+2	0	h+2	falso
-1	h+2	¿II o ID? Depende de FE nodo↑.izq		

Estado después de la inserción:



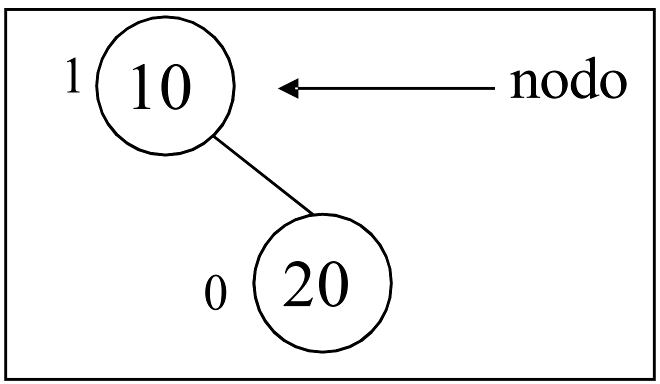
(afecta a dos nodos: nodo, nodo1)



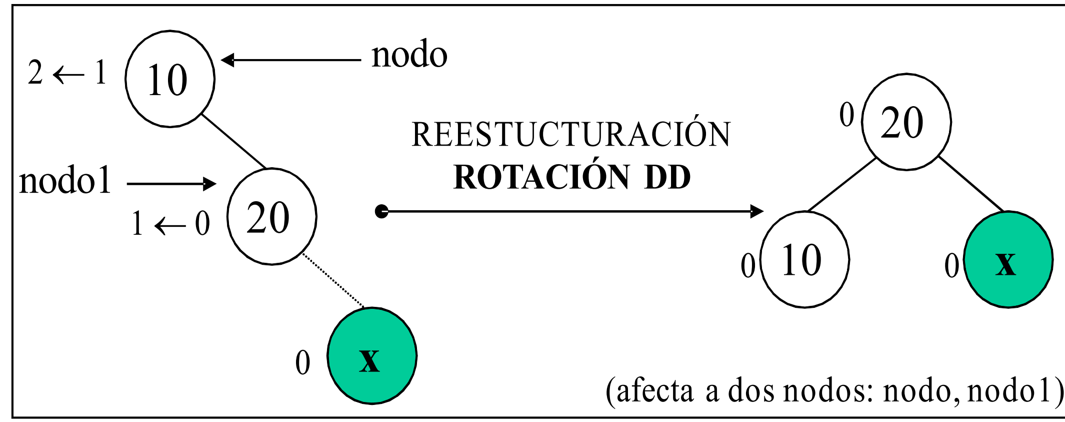
(afecta a tres nodos: nodo, nodo1, nodo2)

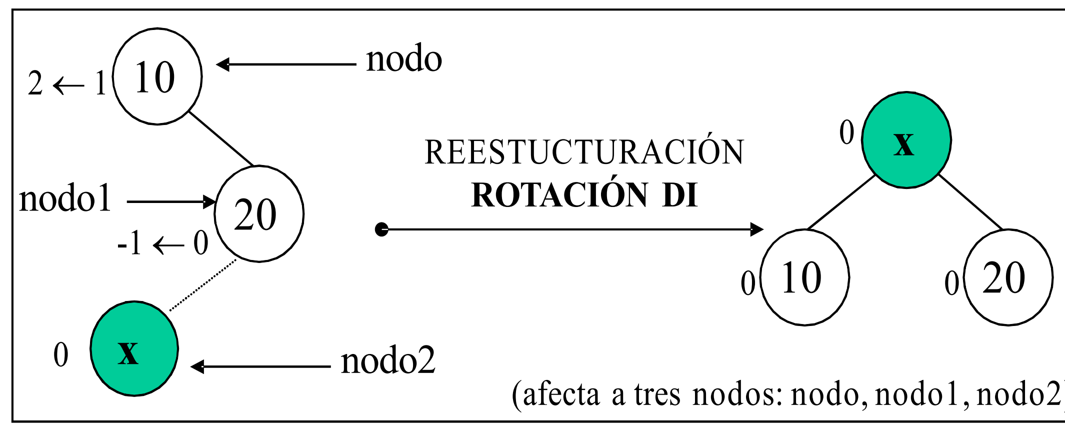
Ejemplo reestructuración subárbol DERECHO (nodo↑.FE = 1)

Estado antes de la inserción:



Estado después de la inserción:





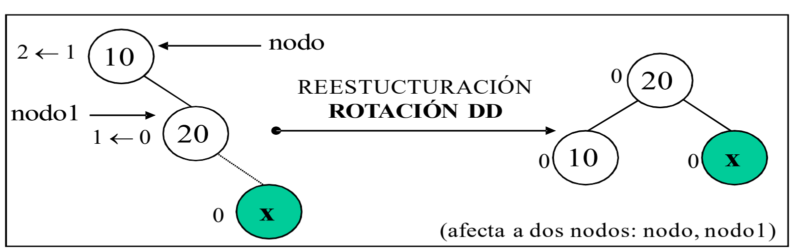
<table>
<tr><th colspan="2">ANTES INSERCIÓN</th><th colspan="3">DESPUÉS INSERCIÓN POR RAMA IZQUIERDA</th><th colspan="3">DESPUÉS INSERCIÓN POR RAMA DERECHA</th></tr>
<tr><th>nodo↑.fe</th><th>altura</th><th>nodo↑.fe</th><th>altura</th><th>cambiaH</th><th>nodo↑.fe</th><th>altura</th><th>cambiaH</th></tr>
<tr><td>0</td><td>h+1</td><td>-1</td><td>h+2</td><td>verdadero</td><td>+1</td><td>h+2</td><td>verdadero</td></tr>
<tr><td>1</td><td>h+2</td><td>0</td><td>h+2</td><td>falso</td><td colspan="3">¿DD o DI?Depende de FE nodo↑.der</td></tr>
<tr><td>-1</td><td>h+2</td><td colspan="3">¿II o ID? Depende de FE nodo↑.izq</td><td>0</td><td>h+2</td><td>falso</td></tr>
</table>

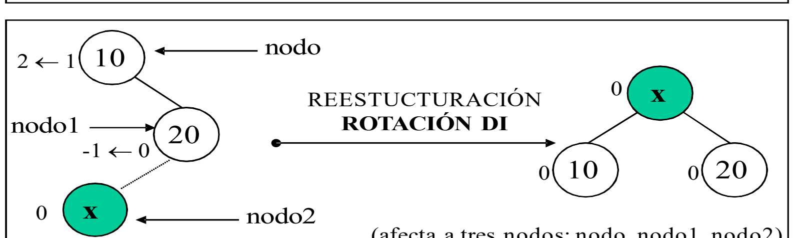
Equilibrar rama derecha

```
procedimiento equilibrarDER( ref nodo: punteroNodo, ref cambiaH: lógico)
caso nodo↑.fe en
      0: nodo↑. fe ← 1
      1: nodo1← nodo↑.der
         si nodo1↑.fe= 1 entonces
               rotaciónDD(nodo, nodo1)
         sino
               nodo2 ← nodo1↑.izq
               rotaciónDI(nodo, nodo1, nodo2)
          fin si
          camibaH ← FALSO
     -1: nodo↑. fe ← 0
         cambiaH  FALSO
fin caso
```

Reestructuración Subárbol DERECHO

Estado después de la inserción:





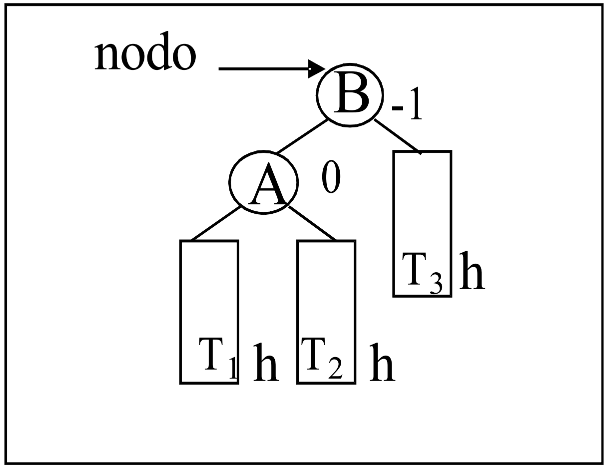
ANTES INSERCIÓN

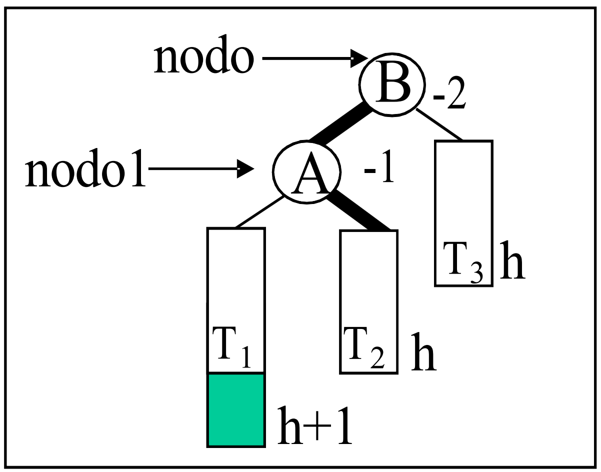
nodo↑.fe	altura
0	h+1
1	h+2
-1	h+2

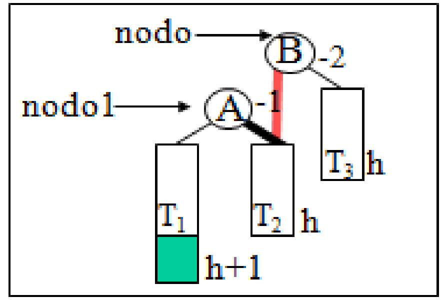
DESPUÉS INSERCIÓN POR RAMA DERECHA

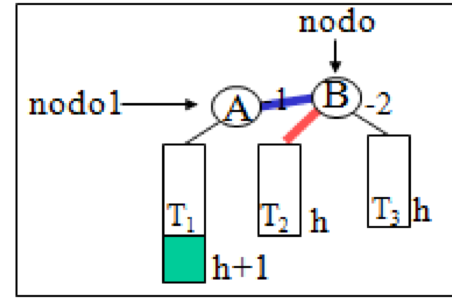
nodo↑.fe	altura	cambiaH
+1	h+2	verdadero
¿DD o DI?Depende de FE nodo↑.der		
0	h+2	falso

Rotación IZQUIERDA IZQUIERDA
Movimiento de los nodos en la rotación y cambio factor equilibrio









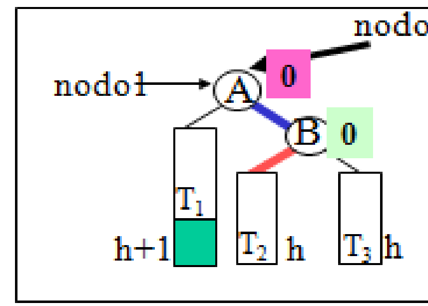
nodo↑.izq ←nodo1↑.der

nodo1↑.der ←nodo

nodo↑.fe ← 0

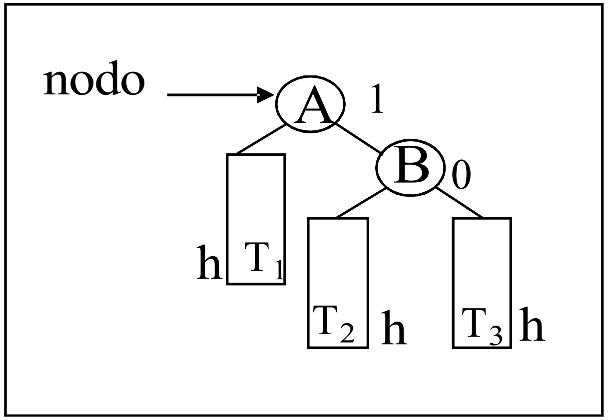
nodo1↑.fe ← 0

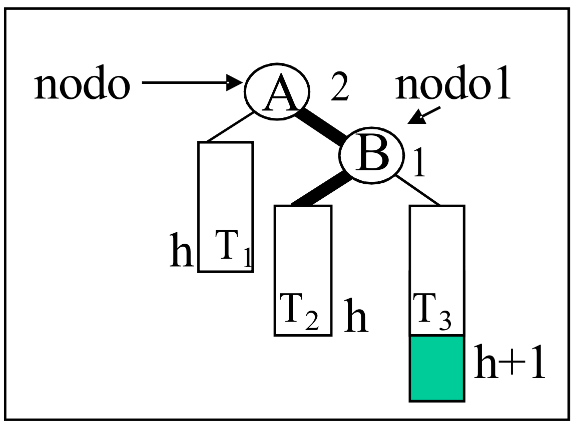
nodo ← nodo1

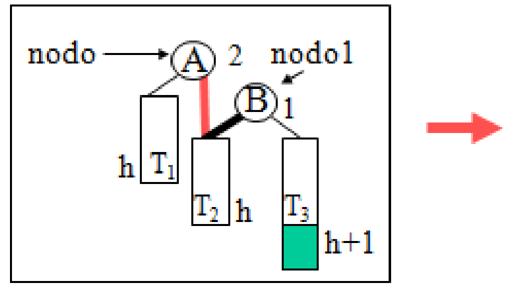


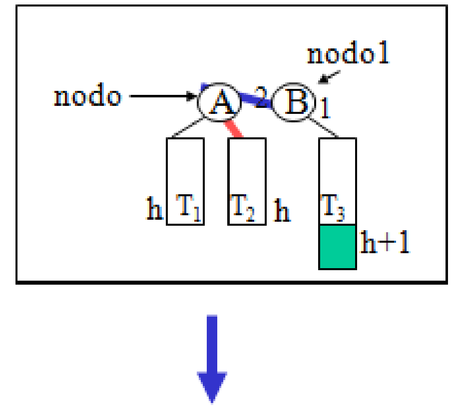
ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA (nodo1=nodo↑.izq)			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	+1	h+2	verdadero
1	h+2	0	h+2	falso	¿DD o DI?Depende FE de nodo↑.der		
-1	h+2	nodo1↑.fe = -1 ⇒ rotación II			0	h+2	falso

Rotación DERECHA DERECHA
Movimiento de los nodos en la rotación y cambio factor equilibrio









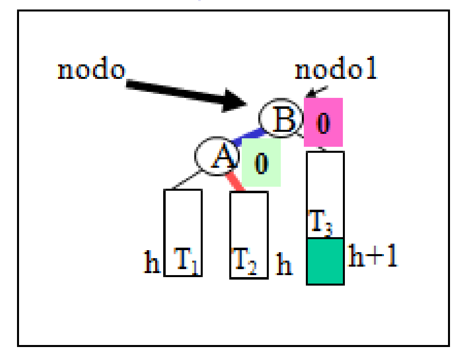
nodo↑.der ← nodo1↑.izq

nodo1↑.izq ← nodo

nodo↑.fe ← 0

nodo1↑.fe ←0

nodo ← nodo1



ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA (nodo1=nodo↑.der)		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	+1	h+2	verdadero
1	h+2	0	h+2	falso	nodo1↑.fe = 1 ⇒ rotación DD		
-1	h+2	¿II o ID? Depende de FE nodo↑.izq			0	h+2	falso

Algoritmos de rotaciones simples: II y DD
Sólo sirven para proceso de inserción

procedimiento rotaciónII(**ref** nodo: punteroNodo, nodo1: punteroNodo)

1. nodo↑.izq ← nodo1↑.der
2. nodo1↑.der ← nodo
3. nodo↑.fe← 0
4. nodo1↑.fe← 0
5. nodo ← nodo1

Movimiento nodos en rotación (pasos 1–2)

Cambio en los factores de equilibrio (pasos 3–4)

Nueva raíz del subárbol (paso 5)

procedimiento rotaciónDD(**ref** nodo: punteroNodo, nodo1: punteroNodo)

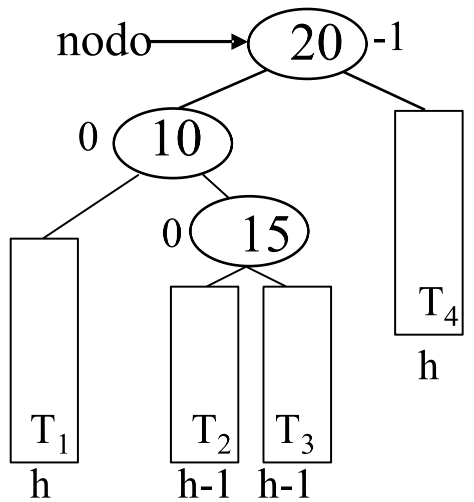
1. nodo↑.der ← nodo1↑.izq
2. nodo1↑.izq ← nodo
3. nodo↑.fe ← 0
4. nodo1↑.fe ← 0
5. nodo ← nodo1

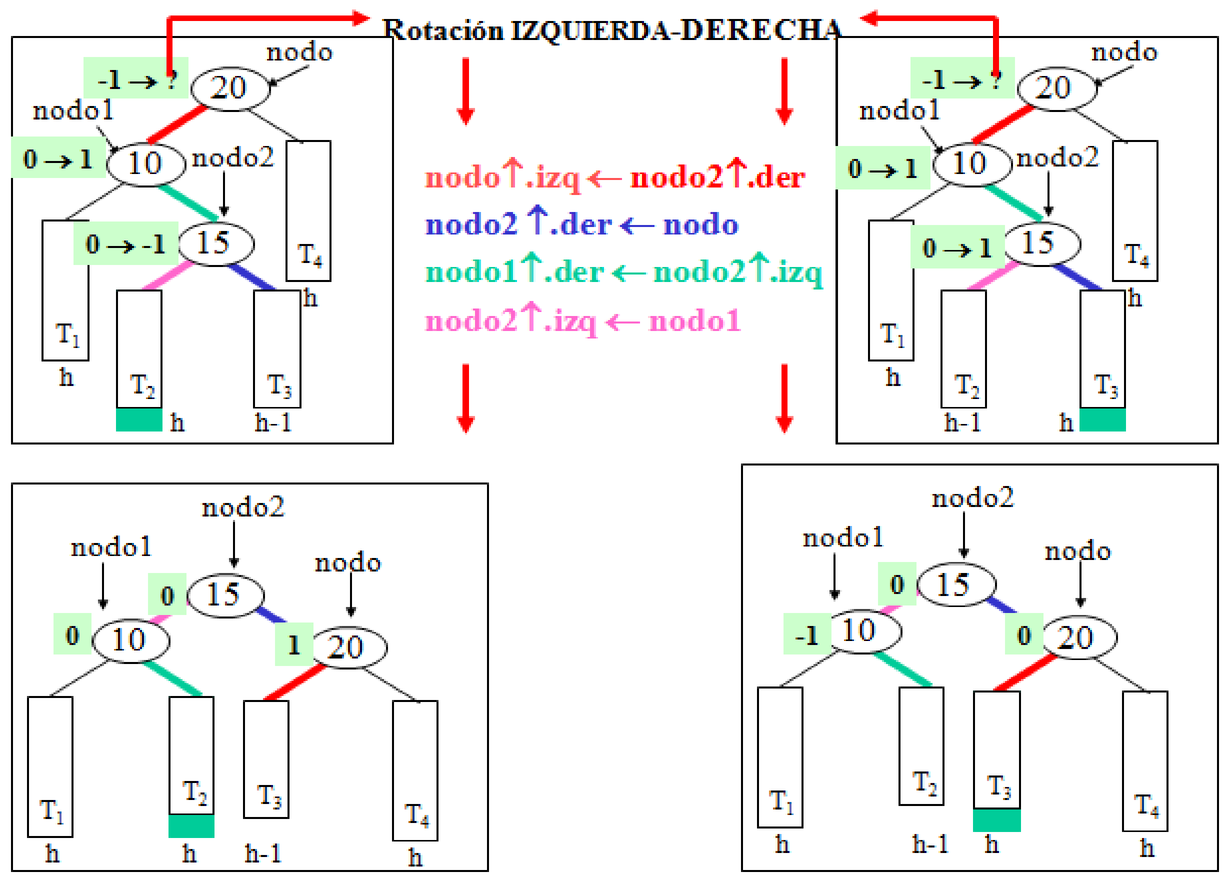
Movimiento nodos en rotación (pasos 1–2)

Cambio en los factores de equilibrio (pasos 3–4)

Nueva raíz del subárbol (paso 5)

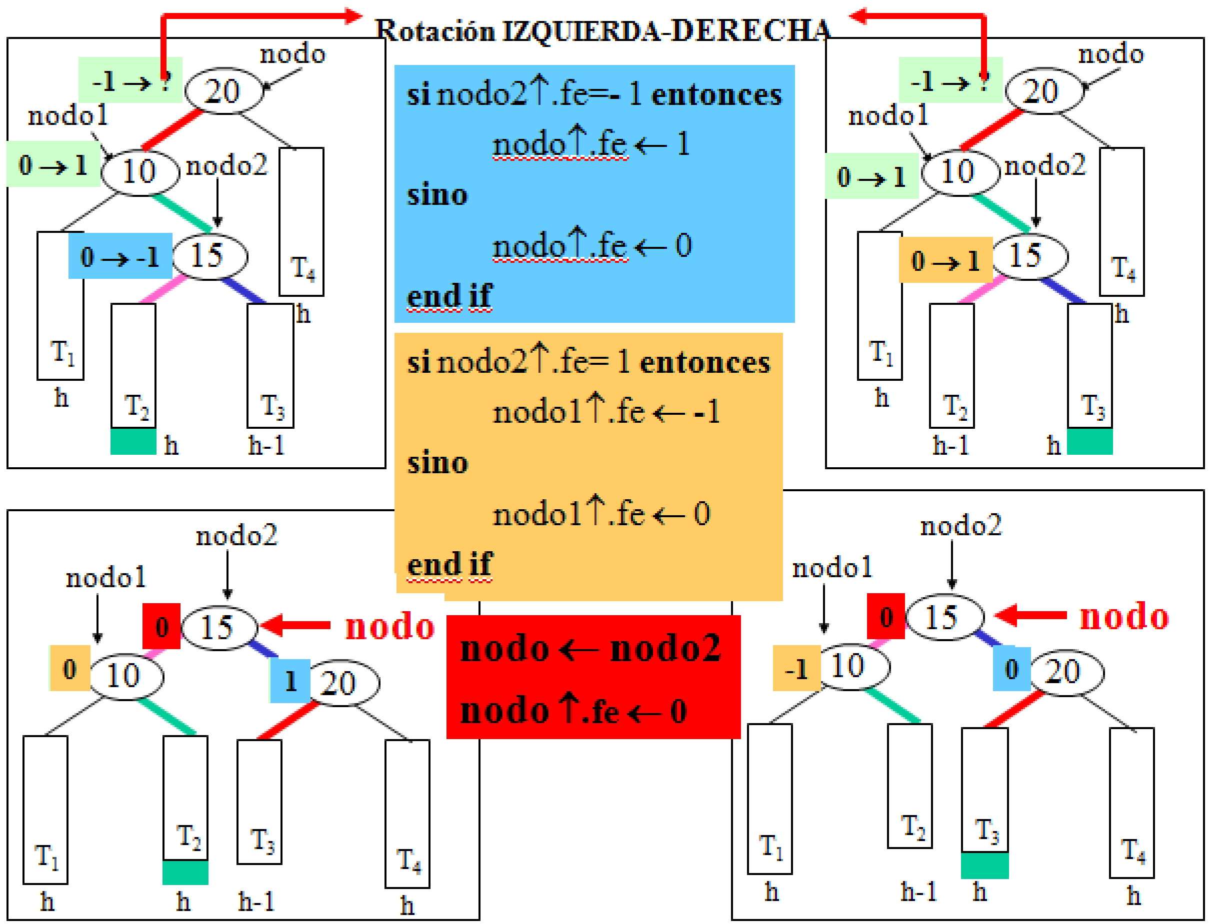
Rotación IZQUIERDA DERECHA
1. Movimiento de los nodos en la rotación





ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA (nodo1=nodo↑.izq)			DESPUÉS INSERCIÓN POR RAMA DERECHA		
nodo↑.fe	altura	nodo↑.fe	altura	cambiaH	nodo↑.fe	altura	cambiaH
0	h+1	-1	h+2	verdadero	+1	h+2	verdadero
1	h+2	0	h+2	falso	¿DD o DI?Depende FE de nodo↑.der		
-1	h+2	nodo1↑.fe = 1 ⇒ rotación ID			0	h+2	falso

Rotación IZQUIERDA DERECHA
2. Cambio de factores de equilibrio



Algoritmo de rotación IZQUIERDA DERECHA

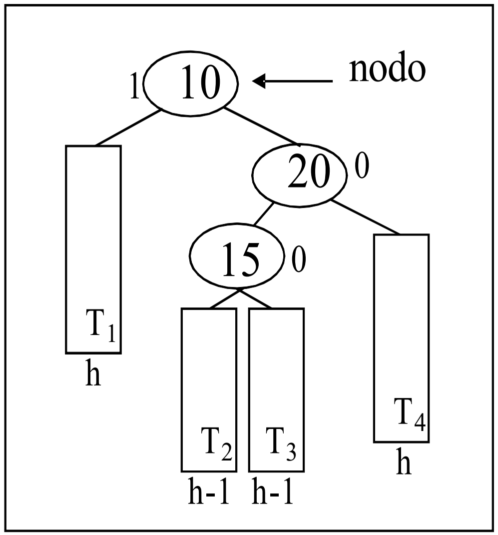
```
procedimiento rotaciónID(ref nodo: punteroNodo,
      nodo1, nodo2: punteroNodo)
    nodo↑.izq ← nodo2↑.der
    nodo2↑der ← nodo
    nodo1↑.der ← nodo2↑.izq
    nodo2↑.izq ← nodo1
    si nodo2↑.fe= -1 entonces
       nodo↑.fe ← 1
    sino
       nodo↑.fe ← 0
    fin si
    si nodo2↑.fe=1 entonces
       nodo1↑.fe ← -1
    sino
       nodo1↑.fe ← 0
    fin si
    nodo ← nodo2
    nodo↑.fe ← 0
```

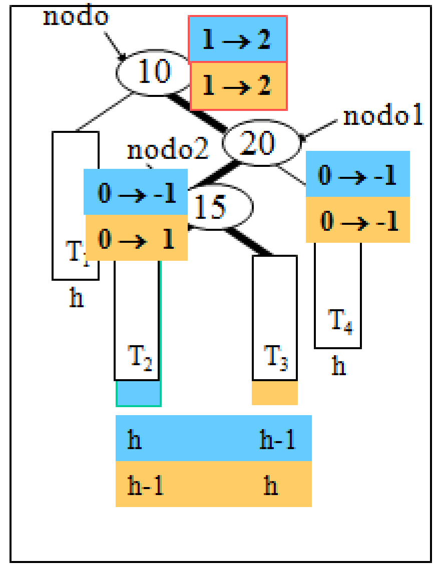
Movimiento nodos en rotación

Cambio en los factores de equilibrio

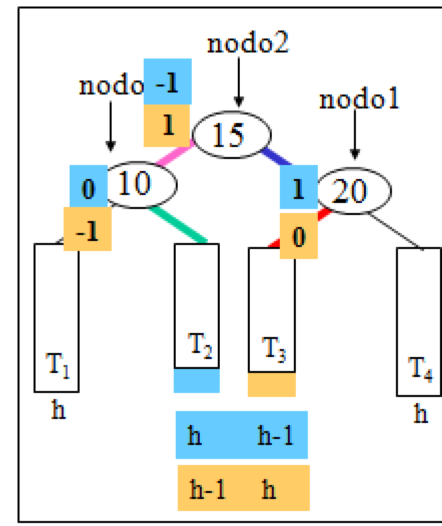
Asignación de la nueva raíz del subárbol y de su factor de equilibrio

Rotación DERECHA IZQUIERDA
1. Movimiento de los nodos en la rotación



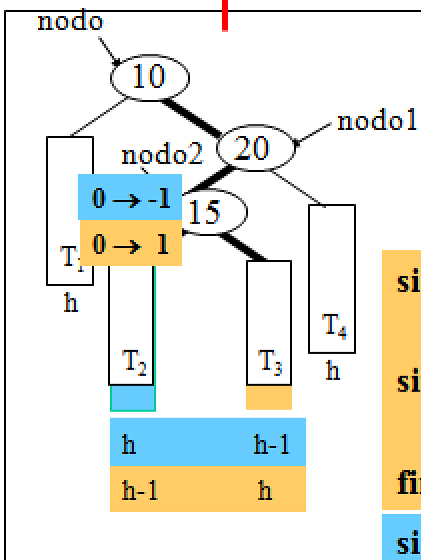


nodo↑.der ← nodo2↑.izq
nodo2 ↑.izq ← nodo
nodo1↑.izq ← nodo2↑.der
nodo2↑.der ← nodo1



ANTES INSERCIÓN		DESPUÉS INSERCIÓN POR RAMA IZQUIERDA			DESPUÉS INSERCIÓN POR RAMA DERECHA (nodo1=nodo↑.der)		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	+1	h+2	verdadero
1	h+2	0	h+2	falso	**nodo1↑.fe = -1 ⇒ rotación DI**		
-1	h+2	**¿II o ID? Depende de FE nodo↑.izq**			0	h+2	falso

Rotación DERECHA IZQUIERDA
2. Cambio de factores de equilibrio



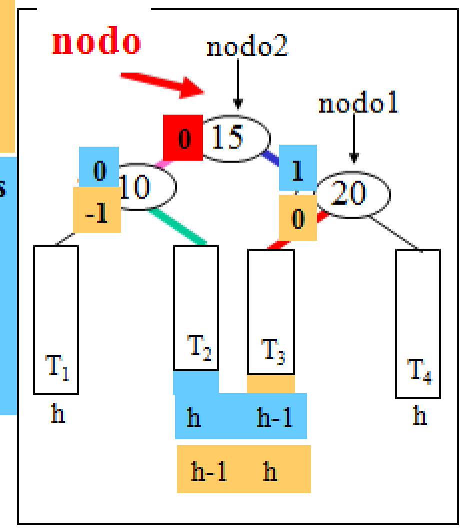
Rotación DERECHA-IZQUIERDA

nodo↑.der ← nodo2↑.izq
nodo2 ↑.izq ← nodo
nodo1↑.izq ← nodo2↑.der
nodo2↑.der ← nodo1

si nodo2↑.fe= 1 **entonces**
  nodo↑.fe ← -1
sino
  nodo↑.fe ← 0
fin

si nodo2↑.fe=- 1 **entonces**
  nodo1↑.fe ← 1
sino
  nodo1↑.fe ← 0
fin

nodo ← nodo2
nodo ↑.FE ← 0



Algoritmo de rotación DERECHA IZQUIERDA

```
procedimiento rotacionDI(ref nodo: puntero_nodo,
        nodo1,nodo2: punteroNodo)
    nodo↑.der ← nodo2↑.izq
    nodo2↑.izq ← nodo
    nodo1↑.izq ← nodo2↑.der
    nodo2↑der ← nodo1
    si  nodo2↑.fe=1 entonces
        nodo↑.fe ← -1
    sino
        nodo↑.fe ← 0
    fin si
    si  nodo2↑.fe=-1 entonces
        nodo1↑.fe ← 1
    sino
        nodo1↑.fe ← 0
    fin si
    nodo ← nodo2
    nodo↑.fe ← 0
```

Movimiento nodos en rotación

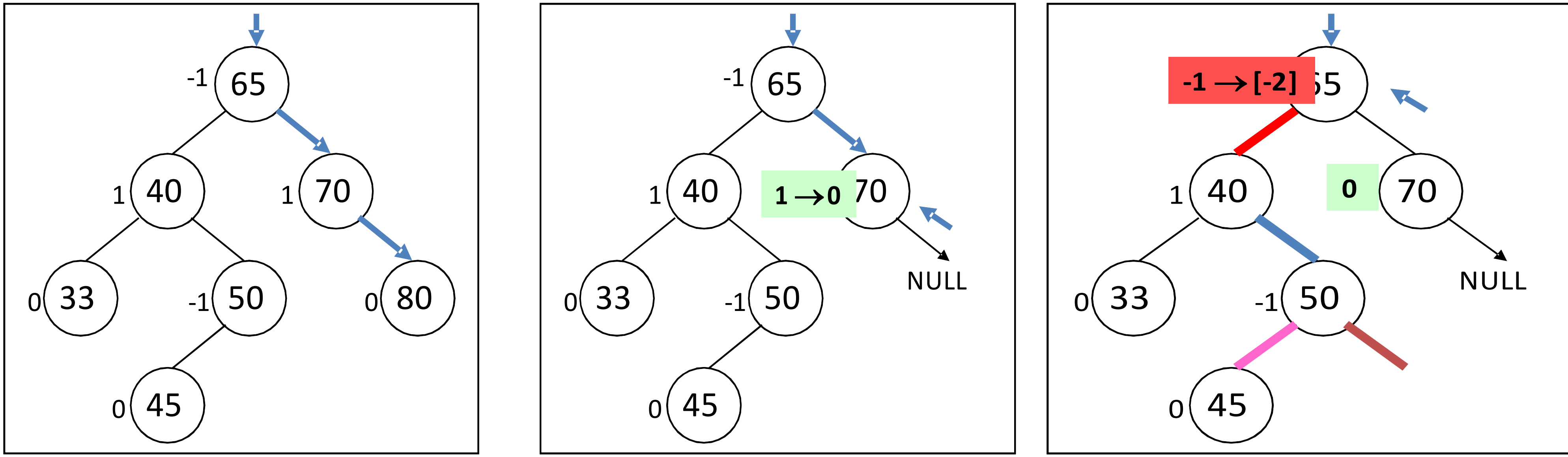
Cambio en los factores de equilibrio

Asignación de la nueva raíz del subárbol y de su factor de equilibrio

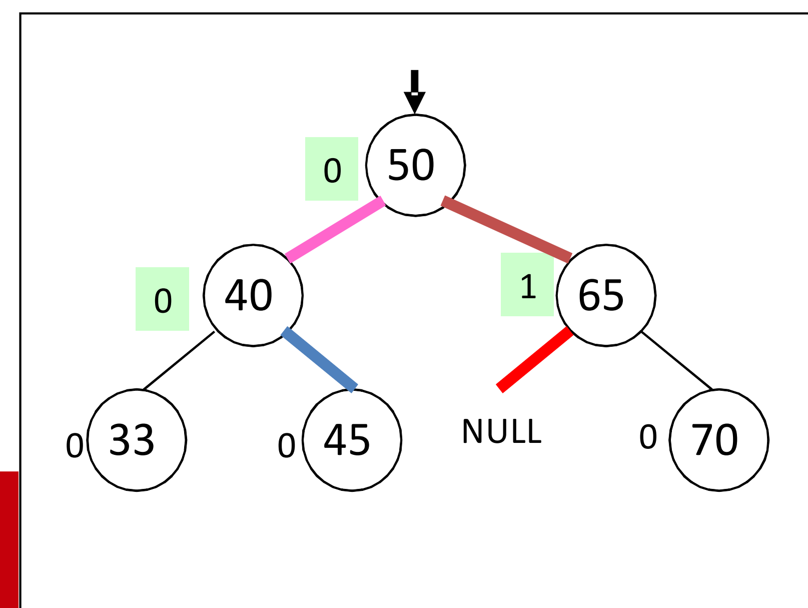
Eliminación en árboles balanceados

- Operación que elimina un nodo de un árbol balanceado sin violar los principios que lo definen
- El algoritmo sigue la misma lógica que el algoritmo de eliminación en a.b.b. añadiéndole las operaciones de reestructuración utilizadas en el algoritmo de inserción en árboles balanceados (rotaciones II, DD, ID y DI). Resulta bastante más complejo
- A diferencia del algoritmo de inserción en árboles balanceados, una vez efectuada una rotación el algoritmo puede continuar, pues se puede producir más de una rotación en el retroceso realizado por el camino de búsqueda, pudiendo llegar hasta la raíz del árbol
- El algoritmo de inserción se detenía después de una rotación ya que la altura de los subárboles después de una rotación es la misma que antes de la inserción y la rotación

Ejemplo: eliminación del nodo con clave 80



Eliminación DERECHA ⇒EQUILIBRAR IZQUIERDA



Procedimiento de ELIMINACIÓN

- Localizar en el árbol la posición del nodo que se desea eliminar teniendo en cuenta los mismos casos que en el procedimiento de eliminación de un a.b.b. (el nodo a suprimir es hoja, tiene un único descendiente o tiene dos descendientes)
- **Regresar** por el camino de búsqueda calculando el factor de equilibrio de los nodos visitados. Si en alguno de ellos se viola el criterio de equilibrio (FE = -2 ó FE = 2) el árbol debe reestructurarse

- Puede implementarse como un algoritmo recursivo que tiene como parámetros:
  - el valor de la clave del nodo que se desea eliminar
  - un puntero que inicialmente señala a la raíz del árbol que permitirá seguir el camino de búsqueda
  - un parámetro de tipo lógico que indicará si la altura del árbol ha cambiado (disminuido) bien por la eliminación de un nodo bien por la reestructuración

Algoritmo de eliminación en árboles balanceados

```
procedimiento eliminar(x: tipoClave, ref cambiaH:tipo_lógico, ref  nodo: punteroNodo)
 aux : punteroNodo
 si nodo = NULO entonces
    /* No existe nodo con clave x en el árbol: Impl. según especif.*/
 sino, si x < nodo↑.clave entonces
        eliminar(x, cambiaH, nodo↑.izq)
        si cambiaH entonces
          equilibarDER(nodo, cambiaH)
        fin si
    sino, si x > nodo↑.clave entonces
        eliminar(x, cambiaH, nodo↑.der)
        si cambiaH entonces
           equilibrarIZQ(nodo, cambiaH)
        fin si
    sino
      aux ← nodo
      si aux↑.der = NULO entonces /* sólo hijo izquierdo o ningún descendiente*/
          nodo ← aux ↑ .izq
          eliminarNodo(aux)
          cambiaH ← VERDADERO
      sino, si aux↑.izq = NULO entonces /* sólo hijo derecho*/
          nodo ← aux ↑ .der
          eliminarNodo(aux)
          cambiaH ← VERDADERO
      sino /* dos hijos*/
          borrar(nodo, aux↑.izq, aux, cambiaH)
          si cambiaH entonces
            equilibrarDER(nodo, cambiaH)
          fin si
      fin si
 fin si
```

Observaciones

- Al regresar por el camino de búsqueda se debe equilibrar, si es necesario, la rama opuesta a la eliminación
    - La rama derecha si se ha eliminado a la izquierda
    - La rama izquierda si se ha eliminado a la derecha
- Procedimiento borrar
    - elimina el nodo más a la derecha del subárbol izquierdo después de haber sustituido sus valores en el nodo que se pretendía eliminar
    - Regresa por el camino de búsqueda recalculando los factores de equilibrio de los nodos encontrados en el recorrido
    - Puede implementarse como un algoritmo recursivo con cuatro parámetros:
        - tres a puntero nodo para buscar el nodo mas derecha subárbol izquierdo y distinguir si ese nodo es o no la raíz del subárbol izquierdo.
        - un parámetro de tipo lógico para controlar el cambio de altura en el subárbol después de la eliminación

Algoritmo borrar

```
procedimiento borrar(ref nodo: punteroNodo, aux, ant: punteroNodo,
                     ref cambiaH: tipo_lógico)
 si aux↑.der ≠ NULO entonces
     borrar(nodo, aux↑.der, aux, cambiaH)
     si cambiaH entonces
       equilibrarIZQ(aux, cambiaH)
     fin si
 sino
   nodo↑.clave ← aux↑.clave
   nodo↑.información ← aux↑.información
   si ant↑.izq = aux entonces
       ant↑.izq ← aux↑.izq
   sino
       ant↑.der ← aux↑.izq
   fin si
   eliminarNodo(aux)
   cambiaH ← VERDADERO
 fin si
```

Bucar nodo más derecha subárbol izquierdo

Sustiuir información

Enlazar subárboles de aux ante de eliminar aux

EQUILIBAR SUBÁRBOL IZQUIERDO

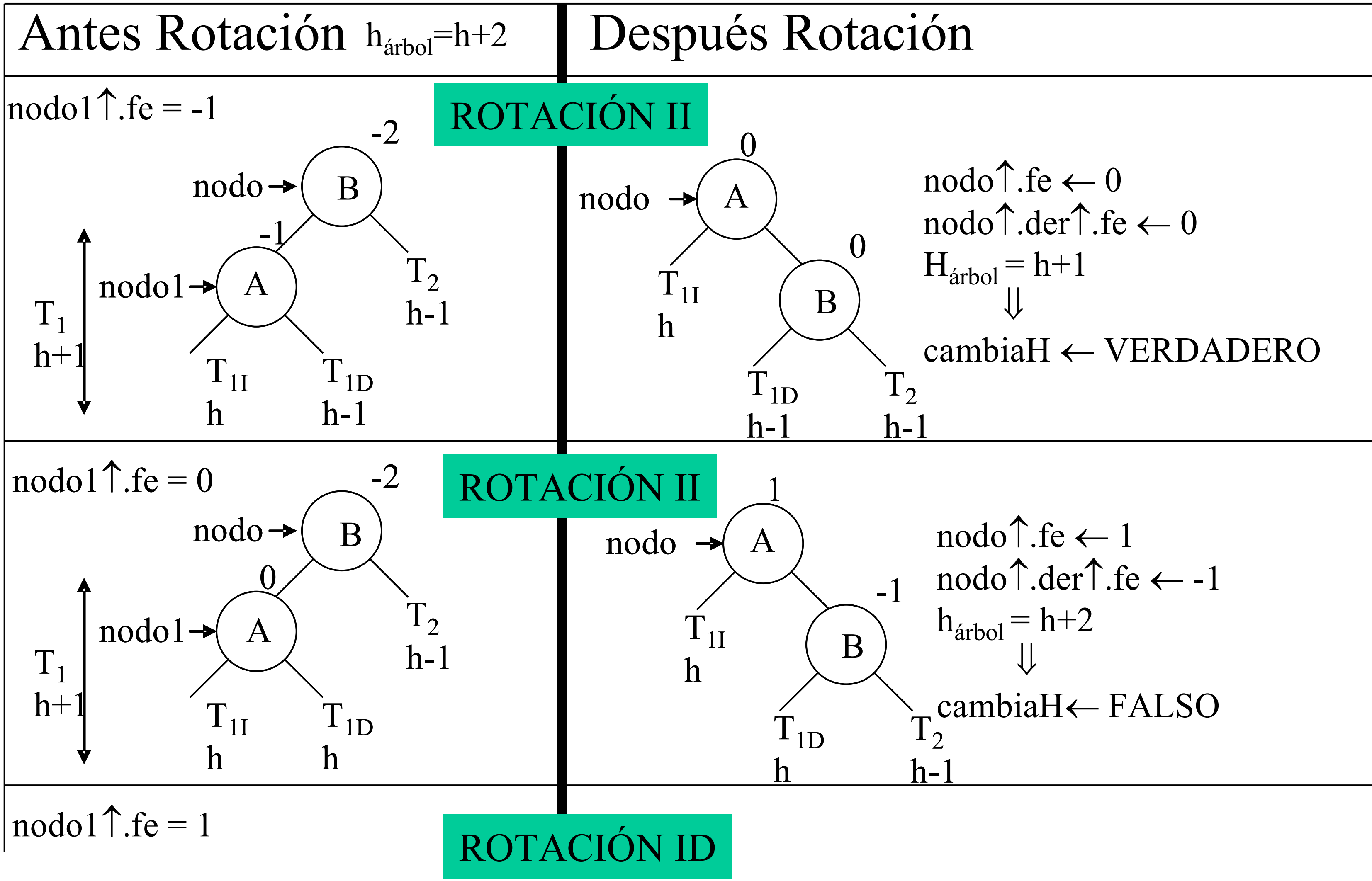
- Este proceso se efectúa al volver por el camino de búsqueda, después de haber eliminado un nodo en el subárbol derecho
- En esta vuelta se van recalculando los factores de equilibrio de los nodos encontrados en el camino
- Antes de que el factor de equilibrio de un nodo (raíz del subárbol) llegue a tomar el valor -2 se debe reestructurar el subárbol izquierdo que será de mayor altura
- Este proceso se implementa con un algoritmo con dos parámetros:
  - Puntero a la raíz del subárbol
  - Parámetro de tipo lógico para controlar el cambio de altura del subárbol
- Como siempre, habrá que tener en cuenta tres casos, los tres posibles valores del factor de equilibrio de la raíz del subárbol (-1,0,1)

Casos al equilibrar subárbol IZQUIERDO (Vuelta de eliminar por DERECHO)

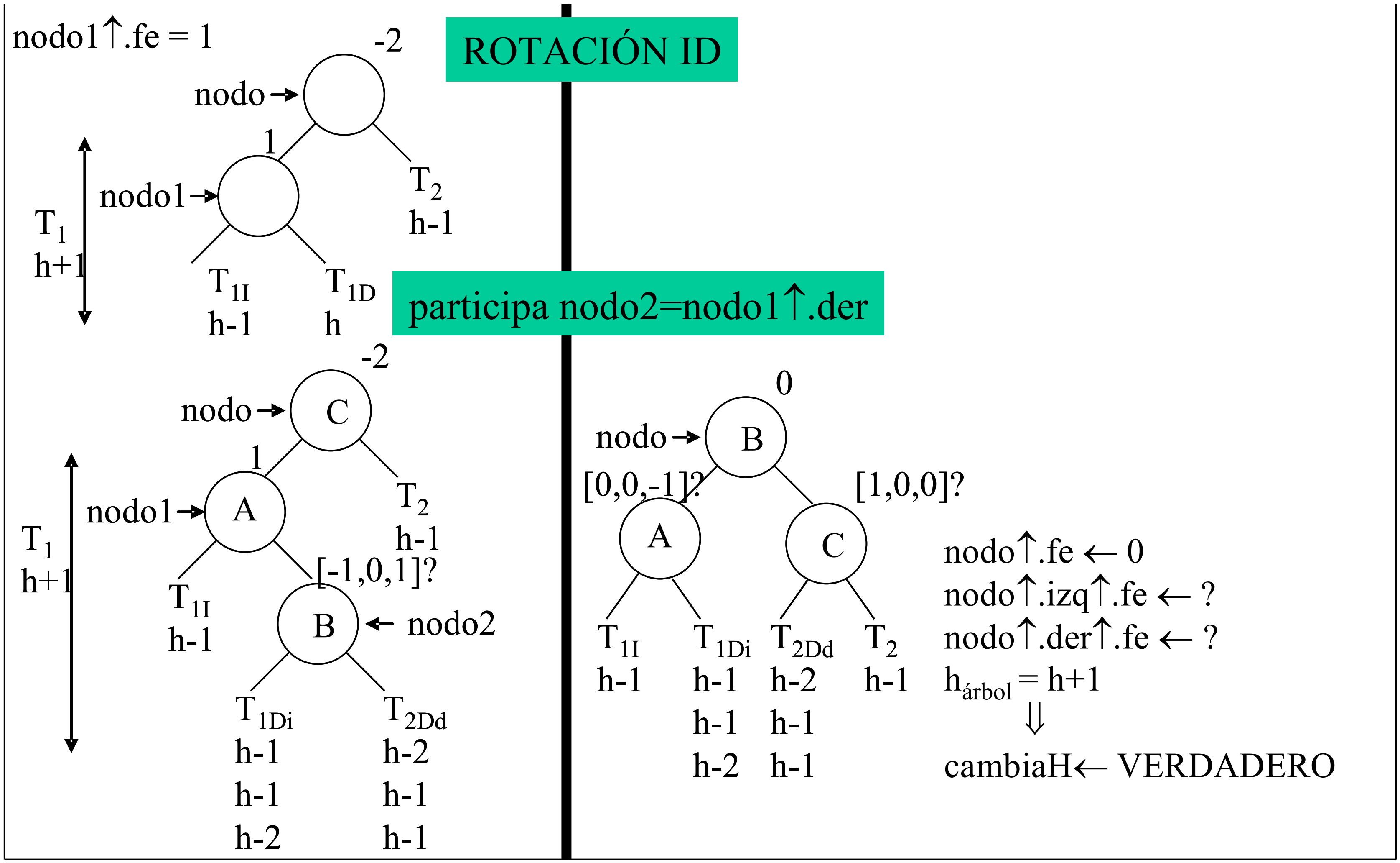
Antes Eliminación			Después Eliminación			
nodo↑.fe	Subárbol	$H_{árbol}$	Subárbol	nodo↑.fe	$H_{árbol}$	CAMBIA_H
0 $H_{RI}=H_{RD}$	nodo → 0; T_1 h, T_2 h	h+1	nodo → -1; T_1 h, T_2 h-1	-1	h+1	FALSO
1 $H_{RI}<H_{RD}$	nodo → 1; T_1 h, T_2 h+1	h+2	nodo → 0; T_1 h, T_2 h	0	h+1	VERDADERO
-1 $H_{RI}>H_{RD}$	nodo → -1; T_1 h+1, T_2 h	h+2	nodo → **-2**; T_1 h+1, T_2 h-1	**REESTRUCTURACIÓN SUBÁRBOL IZQUIERDO. Rotación? Depende nodo↑.izq↑.fe nodo1 ← nodo↑.izq**		

ANTES ELIMINACIÓN		DESPUÉS ELIMINACIÓN POR RAMA IZQUIERDA			DESPUÉS ELIMINACIÓN POR RAMA DERECHA		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1				-1	h+1	falso
1	h+2				0	h+1	verdadero
-1	h+2				Reestruct. SUB. IZQUIERDO		

Reestructuración del subárbol izquierdo después de una eliminación(I)



Reestructuración del subárbol izquierdo después de una eliminación(II)



Resumen casos al equilibar el subárbol IZQUIERDO (Vuelta de eliminar por DERECHO)

ANTES ELIMINACIÓN		DESPUÉS ELIMINACIÓN POR RAMA DERECHA					
nodo$\uparrow$.fe	altura	nodo$\uparrow$.fe				altura	cambiaH
0	h+1	-1				h+1	falso
1	h+2	0				h+1	verdadero
-1	h+2	**-2 Reestructuración SUB. IZQUIERDO** nodo1 $\leftarrow$ nodo$\uparrow$.izq					
		nodo1$\uparrow$.fe		nodo$\uparrow$.fe	nodo1$\uparrow$.fe		
		-1	Rotación II(*)	0	0	h+1	verdadero
		0	Rotación II(*)	-1	1	h+2	faslo
		1	Rotación ID			h+1	verdadero

(*) Modificada para la eliminación

Algoritmo rotación IZQUIERDA IZQUIERDA modificado para eliminación

```
procedimiento rotaciónII(ref nodo: punteroNodo, nodo1: punteroNodo)
inicio
  nodo↑.izq ← nodo1↑.der
  nodo1↑.der ← nodo
  si nodo1↑.fe = -1 entonces    /* SIEMPRE: si la rotación es por una  inserción */
        nodo↑.fe ← 0
        nodo1↑.fe ← 0
  sino
        nodo↑.fe ← -1
        nodo1↑.fe ← 1
  fin si
  nodo ← nodo1
fin
```

ANTES ELIMINACIÓN		DESPUÉS ELIMINACIÓN POR RAMA DERECHA					
nodo↑.fe	altura	nodo↑.fe				altura	cambiaH
0	h+1	-1				h+1	falso
1	h+2	0				h+1	verdadero
-1	h+2	-2 Reestructuración SUB. IZQUIERDO nodo1 ← nodo↑.izq					
		nodo1↑.fe		nodo↑.fe	nodo1↑.fe		
		-1	Rotación II(*)	0	0	h+1	verdadero
		0	Rotación II(*)	-1	1	h+2	faslo
		1	Rotación ID			h+1	verdadero

Operación EQUILIBRAR RAMA IZQUIERDA (eliminación por derecha)

procedimiento equilibrarIZQ(**ref** nodo:punteroNodo, **ref** cambiaH: tipo_lógico)

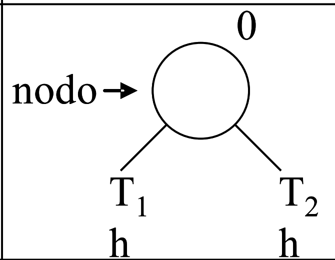
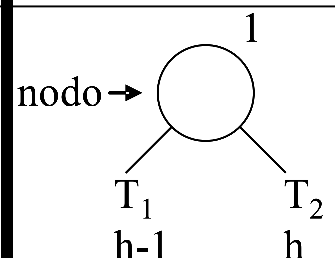
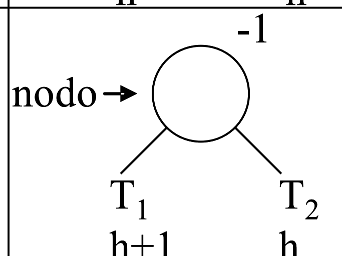
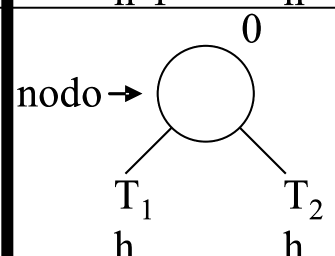
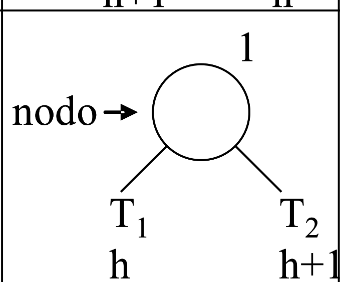
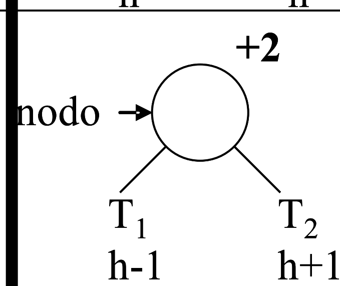
```
    nodo1,nodo2:punteroNodo
    caso nodo↑.fe en
       1 :  nodo↑.fe ← 0
       0 :  nodo↑.fe ← -1
            cambiaH ← FALSO
      -1:   nodo1↑ ← nodo.izq
            si nodo1↑.fe <= 0 entonces
               rotaciónII(nodo, nodo1)
               si nodo1↑.fe = 0 entonces
                  cambiaH ← FALSO
               fin si
            sino
               nodo2 ← nodo1↑.der
               rotaciónID(nodo, nodo1, nodo2)
            fin si
    fin caso
```

<table>
<tr><th colspan="2">ANTES ELIMINACIÓN</th><th colspan="6">DESPUÉS ELIMINACIÓN POR RAMA DERECHA</th></tr>
<tr><th>nodo↑.fe</th><th>altura</th><th colspan="4">nodo↑.fe</th><th>altura</th><th>cambiaH</th></tr>
<tr><td>0</td><td>h+1</td><td colspan="4">-1</td><td>h+1</td><td>falso</td></tr>
<tr><td>1</td><td>h+2</td><td colspan="4">0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td rowspan="5">-1</td><td rowspan="5">h+2</td><td colspan="4">-2 Reestructuración SUB. IZQUIERDO
nodo1 ← nodo↑.izq</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>nodo1↑.fe</td><td></td><td>nodo↑.fe</td><td>nodo1↑.fe</td></tr>
<tr><td>-1</td><td>Rotación II(*)</td><td>0</td><td>0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td>0</td><td>Rotación II(*)</td><td>-1</td><td>1</td><td>h+2</td><td>faslo</td></tr>
<tr><td>1</td><td colspan="3">Rotación ID</td><td>h+1</td><td>verdadero</td></tr>
</table>

EQUILIBAR SUBÁRBOL DERECHO

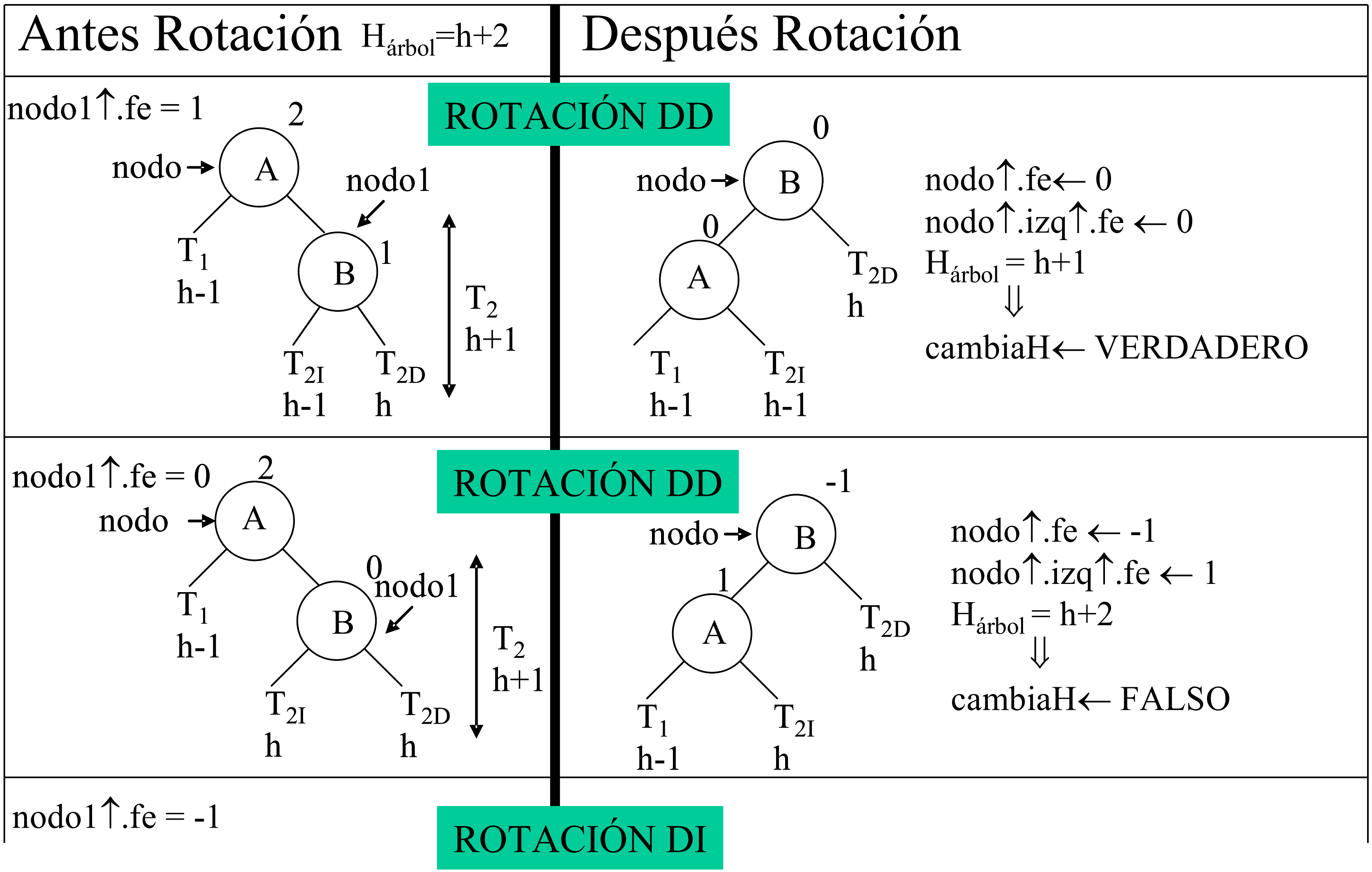
- Este proceso se efectúa al regresar por el camino de búsqueda, después de haber eliminado un nodo en el subárbol izquierdp
- En esta vuelta se van recalculando los factores de equilibrio de los nodos encontrados en el camino
- Antes de que el factor de equilibrio de un nodo (raíz del subárbol) llegue a tomar el valor +2 se debe reestructurar el subárbol derecho que será de mayor altura
- Este proceso se implementa con un algoritmo con dos parámetros:
  - Puntero a la raíz del subárbol
  - Parámetro de tipo lógico para controlar el cambio de altura del subárbol
- De nuevo, habrá que tener en cuenta tres casos, los tres posibles valores del factor de equilibrio de la raíz del subárbol (-1,0,1)

Casos al equilibrar el subárbol DERECHO(Vuelta de eliminar por IZQUIERDO)

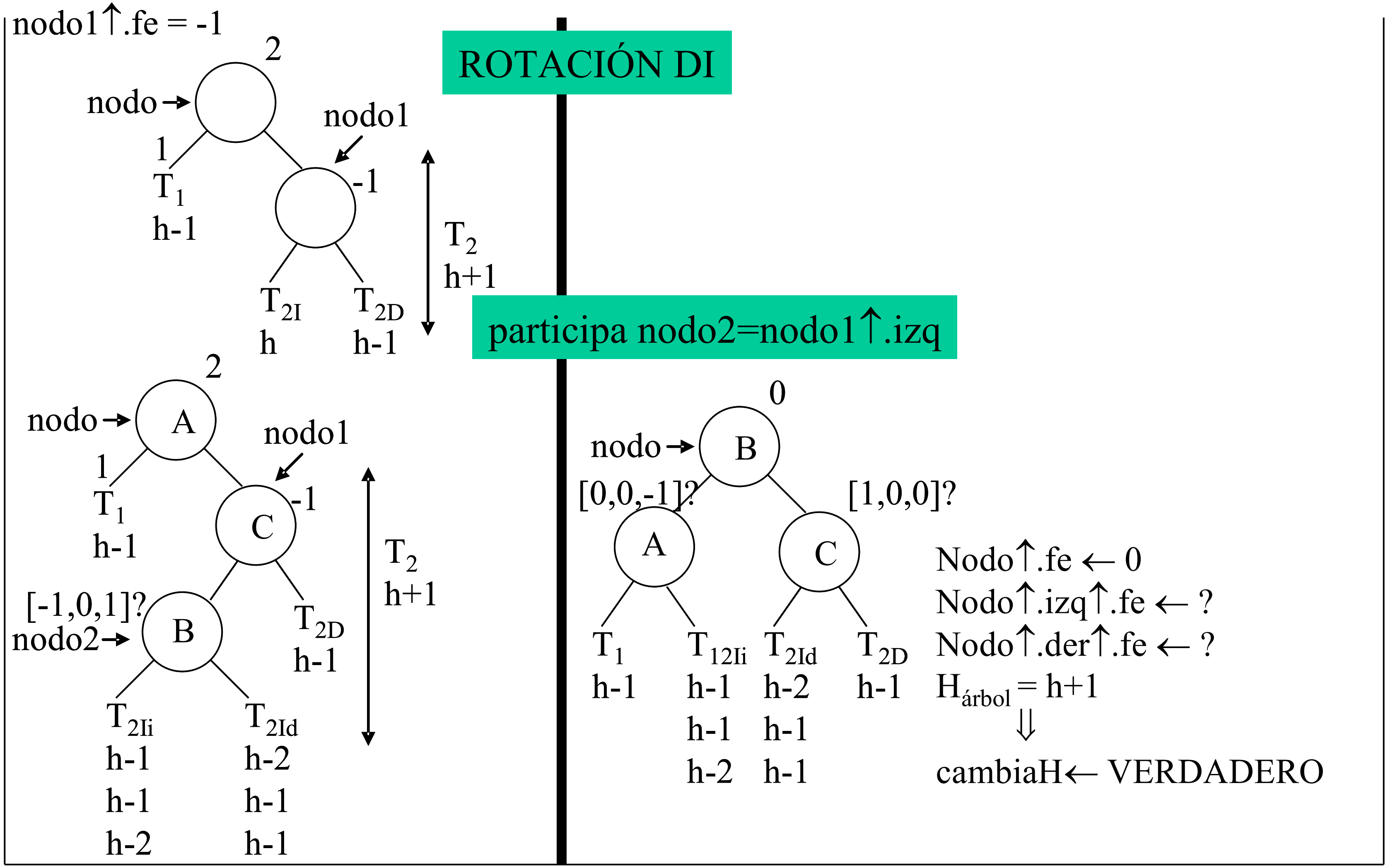
Antes Eliminación			Después Eliminación			
nodo↑.fe	Subárbol	$H_{árbol}$	Subárbol	nodo↑.fe	$H_{árbol}$	cambiaH
0 $H_{RI}=H_{RD}$		h+1		1	h+1	FALSO
-1 $H_{RI}>H_{RD}$		h+2		0	h+1	VERDADERO
1 $H_{RI}<H_{RD}$		h+2		**REESTRUCTURACIÓN SUBÁRBOL DERECHO. Rotación? Depende nodo↑.der↑.fe nodo1 ← nodo↑.der**		

ANTES ELIMINACIÓN		DESPUÉS ELIMINACIÓN POR RAMA IZQUIERDA			DESPUÉS ELIMINACIÓN POR RAMA DERECHA		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	1	h+1	falso	-1	h+1	falso
1	h+2	**Reestruct. SUB. DERECHO**			0	h+1	verdadero
-1	h+2	0	h+1	verdadero	**Reestruct. SUB. IZQUIERDO**		

Reestructuración del subárbol derecho después de una eliminación(I)



Reestructuración del subárbol derecho después de una eliminación(II)



Resumen casos al equilibrar el subárbol DERECHO (Vuelta de eliminar por IZQUIERDO)

<table>
<tr><th colspan="2">ANTES ELIMINACIÓN</th><th colspan="6">DESPUÉS ELIMINACIÓN POR RAMA IZQUIERDA</th></tr>
<tr><th>nodo↑.fe</th><th>altura</th><th colspan="4">nodo↑.fe</th><th>altura</th><th>cambiaH</th></tr>
<tr><td>0</td><td>h+1</td><td colspan="4">1</td><td>h+1</td><td>falso</td></tr>
<tr><td>-1</td><td>h+2</td><td colspan="4">0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td rowspan="5">1</td><td rowspan="5">h+2</td><td colspan="4">+2 Reestructuración SUB. DERECHO
nodo1 ← nodo↑.der</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>nodo1↑.fe</td><td colspan="2">nodo↑.fe</td><td>nodo1↑.fe</td></tr>
<tr><td>1</td><td>Rotac. DD(*)</td><td>0</td><td>0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td>0</td><td>Rotac. DD(*)</td><td>1</td><td>-1</td><td>h+2</td><td>faslo</td></tr>
<tr><td>-1</td><td colspan="3">Rotación DI</td><td>h+1</td><td>verdadero</td></tr>
</table>

(*) Modificada para la eliminación

Algoritmo rotación DERECHA DERECHA modificado para eliminación

procedimiento rotaciónDD(**ref** nodo: punteroNodo, nodo1: punteroNodo)

```
nodo↑.der ← nodo1↑.izq
nodo1↑.izq ← nodo
si nodo1↑.fe = 1 entonces /* SIEMPRE: si la rotación es por una inserción */
        nodo↑.fe ← 0
        nodo1↑.fe ← 0
sino
        nodo↑.fe ← 1
        nodo1↑.fe ← -1
fin si
nodo ← nodo1
```

<table>
<tr><th colspan="2">ANTES ELIMINACIÓN</th><th colspan="5">DESPUÉS ELIMINACIÓN POR RAMA IZQUIERDA</th></tr>
<tr><th>nodo↑.fe</th><th>altura</th><th colspan="3">nodo↑.fe</th><th>altura</th><th>cambiaH</th></tr>
<tr><td>0</td><td>h+1</td><td colspan="3">1</td><td>h+1</td><td>falso</td></tr>
<tr><td>-1</td><td>h+2</td><td colspan="3">0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td rowspan="5">1</td><td rowspan="5">h+2</td><td colspan="3">+2 Reestructuración SUB. DERECHO
nodo1 ← nodo↑.der</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>nodo1↑.fe</td><td>nodo↑.fe</td><td>nodo1↑.fe</td></tr>
<tr><td>1</td><td>Rotac. DD(*) 0</td><td>0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td>0</td><td>Rotac. DD(*) 1</td><td>-1</td><td>h+2</td><td>faslo</td></tr>
<tr><td>-1</td><td colspan="2">Rotación DI</td><td>h+1</td><td>verdadero</td></tr>
</table>

Operación EQUILIBRAR RAMA DERECHA(eliminación por izquierda)

```
procedimiento equilibrarDER( ref nodo:punteroNodo, ref cambiaH: tipo_lógico)
    nodo1,nodo2:punteroNodo
    caso nodo↑.fe en
       -1 :   nodo↑.fe ← 0
        0 :   nodo↑.fe ← 1
              cambiaH ← FALSO
        1 :   nodo1↑ ← nodo ↑.der
              si nodo1↑.fe ≥ 0 entonces
                   rotaciónDD(nodo, nodo1)
                   si  nodo1↑.fe = 0 entonces
                     cambiaH ← FALSO
                   fin si
              sino
                 nodo2 ← nodo1↑.izq
                   rotaciónDI(nodo, nodo1, nodo2)
              fin si
```

<table>
<tr><th colspan="2">ANTES ELIMINACIÓN</th><th colspan="5">DESPUÉS ELIMINACIÓN POR RAMA IZQUIERDA</th></tr>
<tr><th>nodo↑.fe</th><th>altura</th><th colspan="3">nodo↑.fe</th><th>altura</th><th>cambiaH</th></tr>
<tr><td>0</td><td>h+1</td><td colspan="3">1</td><td>h+1</td><td>falso</td></tr>
<tr><td>-1</td><td>h+2</td><td colspan="3">0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td rowspan="5">1</td><td rowspan="5">h+2</td><td colspan="3">+2 Reestructuración SUB. DERECHO
nodo1 ← nodo↑.der</td><td rowspan="2"></td><td rowspan="2"></td></tr>
<tr><td>nodo1↑.fe</td><td>nodo↑.fe</td><td>nodo1↑.fe</td></tr>
<tr><td>1</td><td>Rotac. DD(*) 0</td><td>0</td><td>h+1</td><td>verdadero</td></tr>
<tr><td>0</td><td>Rotac. DD(*) 1</td><td>-1</td><td>h+2</td><td>faslo</td></tr>
<tr><td>-1</td><td colspan="2">Rotación DI</td><td>h+1</td><td>verdadero</td></tr>
</table>

EJERCICIOS

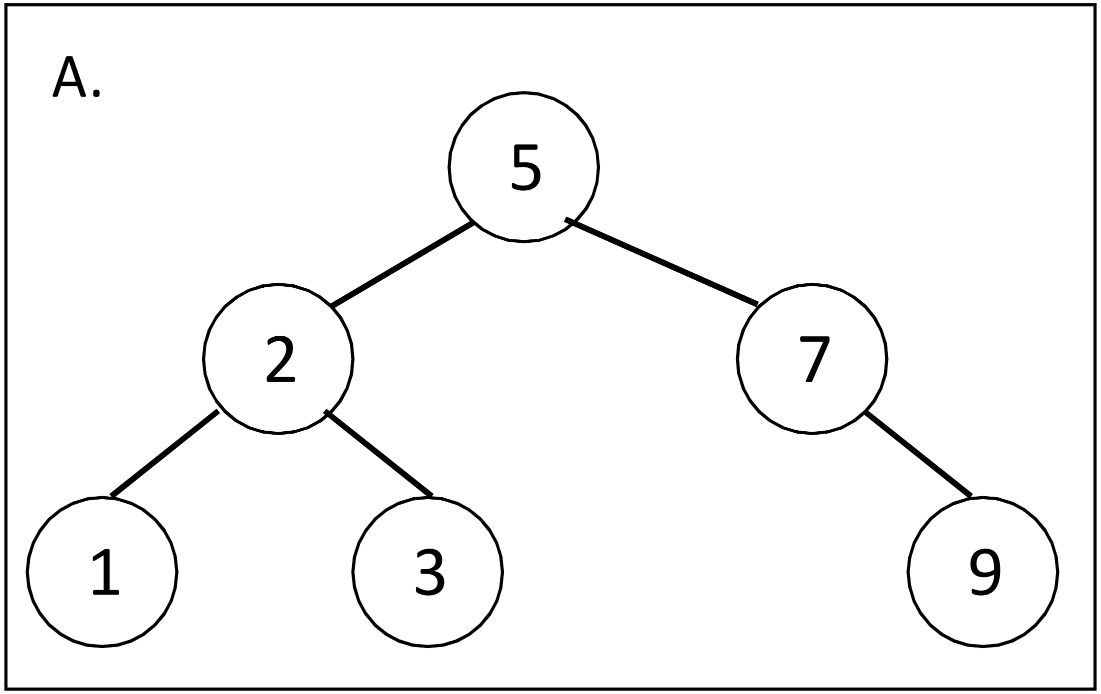
1. Crear el árbol binario de búsqueda que resulta al insertar las siguientes claves:

   A. 5,7,2,3,9,1

   B. 1,2,3,5,7,9

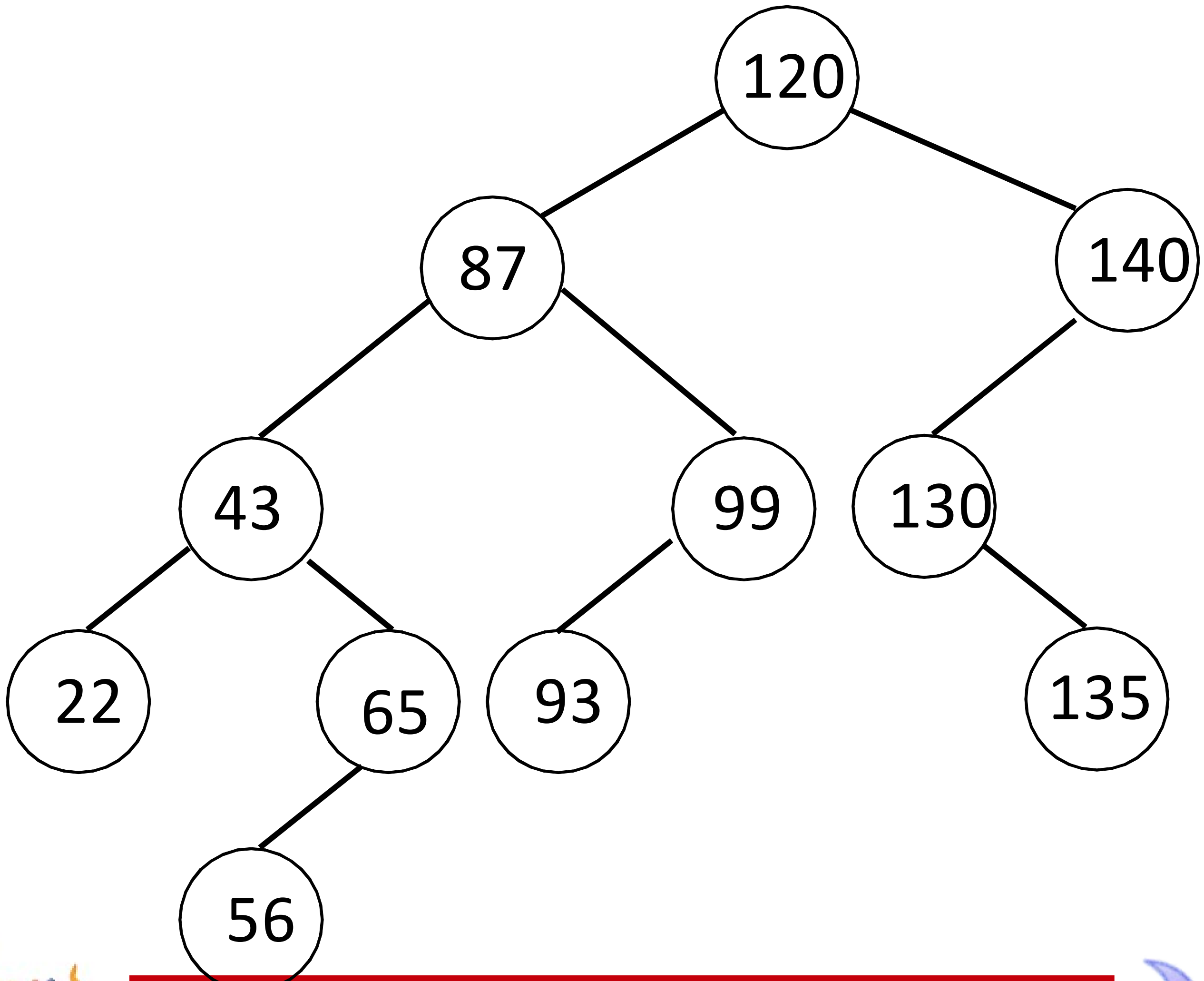
   C. 9,7,5,3,2,1

   D. En cualquier otro orden



EJERCICIOS

2. Eliminar del a.b.búsqueda de la figura los nodos con las claves: 22, 99, 87, 120, 140, 135 y 56



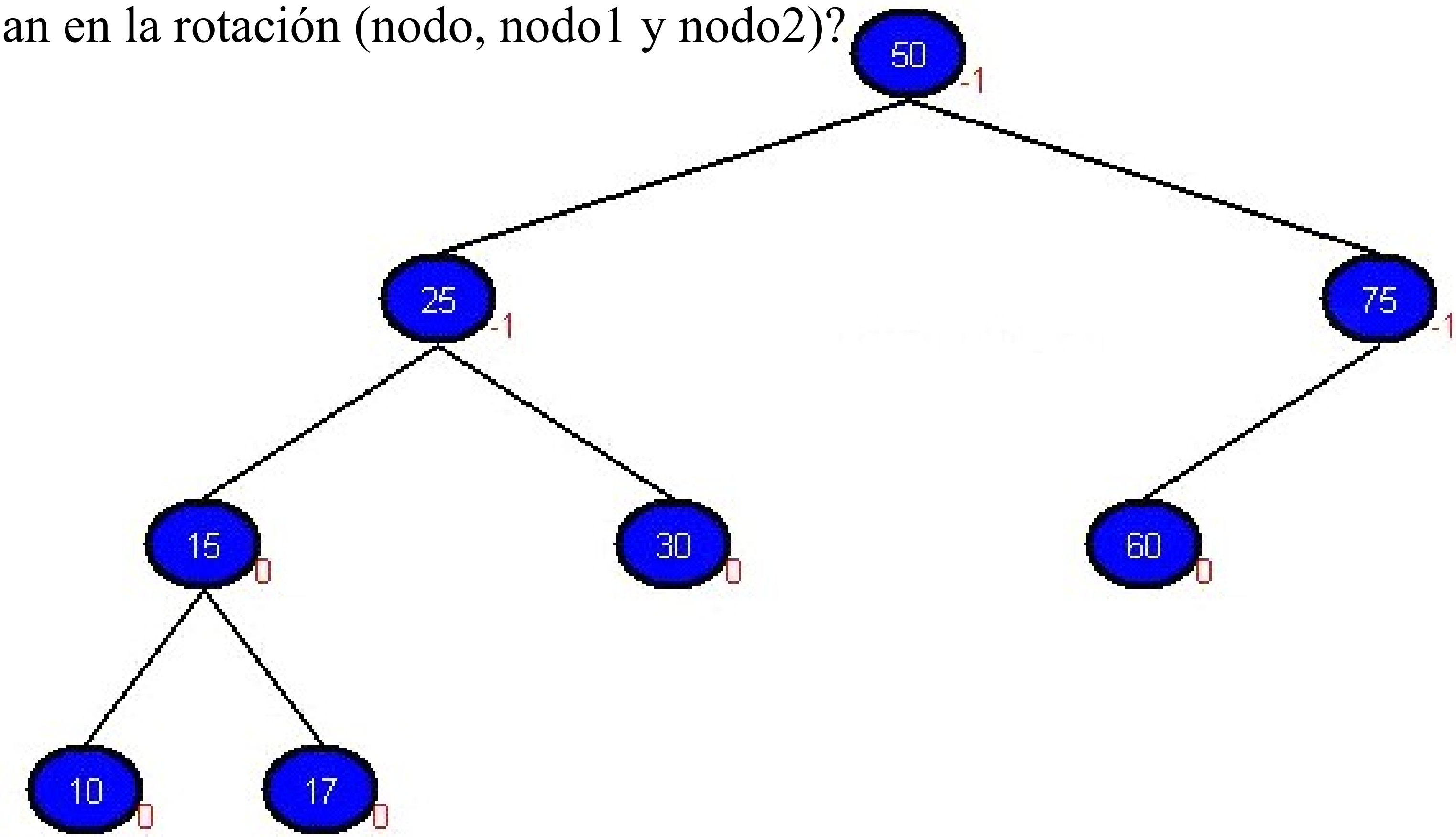
EJERCICIOS

3. Dado el árbol balanceado de la figura.

a) Insertar un nodo con valor 12

b) Insertar un nodo con valor 20

Indicar que tipo de rotación se produce y mostrar gráficamente el árbol resultante. ¿Qué nodos participan en la rotación (nodo, nodo1 y nodo2)?



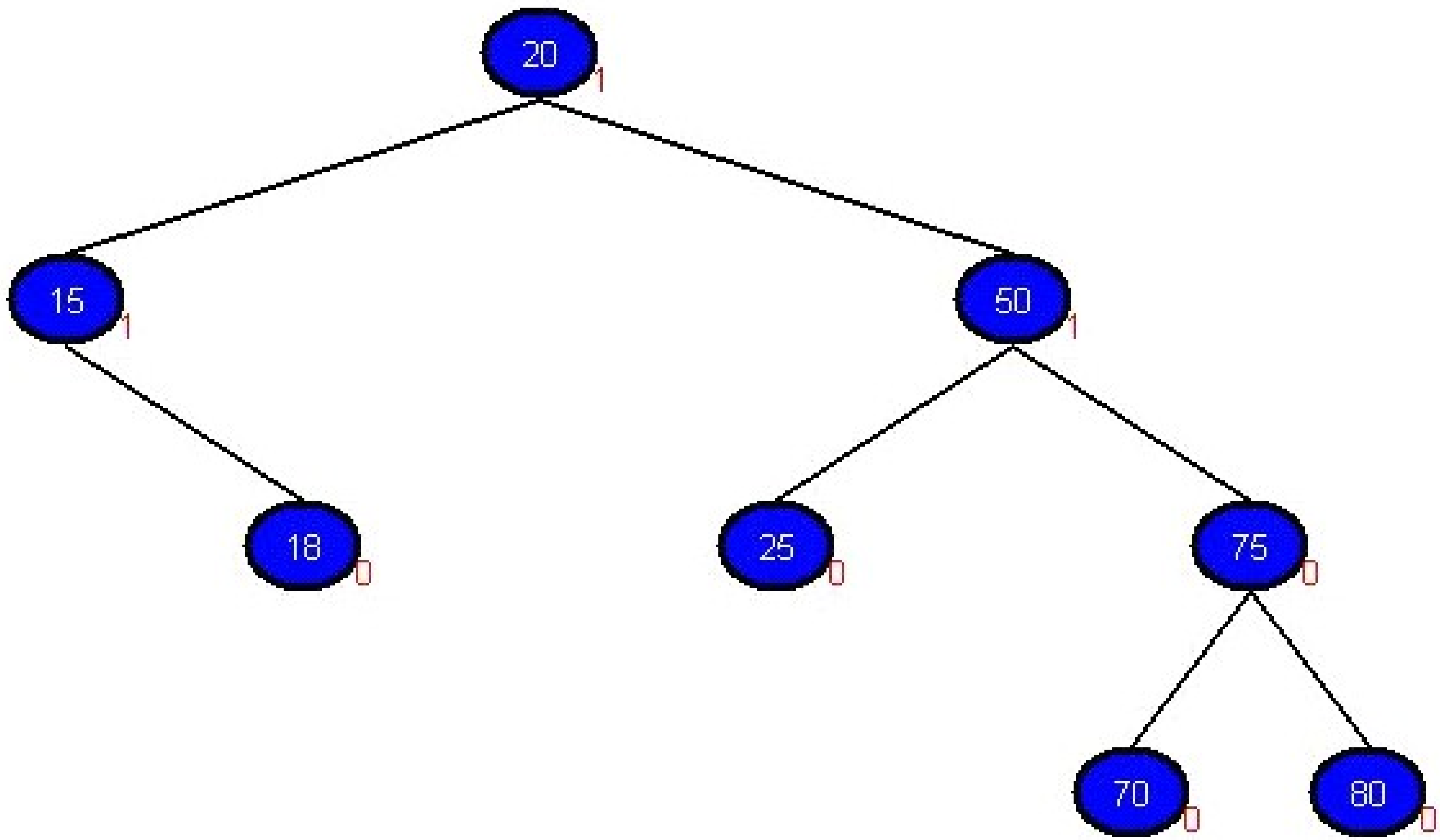
EJERCICIOS

4. Dado el árbol balanceado de la figura.

   a) Insertar un nodo con valor 90

   b) Insertar un nodo con valor 60

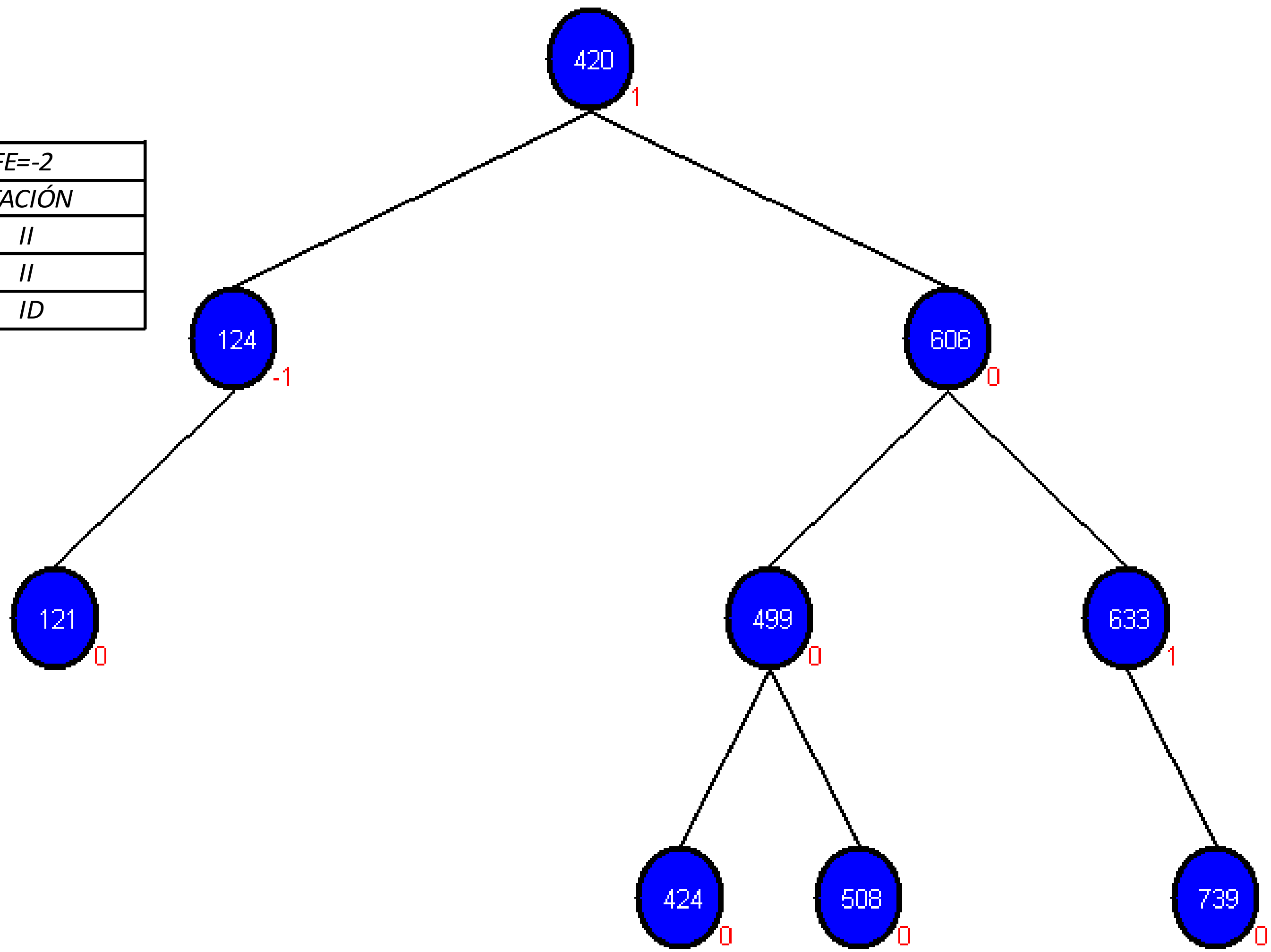
Indicar que tipo de rotación se produce y mostrar gráficamente el árbol resultante ¿Qué nodos participan en la rotación (nodo, nodo1 y nodo2)?



EJERCICIOS

5. Dado el árbol balanceado de la figura y, teniendo en cuenta la tabla que muestra todos los casos de rotaciones que se pueden presentar ante una eliminación en un árbol balanceado, indicar que tipo de rotación se produce y mostrar cómo queda el árbol después de eliminar el nodo con valor 121.

nodo.FE = 2		nodo.FE=-2	
nodo.der.FE	*ROTACIÓN*	*nodo.izq.FE*	*ROTACIÓN*
1	*DD*	*-1*	*II*
0	*DD*	*0*	*II*
-1	*DI*	*1*	*ID*



EJERCICIOS

6. Dado el árbol balanceado de la figura y, teniendo en cuenta la tabla que muestra todos los casos de rotaciones que se pueden presentar ante una eliminación en un árbol balanceado, indicar que tipo de rotación se produce y mostrar cómo queda el árbol después de eliminar el nodo con valor 749.

nodo.FE = 2		nodo.FE=-2	
nodo.der.FE	ROTACIÓN	nodo.izq.FE	ROTACIÓN
1	DD	-1	II
0	DD	0	II
-1	DI	1	ID