Tema 5. ÁRBOLES BINARIOS DE BÚSQUEDA

Estructuras de Datos y Algoritmos II
Grado en Ingeniería Informática

M José Polo Martín
mjpolo@usal.es

Universidad de Salamanca

curso 2022-2023

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

800 AÑOS 1218~2018

DEPARTAMENTO DE iNFORMÁTiCA Y AVTOMÁTiCA

Contenido

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)

Definición formal de **Árbol Binario de Búsqueda**

Árbol binario que o bien es nulo o cada nodo contiene una clave que satisface las siguientes condiciones:

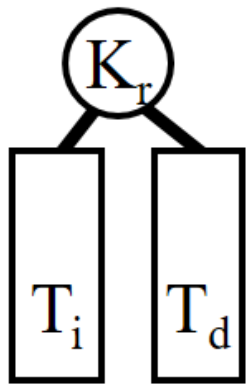
1. Todas las claves, si las hay, en el subárbol izquierdo de la raíz preceden a la clave de la raíz

$$K_i < K_r \; \forall K_i \in T_i$$

2. La clave de la raíz precede a todas las claves, si las hay, que contiene el subárbol derecho

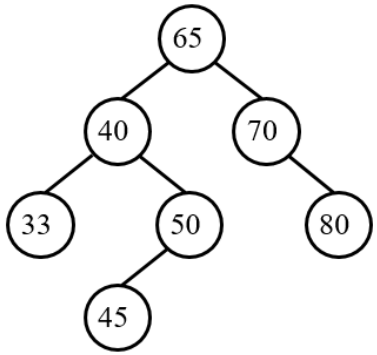
$$K_r < K_d \; \forall K_d \in T_d$$

3. Los subárboles izquierdo y derecho de la raíz son también árboles binarios de búsqueda



(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
- Una vez decidida la clave que se inserta en primer lugar las propiedades del árbol determinan donde deben insertarse las siguientes
- La estructura de un árbol binario de búsqueda particular está determinada por el orden en que se insertan los nodos en el árbol
- 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

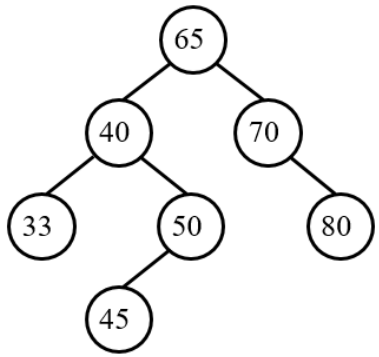
Operaciones básicas sobre árboles binarios de búsqueda

- **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
- **inserción(n, A):** añade el nodo n al árbol binario de búsqueda A. Después de la inserción A continuará siendo un árbol binario de búsqueda
- **eliminación(k, A):** suprime el nodo con valor k en su campo clave del árbol binario de búsqueda A si existe. Después de la eliminación A continuará siendo un árbol binario de búsqueda

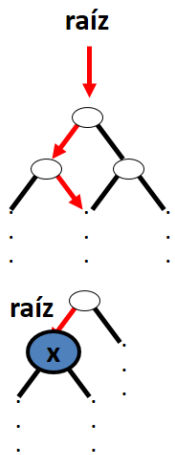
2 Nivel de representación e implementación

Algorithm declaraciones básicas

```
tipos
tipoNodo = registro
  clave: tipoClave
  información: tipoInformación
  izq,der:↑ tipoNodo
fin registro
tipoÁrbol: ↑tipoNodo
punteroNodo: ↑tipoNodo
```



- Recuperamos las declaraciones básicas del tema 1 añadiendo el campo clave necesario para la clasificiación
- La definición de a.b.b implica la existencia de procedimientos **recursivos**, partiendo de un puntero a la raíz del árbol, para
  - encontar en el árbol un nodo con un valor de clave dado
  - encontrar la posición en la que debe insertarse un nuevo nodo
  - eliminar del árbol un nodo con un valor de clave dado



2.1 Búsqueda en árboles binarios de búsqueda

- 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

Algorithm búsqueda(k:tipoClave, **raíz:tipoÁrbol**, **ref** nodo:tipoNodo)

Entrada: *raíz* dirección del nodo ráiz y *k* clave de búsqueda
Salida: *nodo* dirección del nodo con clave *k* si existe y *NULO* en caso contrario

```
 1: si raíz = NULO entonces
 2:     nodo ← NULO
 3: si no
 4:     si k = raíz↑.clave entonces
 5:         nodo ← raíz
 6:     si no
 7:         si k < raíz↑.clave entonces
 8:             búsqueda(k, raíz↑.izq,nodo)
 9:         si no
10:             búsqueda(k, raíz↑.der,nodo)
11:         fin si
12:     fin si
13: 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
    - cambiarán a medida que se vayan insertando nuevos nodos
    - 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 $\Rightarrow$ **Árboles Balanceados**

2.2 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 nueva 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

Algorithm insertar(nuevo:tipoNodo; **ref raíz:tipoÁrbol**)

Entrada: *nuevo* nodo a insertar en árbol raíz
Salida: el árbol *raíz* con el nodo nuevo insertado correctamente

```
 1: si raíz = NULO entonces
 2:     raíz ← nuevo
 3: si no
 4:     si nuevo↑.clave = raíz↑.clave entonces
 5:         /* clave dupliacada implementar según especificacion */
 6:     si no
 7:         si nuevo↑.clave < raíz↑.clave entonces
 8:             insertar(nuevo, raíz↑.izq)
 9:         si no
10:             insertar(nuevo, raíz↑.der)
11:         fin si
12:     fin si
13: fin si
```

Observaciones

- El parámetro *nuevo* es un puntero al nodo que ha de ser insertado en el a.b.b
- 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**
- Podría permitirse la inserción de claves duplicadas

2.3 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
  1. Localizar el nodo que se desea eliminar siguiendo el mismo método que en el procedimiento de búsqueda
  2. Distinguir los siguiente casos:

  caso 1 Si el nodo a eliminar es hoja o terminal, simplemente se suprime

  caso 2 Si el nodo a eliminar tiene un solo descendiente, se sustituye por ese descendiente y se suprime

  caso 3 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

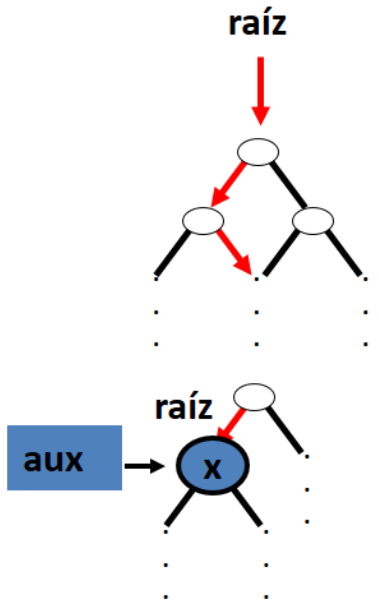
Algoritmo de eliminación en a.b.b.

Primer bosquejo

Algorithm eliminar(x:tipoClave; **ref raíz:tipoÁrbol**)

Entrada: *x* clave del nodo a eliminar del árbol *raíz*
Salida: el árbol *raíz* con el nodo eliminado si existe

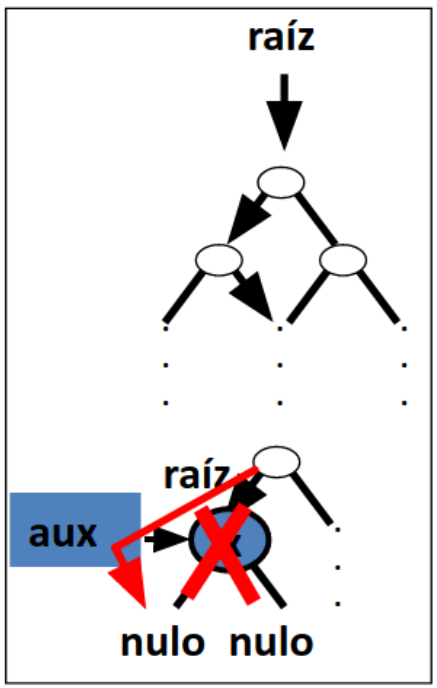
```
si raíz = NULO entonces
    /*no existe nodo con clave x: implementar según especificación*/
si no
    si x < raíz↑.clave entonces
        eliminar(x,raíz↑.izq)
    si no
        si x > raíz↑.clave entonces
            eliminar(x,raíz↑.der)
        si no
            aux ← raíz                    ->¡x=raíz↑ .clave!
            .....                         ->¡Distinguir casos!
            eliminarNodo(aux)
        fin si
    fin si
fin si
```



Distinción de casos: un descendiente o ninguno

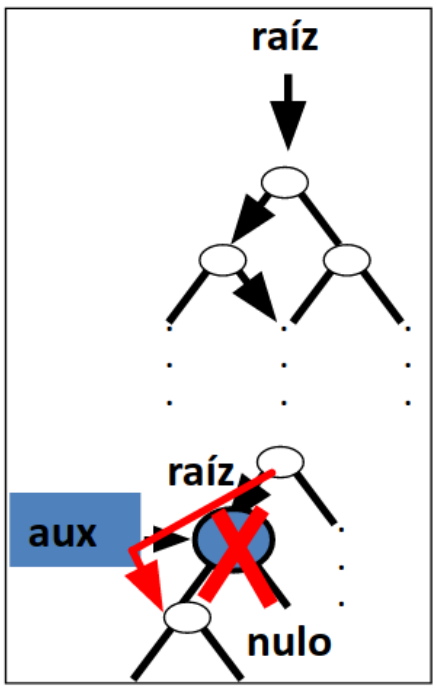
Ningún hijo	Sólo hijo izquierdo	Sólo hijo derecho
aux↑.izq=NULO aux↑.der=NULO	aux↑.izq≠NULO aux↑.der=NULO	aux↑.izq=NULO aux↑.der≠NULO

Ningún hijo



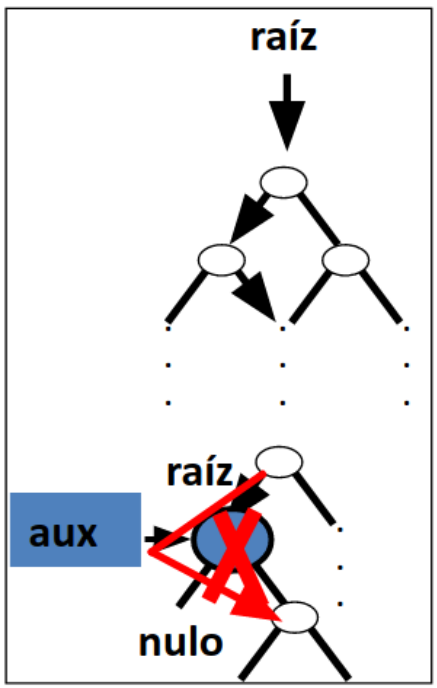
raíz ← aux↑.izq

Sólo hijo Izquierdo



raíz ← aux↑.izq

Sólo hijo Derecho



raíz ← aux↑.der

Distinción de casos: un descendiente o ninguno

Algorithm **eliminar**(x:tipoClave; **ref raíz:tipoÁrbol**)

Entrada: *x* clave del nodo a eliminar del árbol *raíz*
Salida: el árbol *raíz* con el nodo eliminado si existe

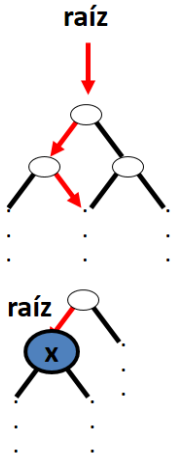
```
si raíz = NULO entonces
    /*no existe nodo con clave x: implementar según especificación*/
si no
    si x < raíz↑ .clave entonces
        eliminar(x,raíz↑.izq)
    si no
        si x > raíz↑.clave entonces
            eliminar(x,raíz↑.der)
        si no
            aux ← raíz
            si aux↑.der = NULO entonces
                raíz ← aux↑.izq              ->sólo hijo izquierdo o ningun descendiente
            si no
                si aux↑.izq = NULO entonces
                    raíz ← aux↑.der          ->sólo hijo derecho
                si no
                    ...                      ->dos hijos
                fin si
            fin si
            eliminarNodo(aux)
        fin si
    fin si
fin si
```

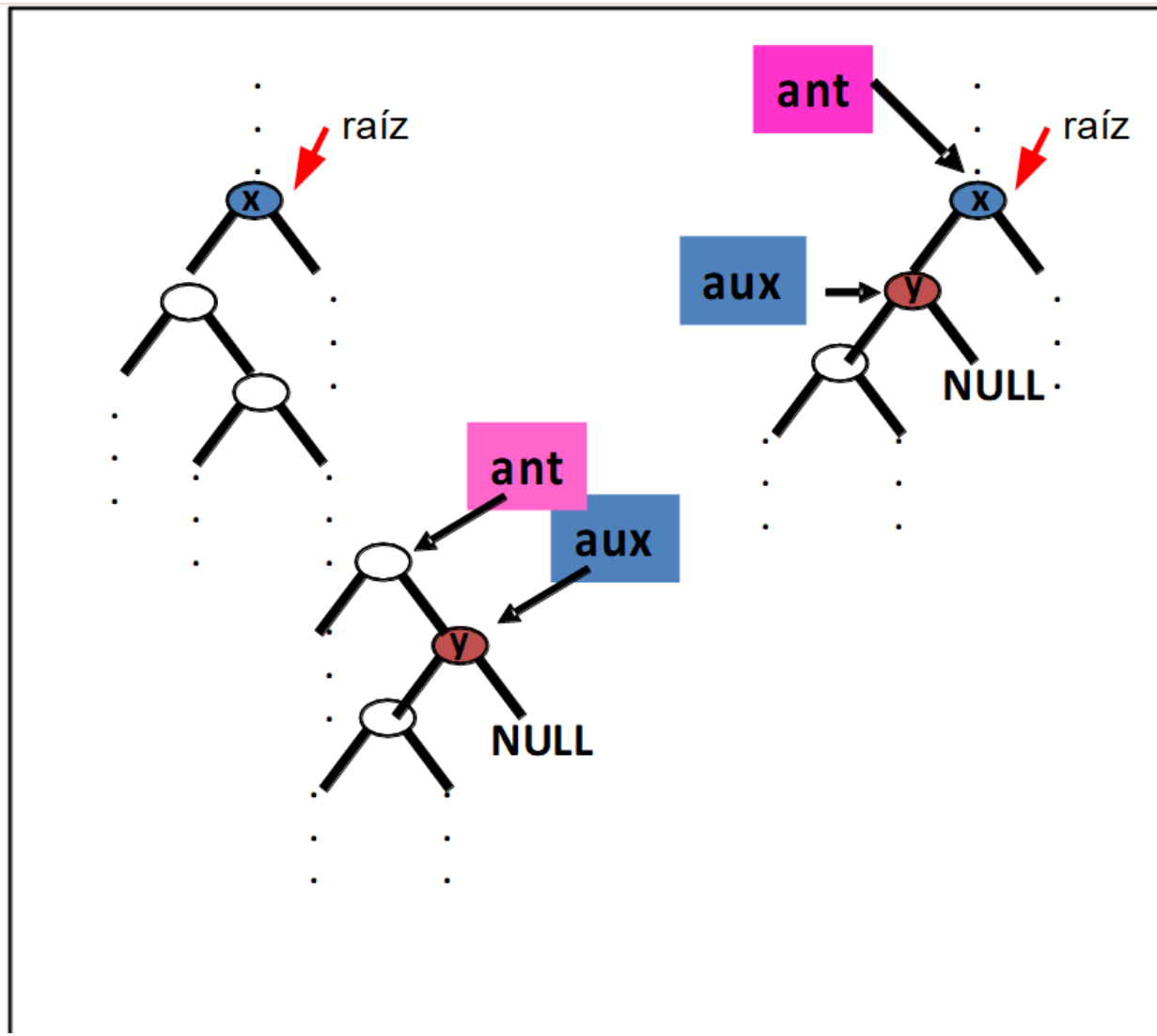
Distinción de casos: dos descendientes

buscar nodo más derecha de subárbol izquierdo

aux↑.izq≠ NULO

aux↑.der≠ NULO



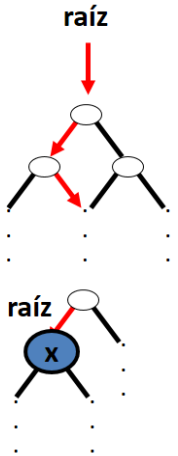


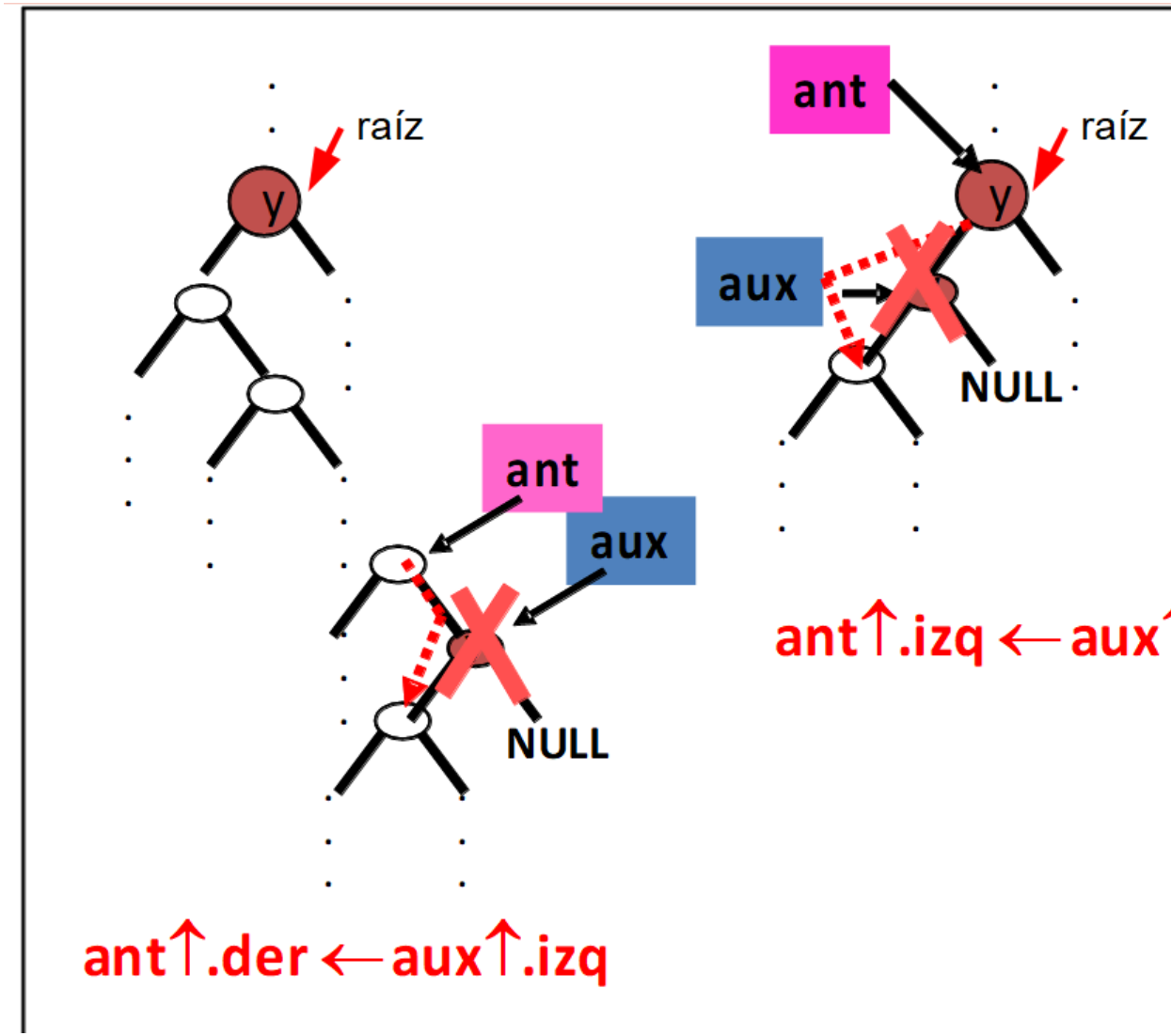
Distinción de casos: dos descendientes

sustituir por nodo más derecha de subárbol izquierdo

aux↑.izq≠ NULO

aux↑.der≠ NULO





Fragmento algortimo eliminación: dos descendientes

Algorithm eliminar(x:tipoClave, **referencia raíz:tipoÁrbol**)

```
aux ← raíz                                  ->raíz↑.clave= x ¡nodo a eliminar!
...
si no
ant ← aux                                                    ->dos hijos
aux ← aux↑.izq                  ->buscar nodo más derecha sub.izquierdo
mientras aux↑.der ≠ NULO hacer
   ant ← aux
   aux ← aux↑.der
fin mientras
raíz↑.clave ← aux↑.clave                          ->sustituir información
raíz↑.info ← aux↑.info
si ant = raíz entonces
   ant↑.izq ← aux↑.izq     ->enlazar subárboles de aux antes de eliminar
si no
   ant↑.der ← aux↑.izq
fin si
...
eliminarNodo(aux)
```

Algorithm eliminar(x:tipoClave; **ref raíz:tipoÁrbol**)

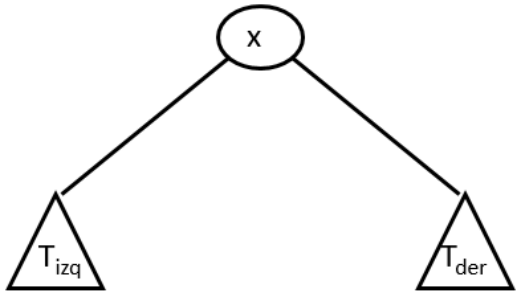
```
1: si raíz = NULO entonces
2:     /*no existe nodo con clave x: implementar según especificación*/
3: si no
4:     si x < raíz↑.clave entonces
5:         eliminar(x,raíz↑ .izq)
6:     si no
7:         si x > raíz↑.clave entonces
8:             eliminar(x,raíz↑.der)
9:         si no
10:            aux ← raíz                          ->raíz↑.clave= x ¡nodo a eliminar!
11:            si aux↑.der = NULO entonces
12:                raíz ← aux↑.izq                 ->sólo hijo izquierdo o ningún hijo
13:            si no
14:                si aux↑.izq = NULO entonces
15:                    raíz ← aux↑.der                      ->sólo hijo derecho
16:                si no
17:                    ant ← aux                            ->dos hijos
18:                    aux ← aux↑.izq          ->buscar nodo más derecha sub.izquierdo
19:                    mientras aux↑.der ≠ NULO hacer
20:                        ant ← aux
21:                        aux ← aux↑.der
22:                    fin mientras
23:                    raíz↑.clave ← aux↑.clave             ->sustituir información
24:                    raíz↑.info ← aux↑.info
25:                    si ant = raíz entonces
26:                        ant↑.izq ← aux↑.izq  ->enlazar subárboles de aux antes de eliminar
27:                    si no
28:                        ant↑.der ← aux↑.izq
29:                    fin si
30:                fin si
31:            fin si
32:            eliminarNodo(aux)
33:        fin si
34:    fin si
35: fin si
```

2.4 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 $0 \leq i < n$, 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

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

- Por tanto la profundidad esperada de cualquier nodo es O(logn)
- ¿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 de un nodo

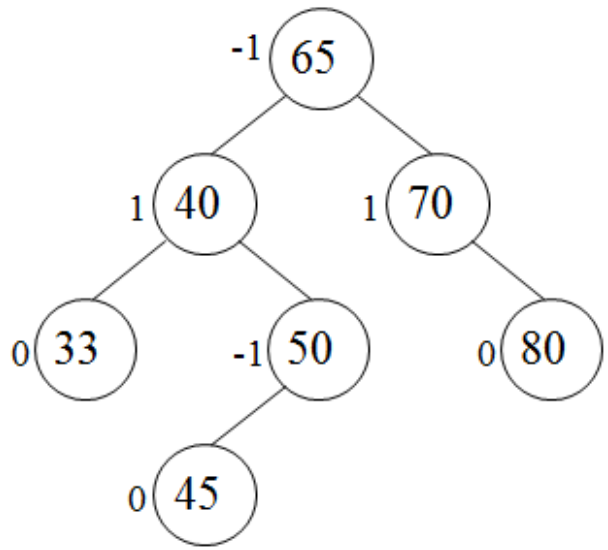
Diferencia entre las alturas de sus subárboles derecho e izquierdo

$$fe = h_d - h_i$$

- Valores posibles 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**

Algorithm declaraciones básicas

```
tipos
tipoNodo = registro
  clave: tipoClave
  información: tipoInformación
  fe: -1 .. 1
  izq, der: ↑tipoNodo
fin registro
tipoÁrbol: ↑tipoNodo
punteroNodo: ↑tipoNodo
```

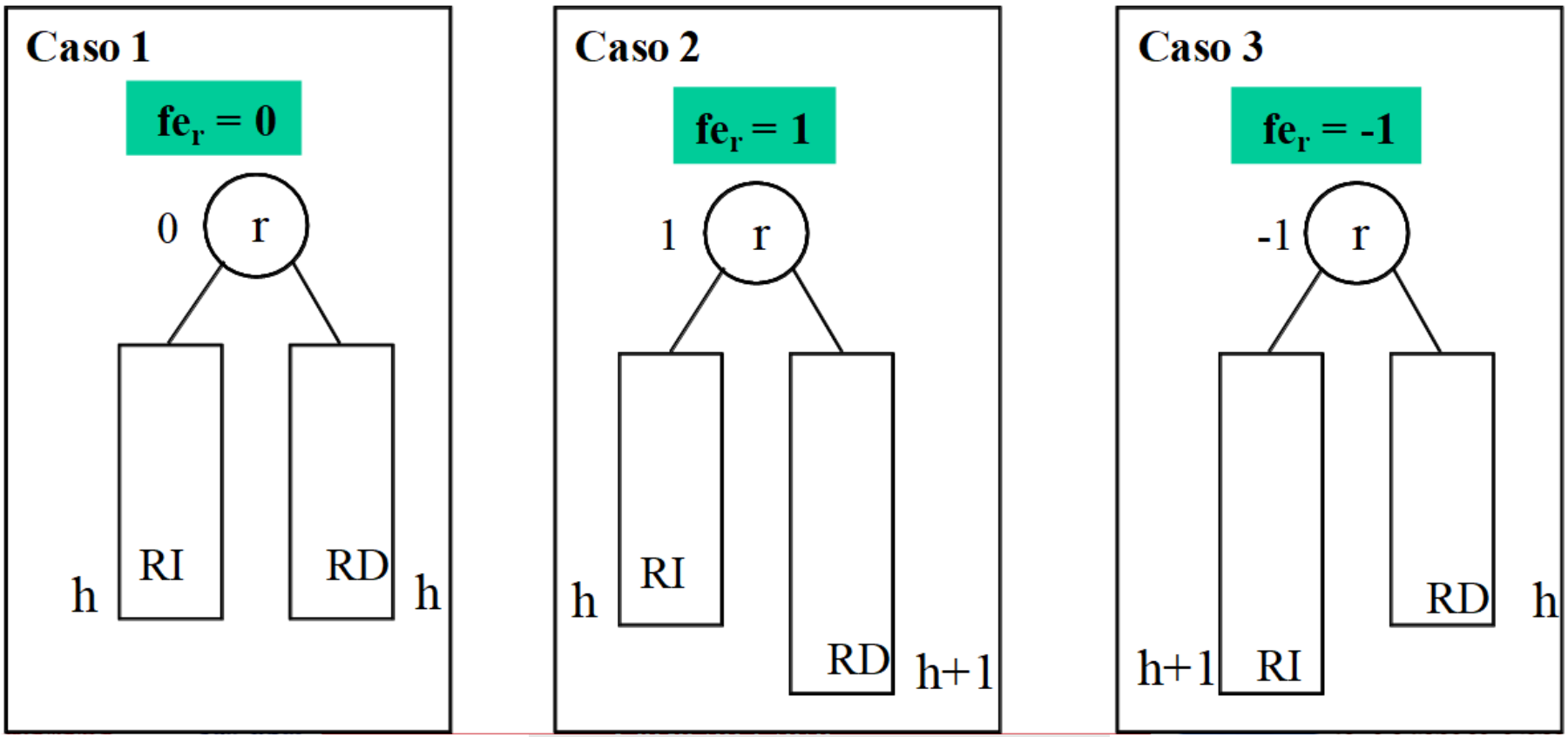


4 Inserción y equilibrio del árbol

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



Proceso de Inserción

1. 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)
2. **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 **subárbol** en aquellos casos en que sea necesario, evitando que el factor de equilibrio tome valores 2 ó -2

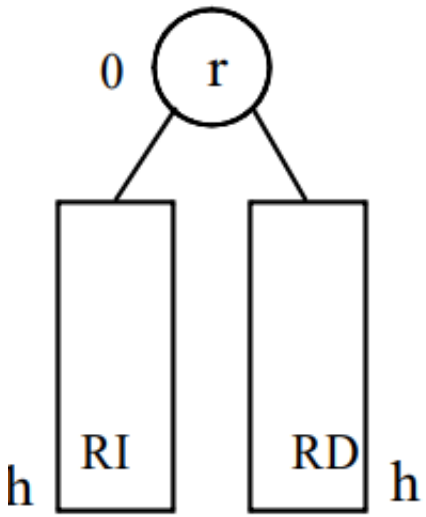
- Puede implementarse como un algoritmo recursivo que tiene como parámetros:
  - puntero al nuevo nodo
  - puntero que inicialmente señala a la raíz del árbol, que permitirá seguir el camino de búsqueda
  - un parámetro de tipo lógico que indicará si la altura del subárbol ha cambiado (aumentado) como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

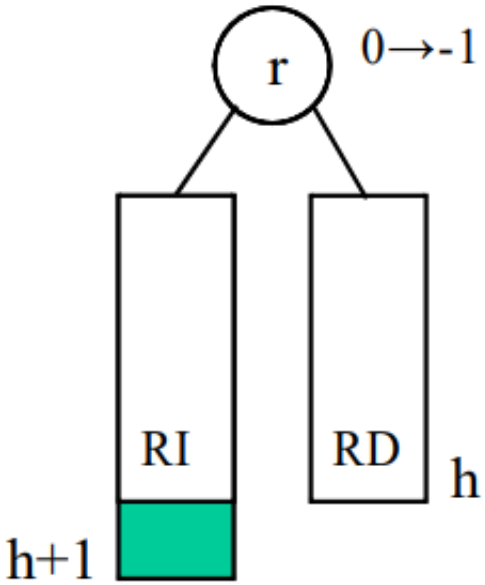
Algoritmo de inserción

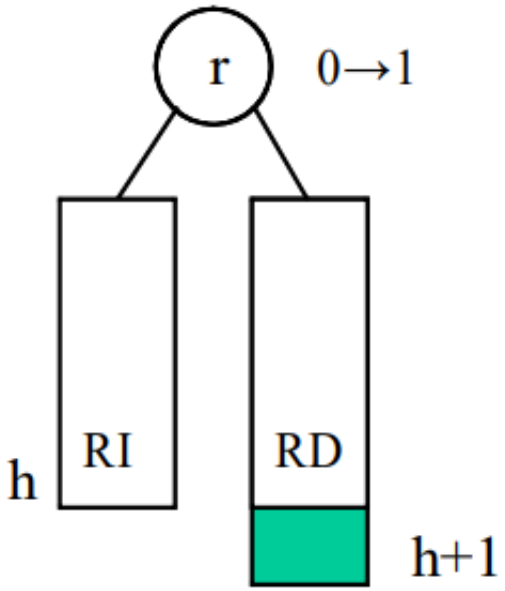
Algorithm insertar(nuevo:tipoNodo; **ref cambiaH:lógico**; **ref nodo:tipoÁrbol**)

```
 1: si nodo = NULO entonces
 2:     nodo ← nuevo
 3:     cambiaH ← VERDADERO
 4: si no
 5:     si nuevo↑.clave = nodo↑.clave entonces
 6:         /* clave dupliacada implementar según especificación */
 7:     si no
 8:         si nuevo↑.clave < nodo↑.clave entonces
 9:             insertar(nuevo,cambiaH, nodo↑.izq)
10:             si cambiaH entonces
11:                 equilibrarIzq(nodo,cambiaH)
12:             fin si
13:         si no
14:             insertar(nuevo,cambiaH,nodo↑.der)
15:             si cambiaH entonces
16:                 equilibrarDer(nodo,cambiaH)
17:             fin si
18:         fin si
19:     fin si
20: fin si
```

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

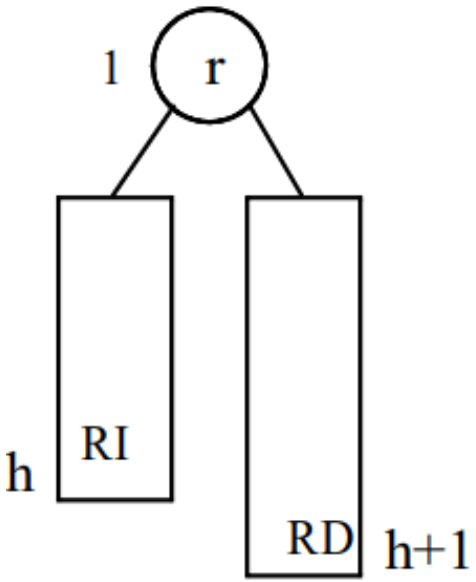


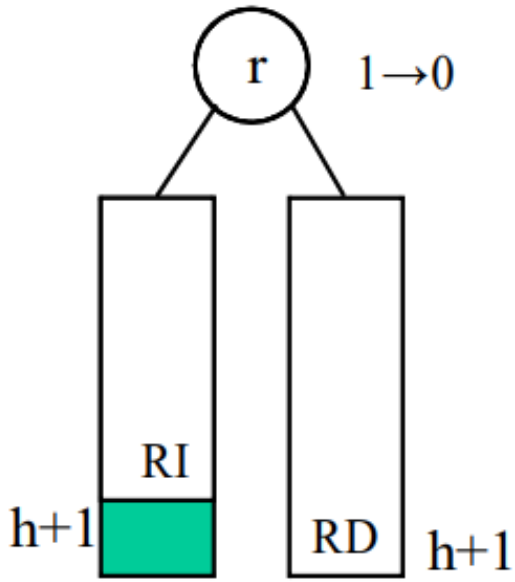


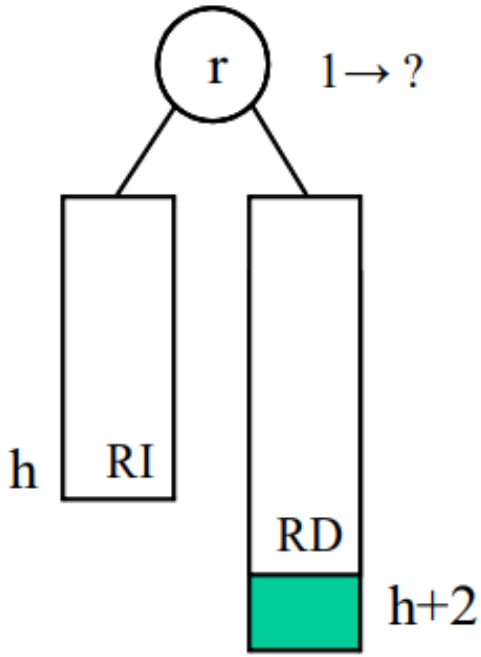


Antes Inserción		Después Inserción por rama Izquierda			Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero

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

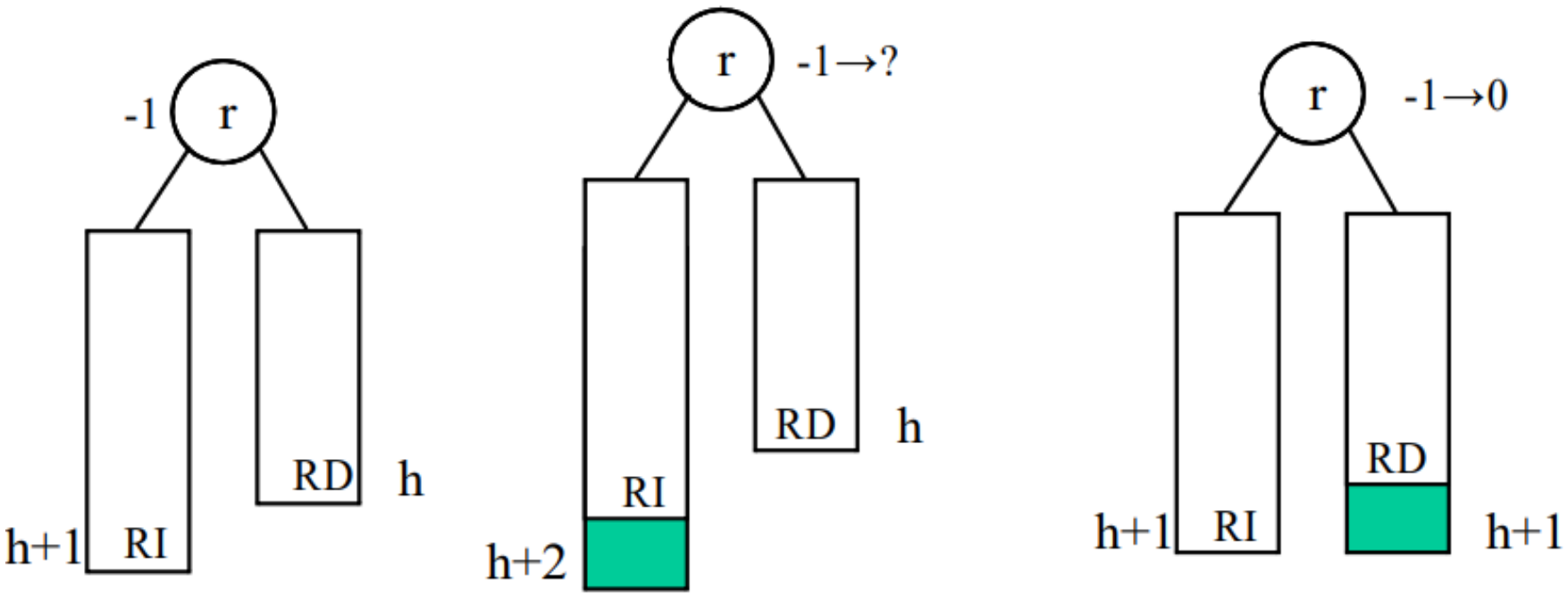






Antes Inserción		Después Inserción por rama Izquierda			Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	**Reestruct. Sub.Derecho**		

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

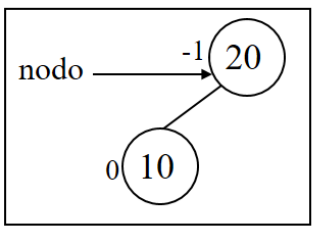


Antes Inserción		Después Inserción por rama Izquierda			Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	**Reestruct.Sub.Derecho**		
-1	h+2	**Reestruct.Sub.Izquierdo**			0	h+2	falso

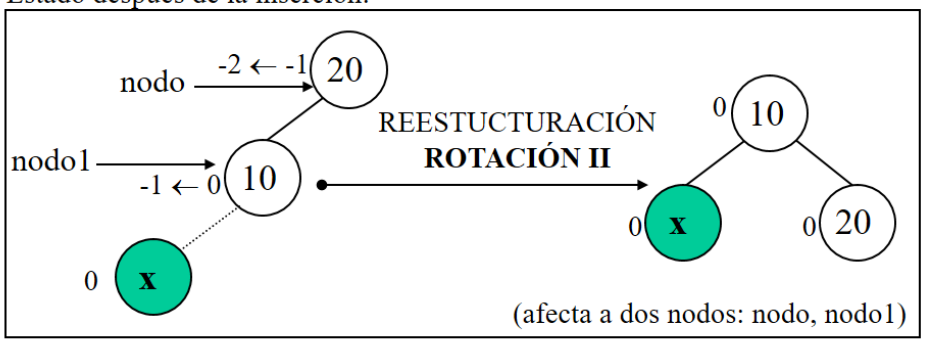
4.1 Equilibrar rama izquierda

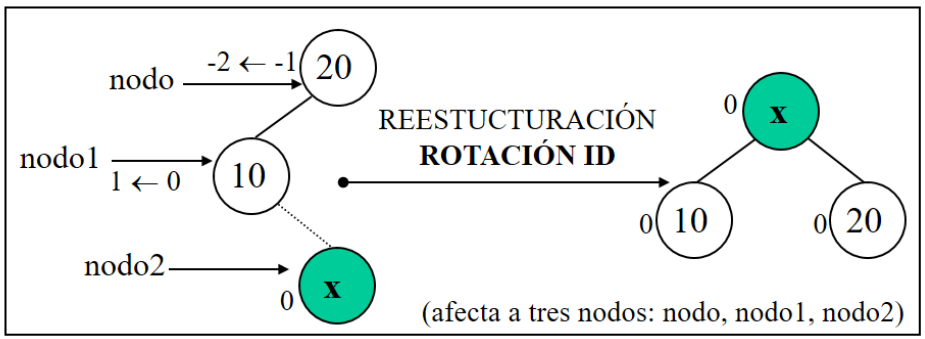
Ejemplo reestructuración subárbol Izquierdo (nodo↑.fe = -1)

Estado antes de la inserción:



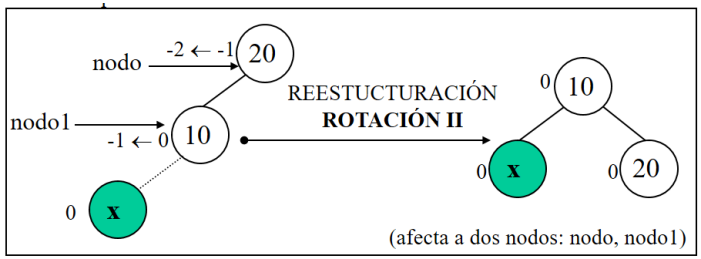
Estado después de la inserción:

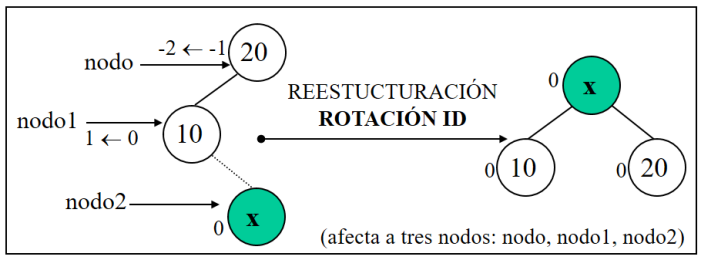




Antes Inserción		Después Inserción por rama Izquierda			Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	**Reestruct.Sub.Derecho**		
-1	h+2	**¿II o ID? Depende de FE nodo ↑.izq**			0	h+2	falso

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	**nodo1↑.fe = -1 Rotación II**		falso
		nodo1↑.fe = 1 Rotación ID		falso





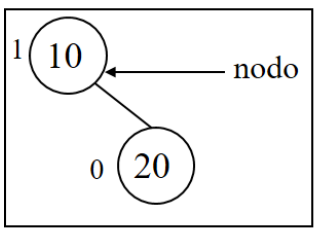
Algorithm equilibarIzq(**ref** nodo: punteroNodo; **ref** cambiaH:lógico)

```
 1: casos nodo↑.fe en
 2:     0: nodo↑.fe ← -1
 3:     1: nodo↑.fe ← 0
 4:        cambiaH ← FALSO
 5:    -1: nodo1 ← nodo↑.izq
 6:        si nodo1↑.fe = -1 entonces
 7:          rotaciónII(nodo,nodo1)
 8:        si no
 9:          nodo2 ← nodo1↑.der
10:          rotaciónID(nodo,nodo1,nodo2)
11:        fin si
12:        cambiaH ← FALSO
13: fin casos
```

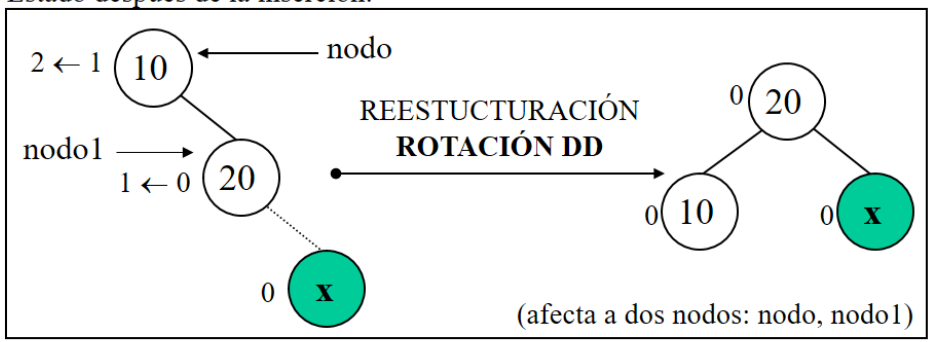
4.2 Equilibrar rama derecha

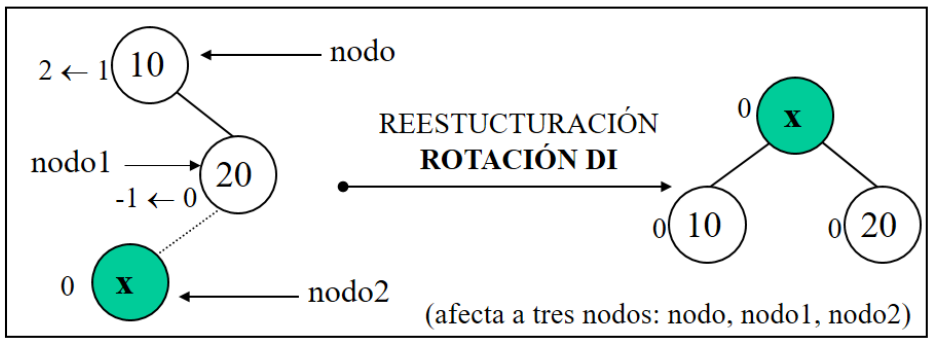
Reestructuración subárbol derecho (nodo↑.fe = 1)

Estado antes de la inserción:



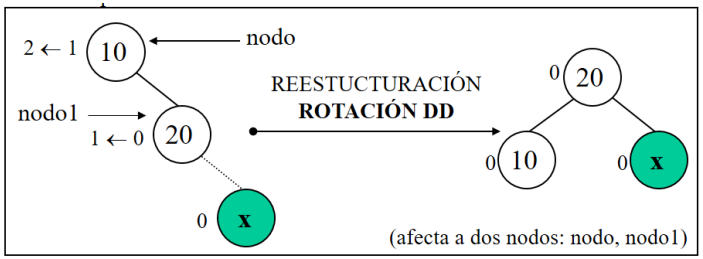
Estado después de la inserción:

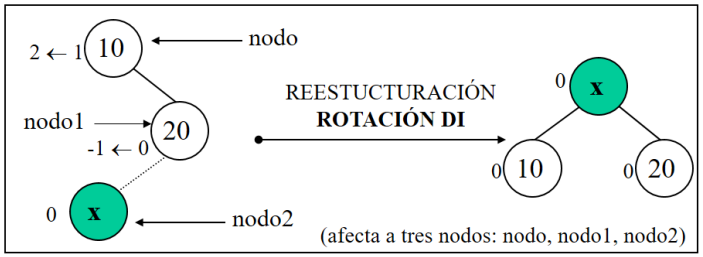




Antes Inserción		Después Inserción por rama Izquierda			Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	-1	h+2	verdadero	1	h+2	verdadero
1	h+2	0	h+2	falso	**¿DD o DI? Depende de FE nodo ↑.der**		
-1	h+2	**¿II o ID? Depende de FE nodo ↑.izq**			0	h+2	falso

Antes Inserción		Después Inserción por rama Derecha		
nodo↑.fe	**altura**	**nodo↑.fe**	**altura**	**cambiaH**
0	h+1	1	h+2	verdadero
1	h+2	**nodo1↑.fe = 1 Rotación DD**		falso
		nodo1↑.fe = -1 Rotación DI		falso
-1	h+2	0	h+2	falso



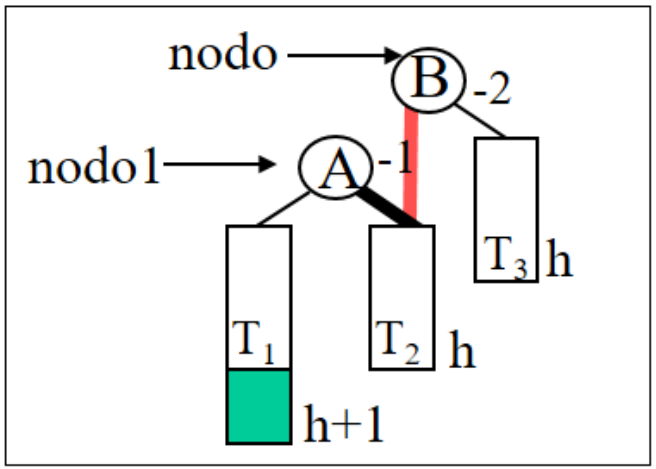


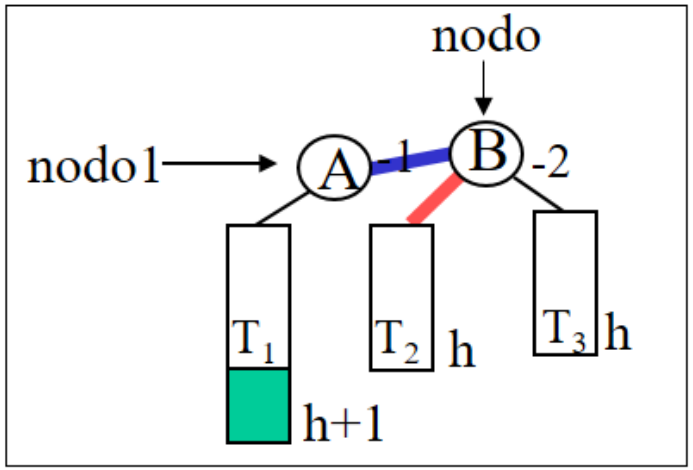
Algorithm equilibarDer(**ref** nodo: punteroNodo; **ref** cambiaH:lógico)

```
 1: casos nodo↑.fe en
 2:     0: nodo↑.fe ← 1
 3:     1: nodo1 ← nodo↑.der
 4:        si nodo1↑.fe = 1 entonces
 5:          rotaciónDD(nodo,nodo1)
 6:        si no
 7:          nodo2 ← nodo1↑.izq
 8:          rotaciónDI(nodo,nodo1,nodo2)
 9:        fin si
10:        cambiaH ← FALSO
11:    -1: nodo↑.fe ← 0
12:        cambiaH ← FALSO
13: fin casos
```

4.3 Rotación Izquierda Izquierda

Movimiento de los nodos en la rotación y cambio factor equilibrio





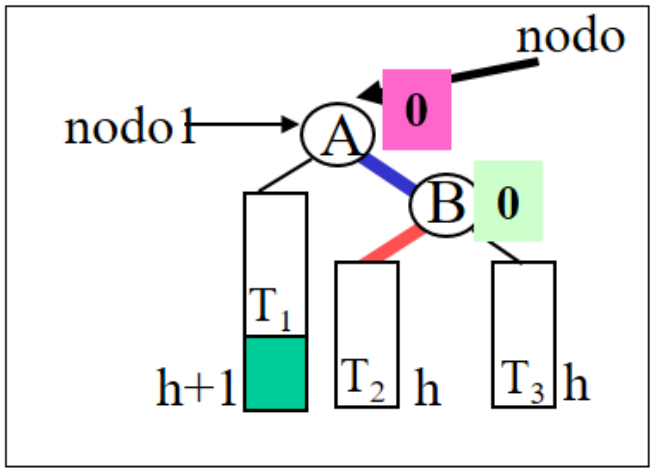
nodo↑.izq ←nodo1↑.der

nodo1↑.der ←nodo

nodo↑.fe ← 0

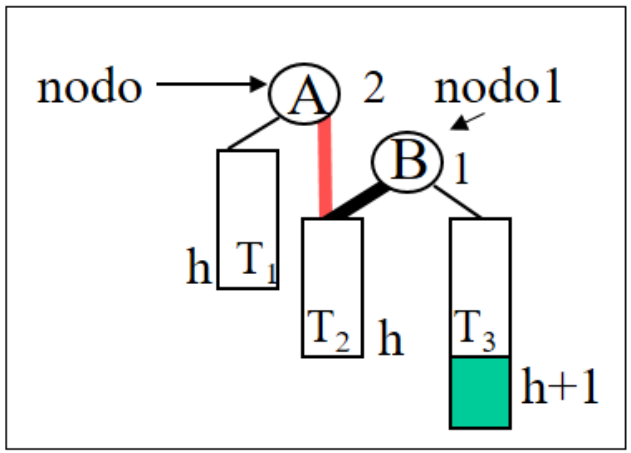
nodo1↑.fe ← 0

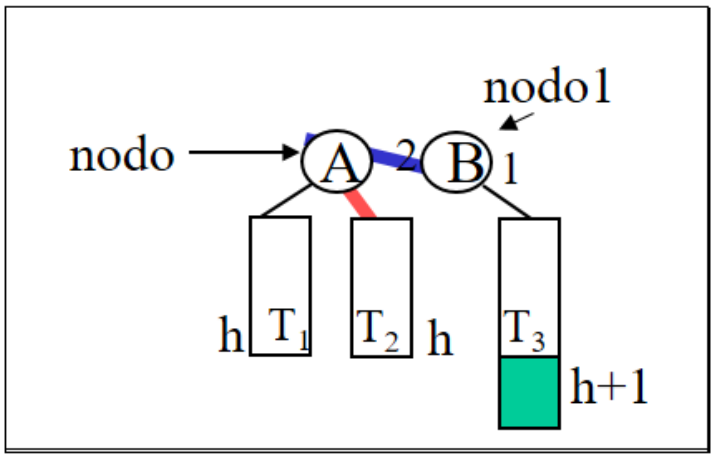
nodo ← nodo1



4.4 Rotación Derecha Derecha

Movimiento de los nodos en la rotación y cambio factor equilibrio





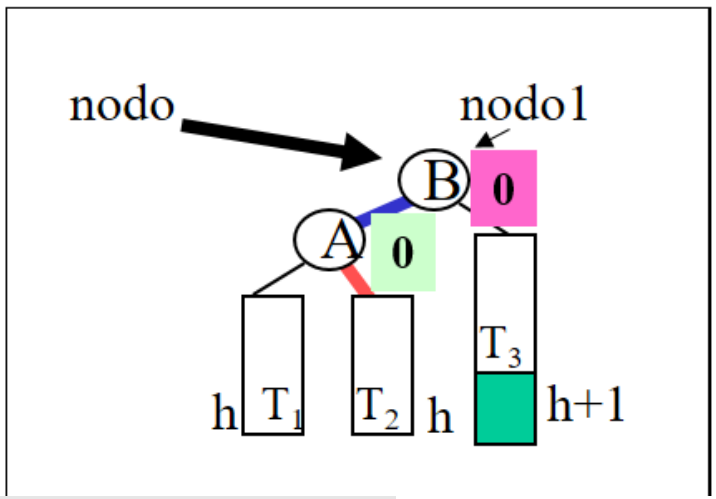
nodo↑.der ← nodo1↑.izq

nodo1↑.izq ← nodo

nodo↑.fe ← 0

nodo1↑.fe ←0

nodo ← nodo1



Algoritmos de rotaciones simples: II y DD

Sólo sirven para proceso de inserción

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

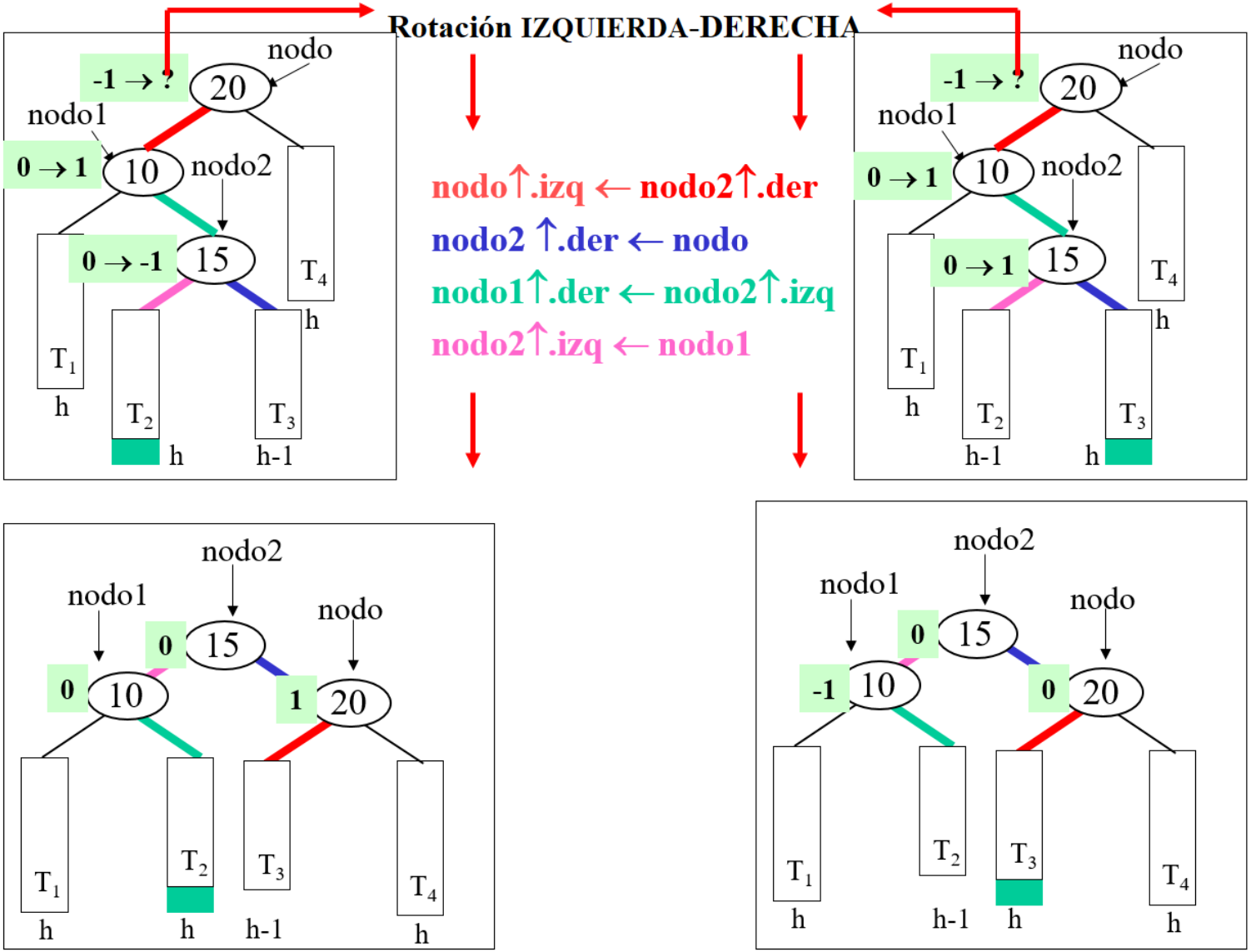
```
1: nodo↑.izq ← nodo1↑.der
2: nodo1↑.der ← nodo                    ->Movimiento nodos en rotación
3: nodo↑.fe ← 0
4: nodo1↑.fe ← 0                        ->Cambio en los factores de equilibrio
5: nodo ← nodo1                         ->Nueva raíz del subárbol
```

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

```
1: nodo↑.der ← nodo1↑.izq
2: nodo1↑.izq ← nodo                    ->Movimiento nodos en rotación
3: nodo↑.fe ← 0
4: nodo1↑.fe ← 0                        ->Cambio en los factores de equilibrio
5: nodo ← nodo1                         ->Nueva raíz del subárbol
```

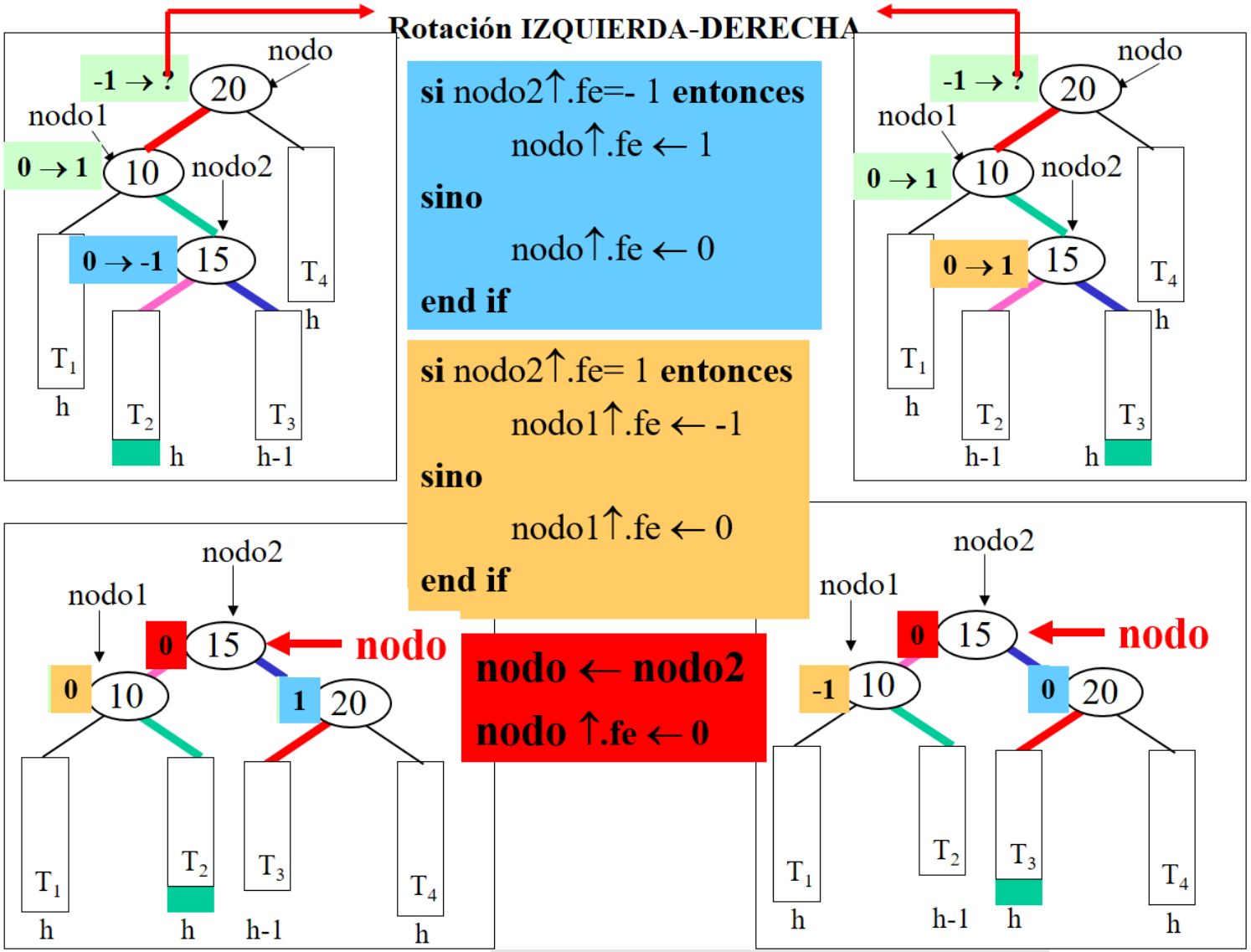
4.5 Rotación Izquierda Derecha

1. Movimiento de los nodos en la rotación



Rotación Izquierda Derecha

2.Cambio de factores de equilibrio



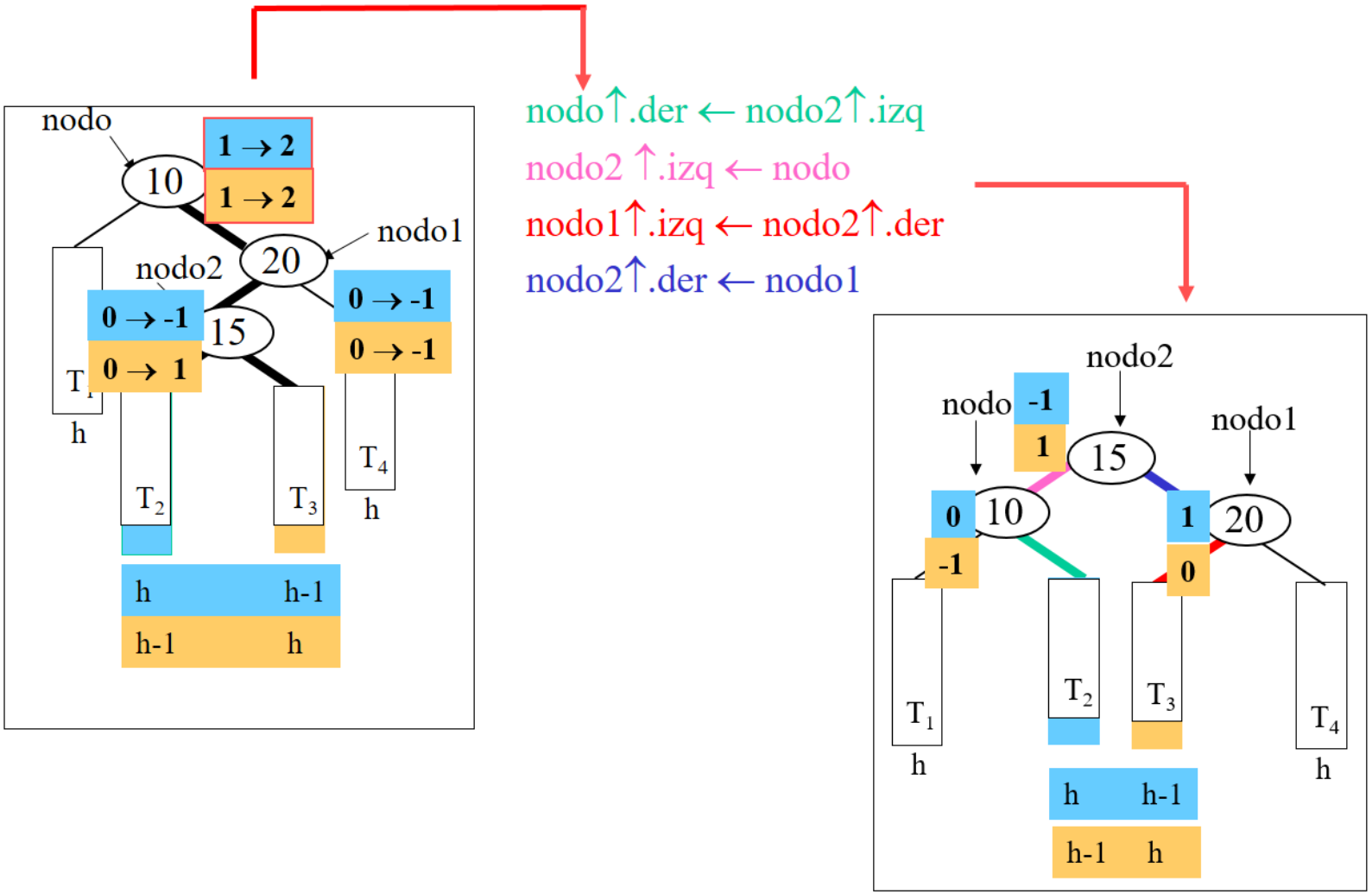
Algoritmo de rotación Izquierda Derecha

Algorithm rotaciónID(**ref** nodo: punteroNodo, nodo1,nodo2: punteroNodo)

```
    1: no es nodo1 errata
 1: nodo1↑.izq ← nodo2↑.der
 2: nodo2↑.der ← nodo
 3: nodo1↑.der ← nodo2↑.izq
 4: nodo2↑.izq ← nodo1                          ->Movimiento nodos en rotación
 5: si nodo2↑.fe = -1 entonces
 6:     nodo↑.fe ← 1
 7: si no
 8:     nodo↑.fe ← 0
 9: fin si
10: si nodo2↑.fe = 1 entonces
11:     nodo1↑.fe ← -1
12: si no
13:     nodo1↑.fe ← 0
14: fin si                                      ->Cambio en los factores de equilibrio
15: nodo ← nodo2                                ->Nueva raíz del subárbol
16: nodo ↑.fe ← 0
```

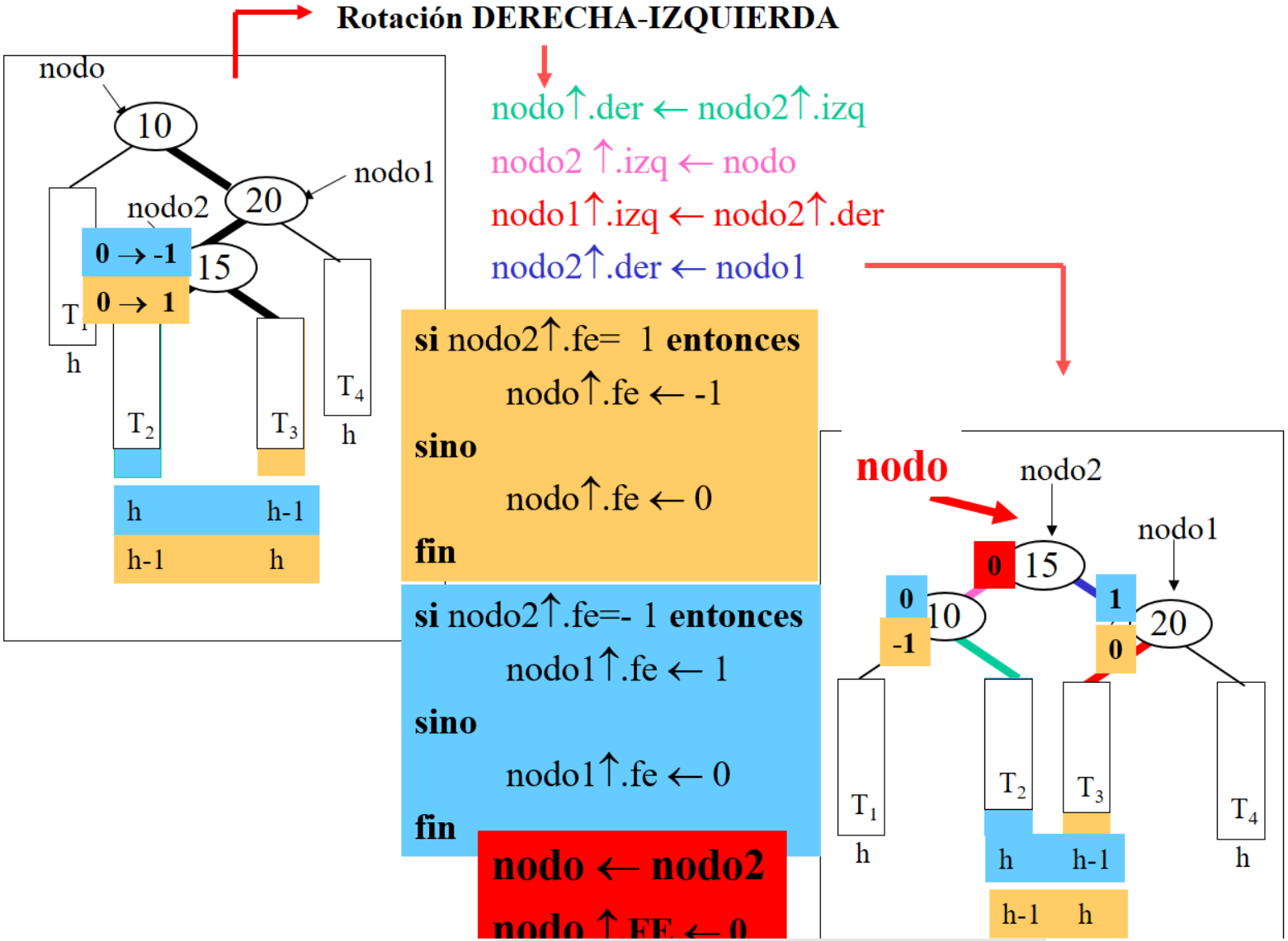
4.6 Rotación Derecha Izquierda

1. Movimiento de los nodos en la rotación



Rotación Derecha Izquierda

2.Cambio de factores de equilibrio



Algoritmo de rotación Derecha Izquierda

Algorithm rotaciónDI(**ref** nodo: punteroNodo, nodo1,nodo2: punteroNodo)

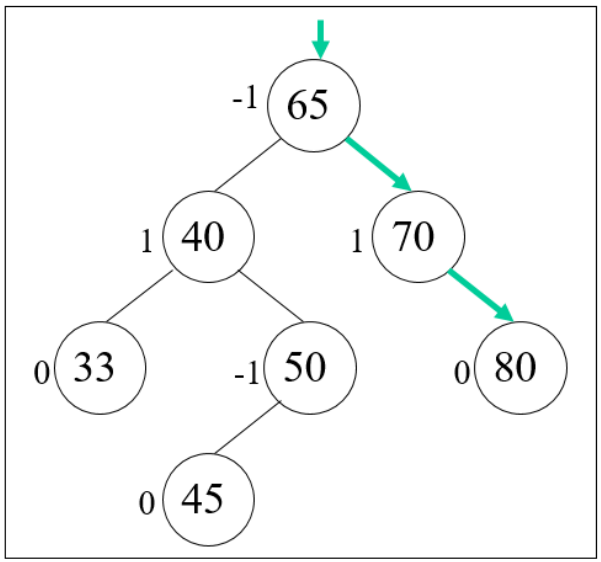
```
 1: nodo↑.der ← nodo2↑.izq
 2: nodo2↑.izq ← nodo
 3: nodo1↑.izq ← nodo2↑.der
 4: nodo2↑.der ← nodo1                    ->Movimiento nodos en rotación
 5: si nodo2↑.fe = 1 entonces
 6:     nodo↑.fe ← -1
 7: si no
 8:     nodo↑.fe ← 0
 9: fin si
10: si nodo2↑.fe = -1 entonces
11:     nodo1↑.fe ← 1
12: si no
13:     nodo1↑.fe ← 0
14: fin si                                ->Cambio en los factores de equilibrio
15: nodo← nodo2                           ->Nueva raíz del subárbol
16: nodo↑.fe ← 0
```

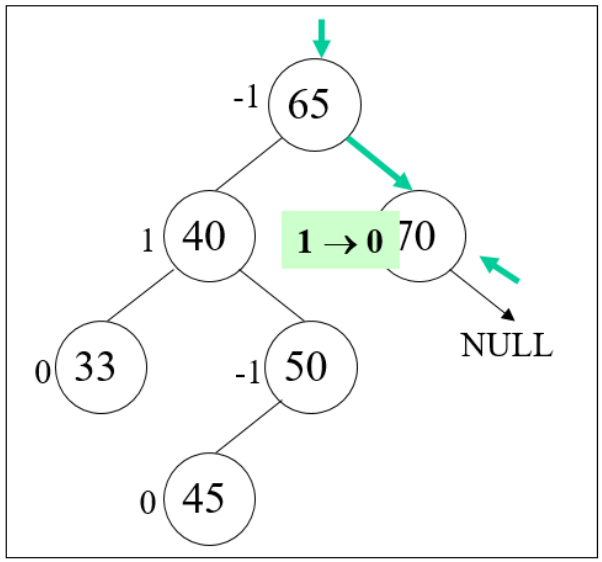
5 Eliminación y equilibrio del árbol

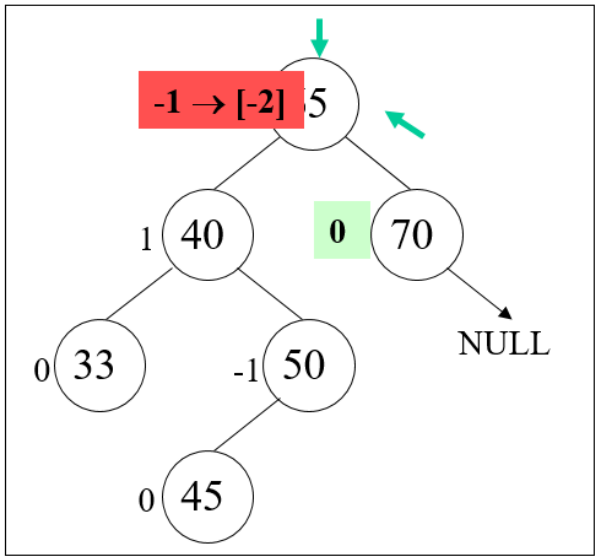
- El proceso sigue la misma lógica que el algoritmo de eliminación en a.b.b. añadiéndole las operaciones de reestructuración utilizadas en el algoritmo de inserción en árboles balanceados (rotaciones II, DD, ID y DI) $\Rightarrow$ Resulta bastante más complejo
- En el proceso de inserción, los subárboles después de una rotación tienen la misma altura que antes de la inserción y la rotación. Por tanto, después de una rotación no se producirán más rotaciones.
- En eliminación, sin embargo, una vez efectuada una rotación el algoritmo puede continuar. Es decir, se puede producir más de una rotación en el retroceso realizado por el camino de búsqueda, pudiendo llegar hasta la raíz del árbol
- La rama que hay que equilibrar es la contraria a la rama por la que se hace la eliminación

Eliminación por Derecha ⇒ Equilibrar por Izquierda

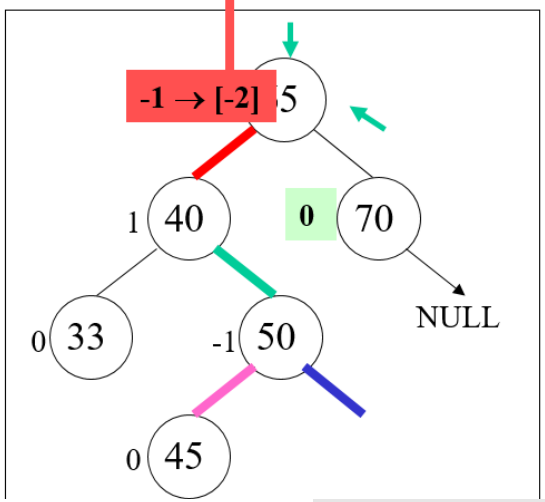
Ejemplo: eliminación del nodo con clave 80

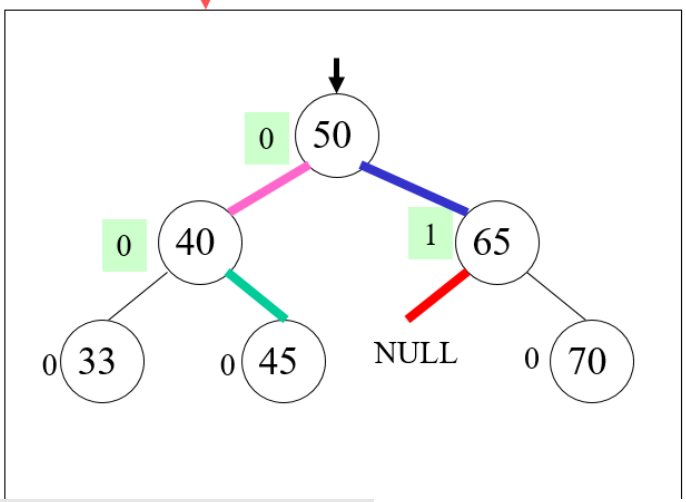






Rotación_ID





Proceso de Eliminación

1. 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)
2. **Regresar** por el camino de búsqueda recalculando el factor de equilibrio de los nodos visitados. 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:
  - 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

Algorithm **eliminar**(x:tipoClave; **ref** cambiaH:lógico; **ref** nodo:tipoÁrbol)

```
1: si nodo = NULO entonces
2:     /*no existe nodo con clave x: implementar según especificación*/
3: si no
4:     si x < nodo↑.clave entonces
5:         eliminar(x,cambiaH,nodo↑.izq)
6:         si cambiaH entonces
7:             equilibarDer(nodo,cambiaH)
8:         fin si
9:     si no
10:        si x > nodo↑.clave entonces
11:            eliminar(x,cambiaH,raíz↑.der)
12:            si cambiaH entonces
13:                equilibarIzq(nodo,cambiaH)
14:            fin si
15:        si no
16:            aux ← nodo                          ->nodo↑.clave = x ¡nodo a eliminar!
17:            si aux↑.der = NULO entonces
18:                nodo← aux↑.izq                  ->sólo hijo izquierdo o ningún hijo
19:                eliminarNodo(aux)
20:                cambiaH← VERDADERO
21:            si no
22:                si aux↑.izq = NULO entonces
23:                    nodo← aux↑.der              ->sólo hijo derecho
24:                    eliminarNodo(aux)
25:                    cambiaH← VERDADERO
26:                si no
27:                    borrar(nodo, aux↑.izq, aux, cambiaH)   ->dos hijos
28:                    si cambiaH entonces
29:                        equilibarDer(nodo,cambiaH)
30:                    fin si
31:                fin si
32:            fin si
33:        fin si
34:    fin si
35: 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 de tipo 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

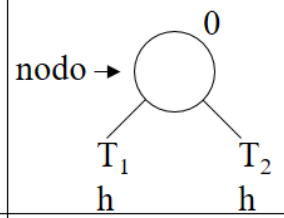
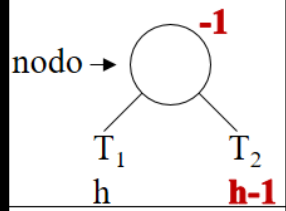
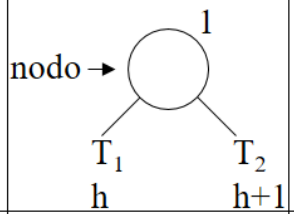
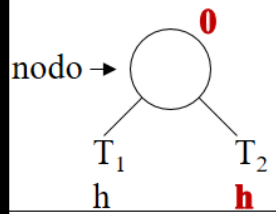
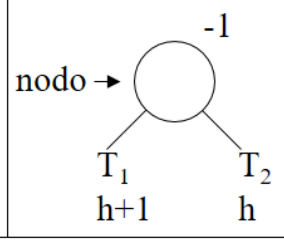
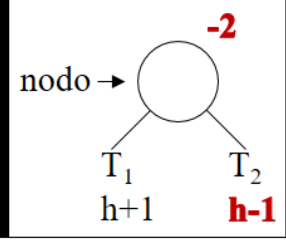
Algorithm borrar(**ref** nodo: punteroNodo; aux, ant: punteroNodo;
ref cambiaH:lógico)

```
 1: si aux↑.der ≠ NULO entonces
 2:     borrar(nodo, aux↑.der, aux, cambiaH)
 3:     si cambiaH entonces
 4:         equilibrarIzq(nodo,cambiaH)
 5:     fin si
 6: si no
 7:     nodo↑.clave ← aux↑.clave
 8:     nodo↑.info ← aux↑.info
 9:     si ant↑.izq = aux entonces
10:         ant↑.izq ← aux↑.izq
11:     si no
12:         ant↑.der ← aux↑.izq
13:     fin si
14:     liminarNodo(aux)
15:     cambiaH ← VERDADERO
16: fin si
```

5.1 Equilibrar Subárbol Izquierdo

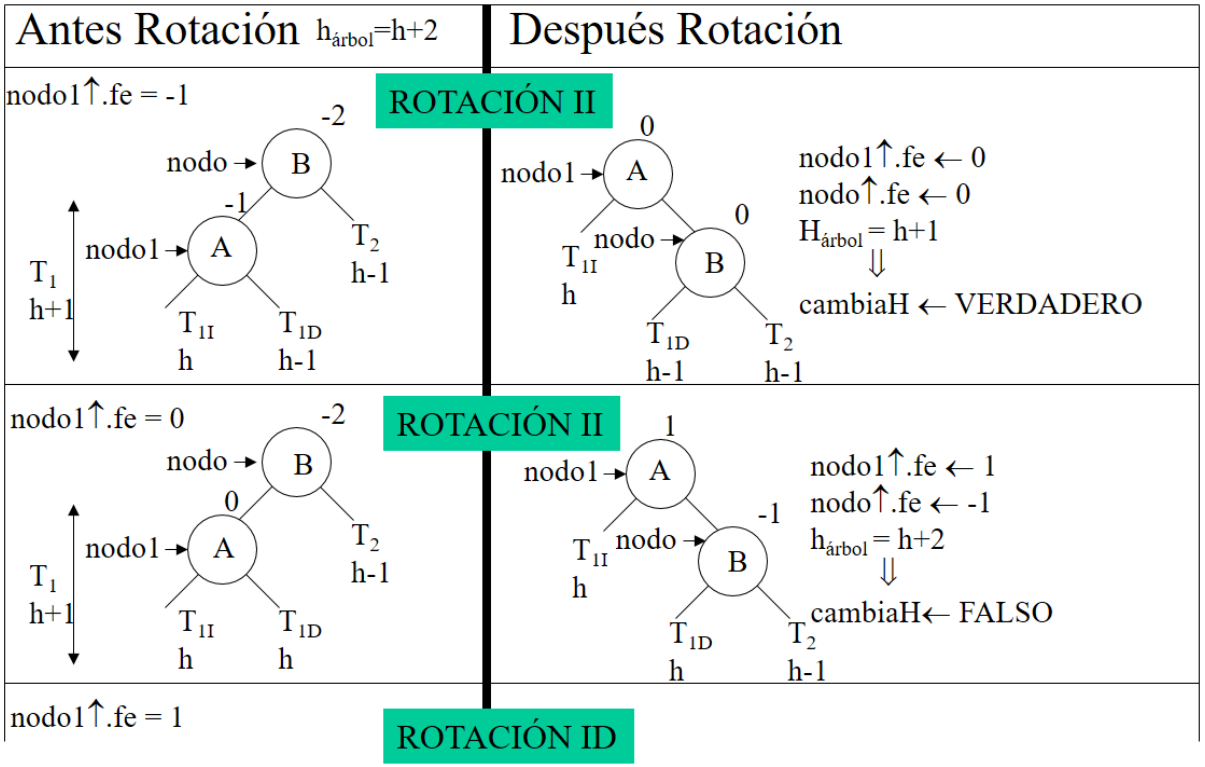
- Este proceso se efectúa al volver por el camino de búsqueda, después de haber **eliminado** un nodo en el **subárbol derecho**
- En esta vuelta se van recalculando los factores de equilibrio de los nodos encontrados en el camino
- Antes de que el factor de equilibrio de un nodo (raíz del subárbol) llegue a tomar el valor -2 se debe reestructurar el **subárbol izquierdo** que será de mayor altura
- Este proceso se implementa mediante 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 $H_{RI}=H_{RD}$		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 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				**Reestructuración Subárbol Izquierdo**		

Casos al equilibrar subárbol IZQUIERDO (continuación I)



Antes Eliminación		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	nodo1↑.fe		nodo↑.fe	nodo1↑.fe		
Reest.Sub.Izquierdo		-1	RotaciónII(*)	0	0	h+1	verdadero
nodo1 ←nodo↑.izq		0	RotaciónII(*)	-1	1	h+1	falso
¿II o ID?		1	RotaciónID			h+1	verdadero

(*)Modificada para elimiación

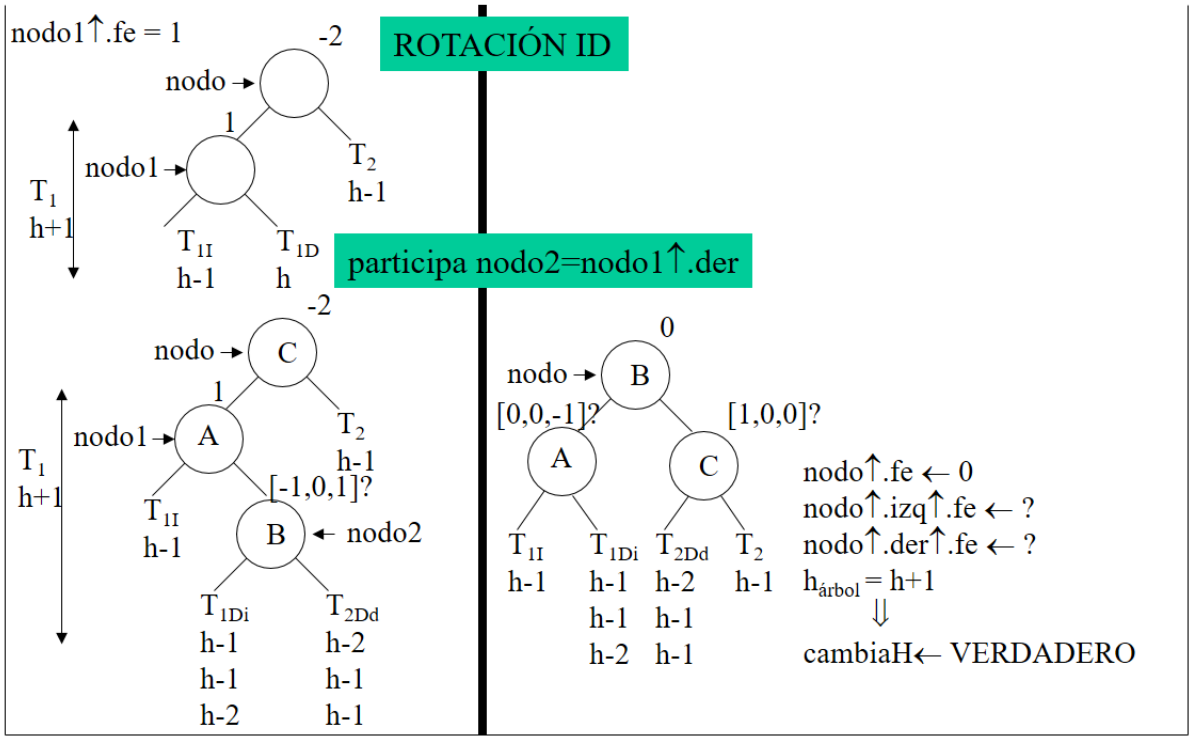
Algoritmo rotación Izquierda Izquierda modificado para eliminación

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

```
 1: nodo↑.izq ← nodo1↑.der
 2: nodo1↑.der ← nodo                    ->Movimiento nodos en rotación
 3: si nodo1↑.fe = -1 entonces
 4:     nodo↑.fe ← 0
 5:     nodo1↑.fe ← 0                    ->Siempre si la rotación es por una inserción
 6: si no
 7:     nodo↑.fe ← -1
 8:     nodo1↑.fe ← 1
 9: fin si
10: nodo ← nodo1                         ->Nueva raíz del subárbol
```

Antes Eliminación		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	nodo1↑.fe		nodo↑.fe	nodo1↑.fe		
Reest.Sub.Izquierdo		-1	RotaciónII(*)	0	0	h+1	verdadero
nodo1 ←nodo↑.izq		0	RotaciónII(*)	-1	1	h+1	falso
¿II o ID?		1	RotaciónID			h+1	verdadero

Casos al equilibrar subárbol IZQUIERDO (continuación II)



Equilibra Rama Izquierda (elimanción por derecha)

Antes Eliminación		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	nodo1↑.fe			nodo↑.fe	nodo1↑.fe	
Reest.Sub.Izquierdo		-1	RotaciónII(*)		0	0	h+1	verdadero
nodo1 ←nodo↑.izq		0	RotaciónII(*)		-1	1	h+1	falso
¿II o ID?		1	RotaciónID				h+1	verdadero

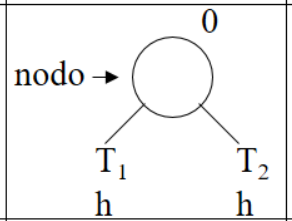
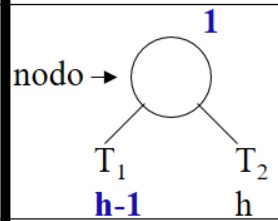
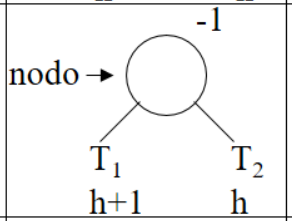
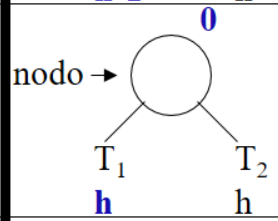
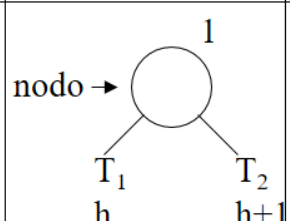
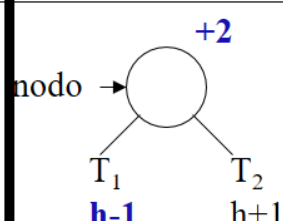
Algorithm equilibarIzq(**ref nodo: punteroNodo**; **ref cambiaH:lógico**)

```
 1: casos nodo↑.fe en
 2:     1: nodo↑.fe ← 0
 3:     0: nodo↑.fe ← -1
 4:        cambiaH ← FALSO
 5:    -1: nodo1 ← nodo↑.izq
 6:        si nodo1↑.fe ≤ 0  entonces
 7:          rotaciónII(nodo,nodo1)
 8:          si nodo1↑.fe = 0 entonces
 9:            cambiaH ← FALSO
10:          si no
11:        si no
12:          nodo2← nodo1↑.der
13:          rotaciónID(nodo,nodo1,nodo2)
14:        fin si
15:        cambiaH ← FALSO
16: fin casos
```

5.2 Equilibrar Subárbol Derecho

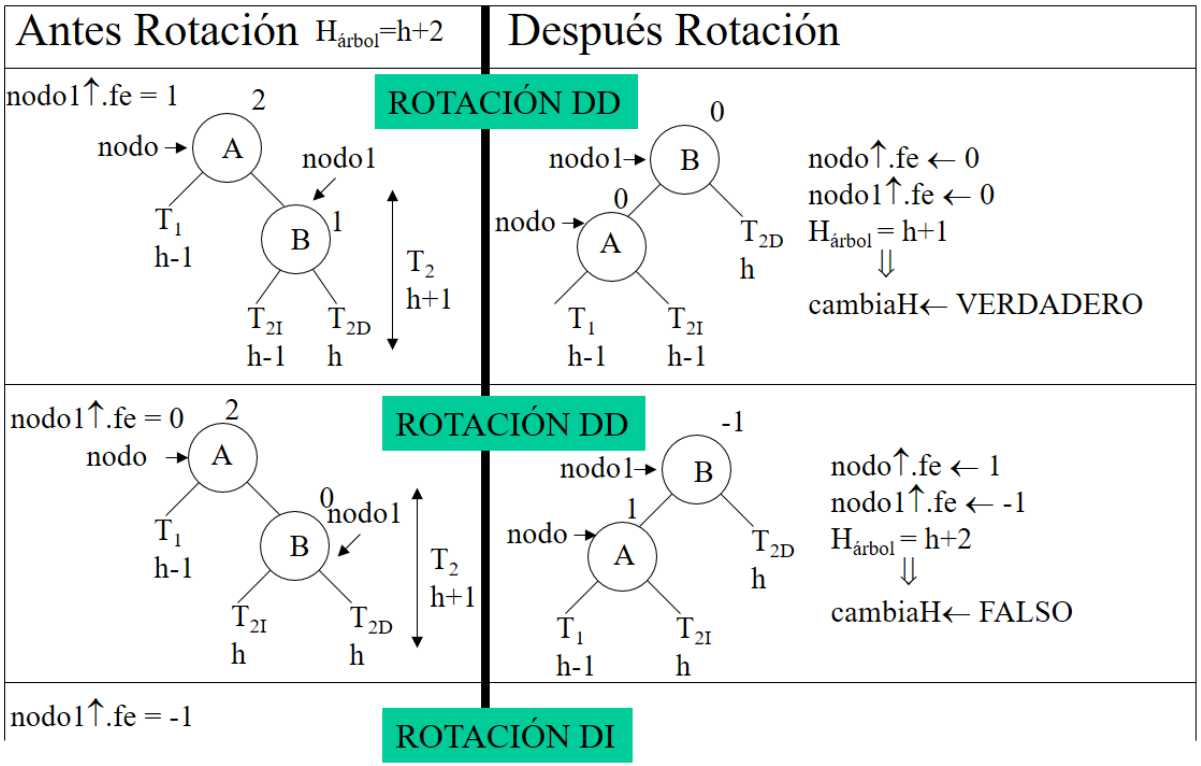
- Este proceso se efectúa al volver por el camino de búsqueda, después de haber **eliminado** un nodo en el **subárbol izquierdo**
- En esta vuelta se van recalculando los factores de equilibrio de los nodos encontrados en el camino
- Antes de que el factor de equilibrio de un nodo (raíz del subárbol) llegue a tomar el valor $+2$ se debe reestructurar el **subárbol derecho** que será de mayor altura
- Este proceso se implementa mediante 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 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	**Reestructuración Subárbol Derecho**			0	h+1	verdadero
-1	h+2	0	h+1	verdadero	**Reestructuración Subárbol Izquierdo**		

Casos al equilibrar subárbol DERECHO (continuación I)



Antes Eliminación		Después Eliminación por rama Izquierda					
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	nodo1↑.fe		nodo↑.fe	nodo1↑.fe		
Reest.Sub.Derecho		1	RotaciónDD(*)	0	0	h+1	verdadero
nodo1 ←nodo↑.der		0	RotaciónDD(*)	-1	1	h+1	falso
¿DD o DI?		1	RotaciónDI			h+1	verdadero

(*)Modificada para elimiación

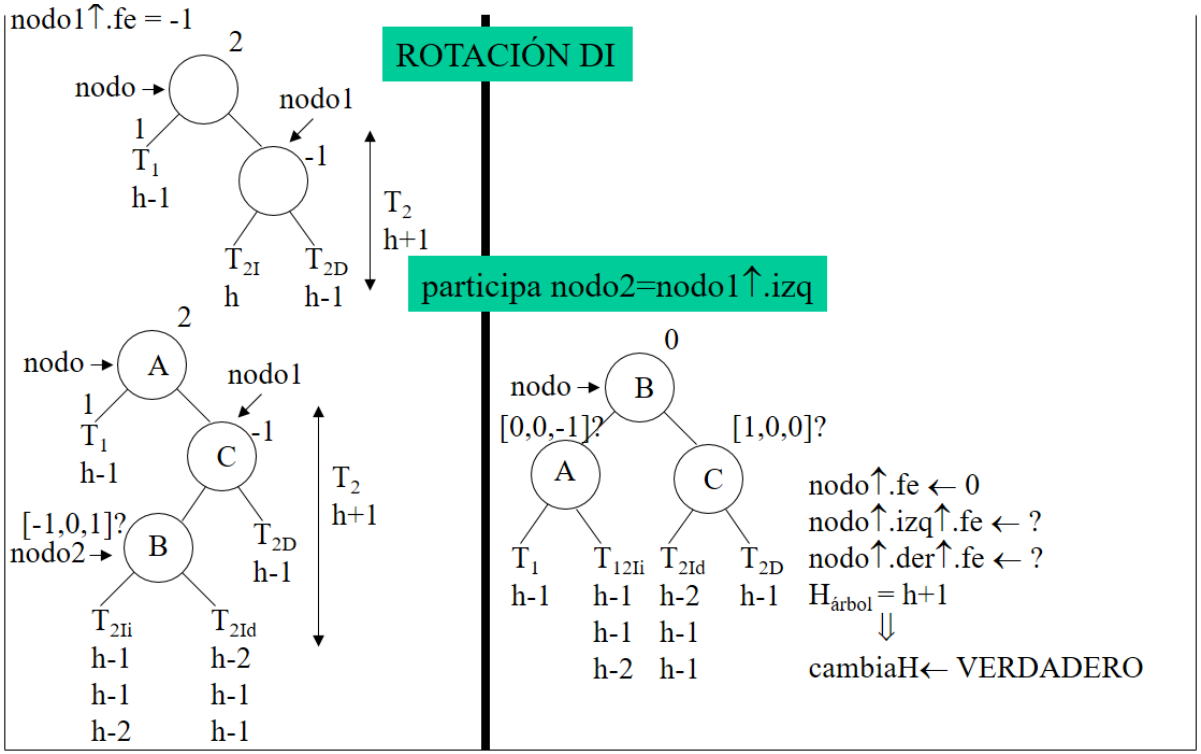
Algoritmo rotación Derecha Derecha modificado para eliminación

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

```
 1: nodo↑.der ← nodo1↑.izq
 2: nodo1↑.izq ← nodo                    ->Movimiento nodos en rotación
 3: si nodo1↑.fe = 1 entonces
 4:     nodo↑.fe ← 0
 5:     nodo1↑.fe ← 0                    ->Siempre si la rotación es por una inserción
 6: si no
 7:     nodo↑.fe ← 1
 8:     nodo1↑.fe ← −11
 9: fin si
10: nodo ← nodo1                         ->Nueva raíz del subárbol
```

Antes Eliminación		Después Eliminación por rama Izquierda					
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	nodo1↑.fe			nodo↑.fe	nodo1↑.fe		
Reest.Sub.Derecho		1	RotaciónDD(*)		0	0	h+1	verdadero
nodo1 ←nodo↑.der		0	RotaciónDD(*)		-1	1	h+1	falso
¿DD o DI?		1	RotaciónDI				h+1	verdadero

Casos al equilibrar subárbol DERECHO (continuación II)



Equilibra Rama Derecha (elimanción por izquierda)

Antes Eliminación		Después Eliminación por rama Izquierda					
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	nodo1↑.fe			nodo↑.fe	nodo1↑.fe		
Reest.Sub.Derecho		1	RotaciónDD(*)		0	0	h+1	verdadero
nodo1 ←nodo↑.der		0	RotaciónDD(*)		-1	1	h+1	falso
¿DD o DI?		1	RotaciónDI				h+1	verdadero

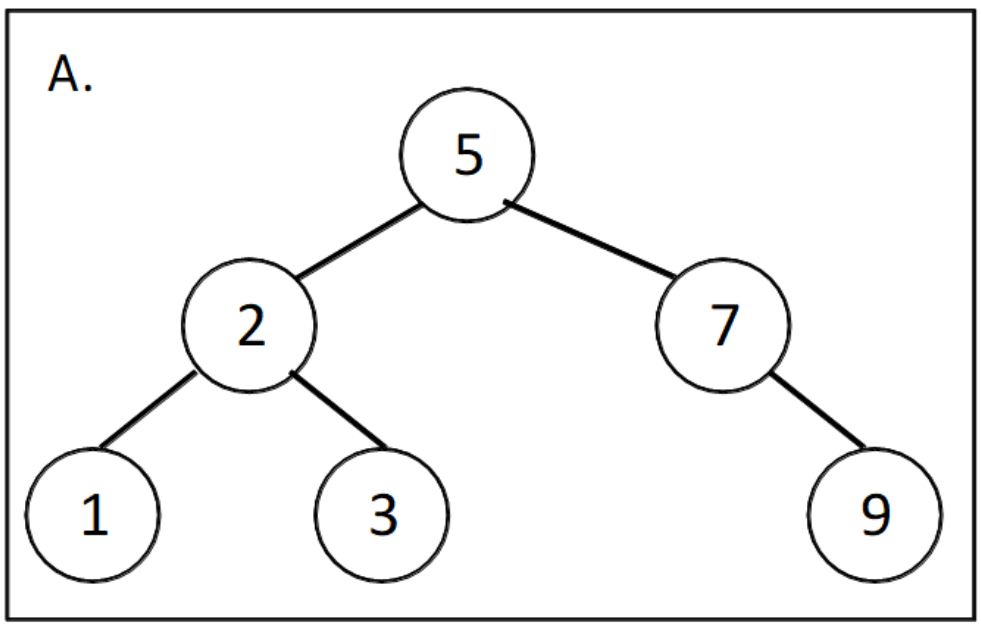
Algorithm equilibarDer(**ref nodo: punteroNodo**; **ref cambiaH:lógico**)

```
 1: casos nodo↑.fe en
 2:     -1: nodo↑.fe ← 0
 3:     0: nodo↑.fe ← 1
 4:        cambiaH ← FALSO
 5:    1: nodo1 ← nodo↑.der
 6:        si nodo1↑.fe ≥ 0 entonces
 7:          rotaciónDD(nodo,nodo1)
 8:          si nodo1↑.fe = 0 entonces
 9:            cambiaH ← FALSO
10:          si no
11:        si no
12:          nodo2← nodo1↑.izq
13:          rotaciónDI(nodo,nodo1,nodo2)
14:        fin si
15:        cambiaH ← FALSO
16: fin casos
```

Ejercicio 1

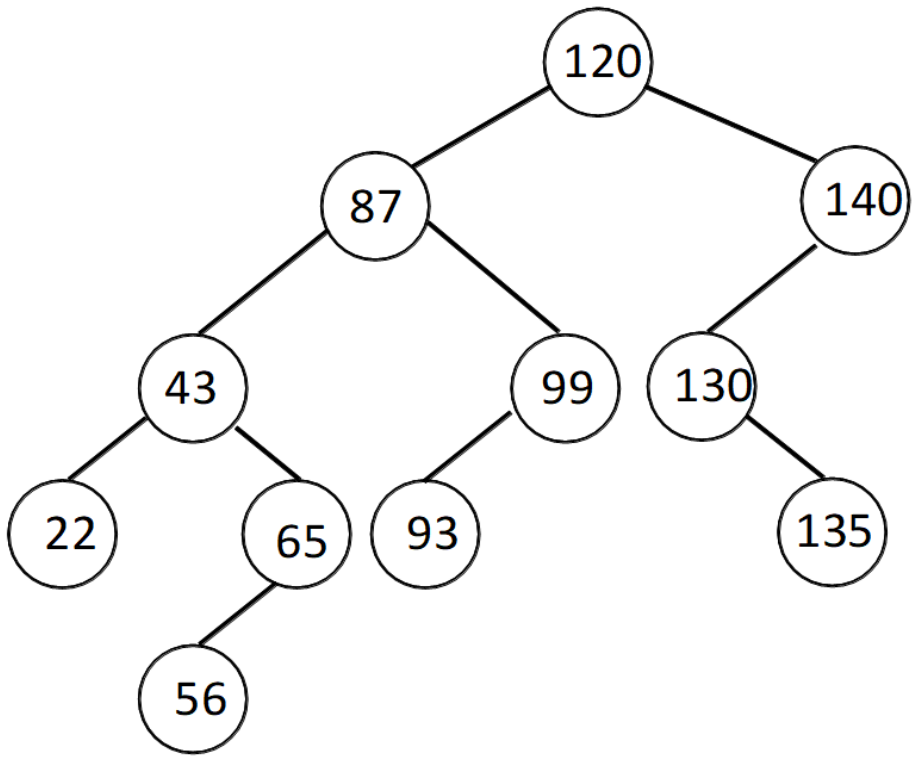
Crear el árbol binario de búsqueda que resulta al insertar las siguientes claves:

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



Ejercicio 2

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

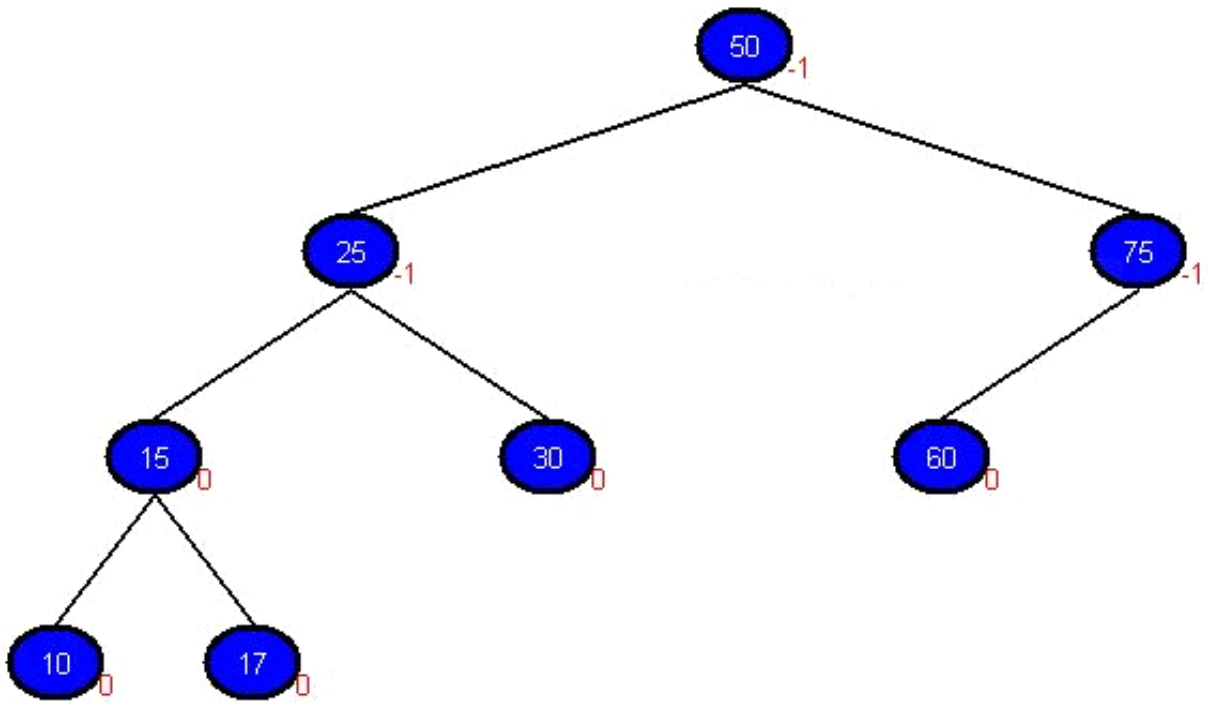


Ejercicio 3

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)?

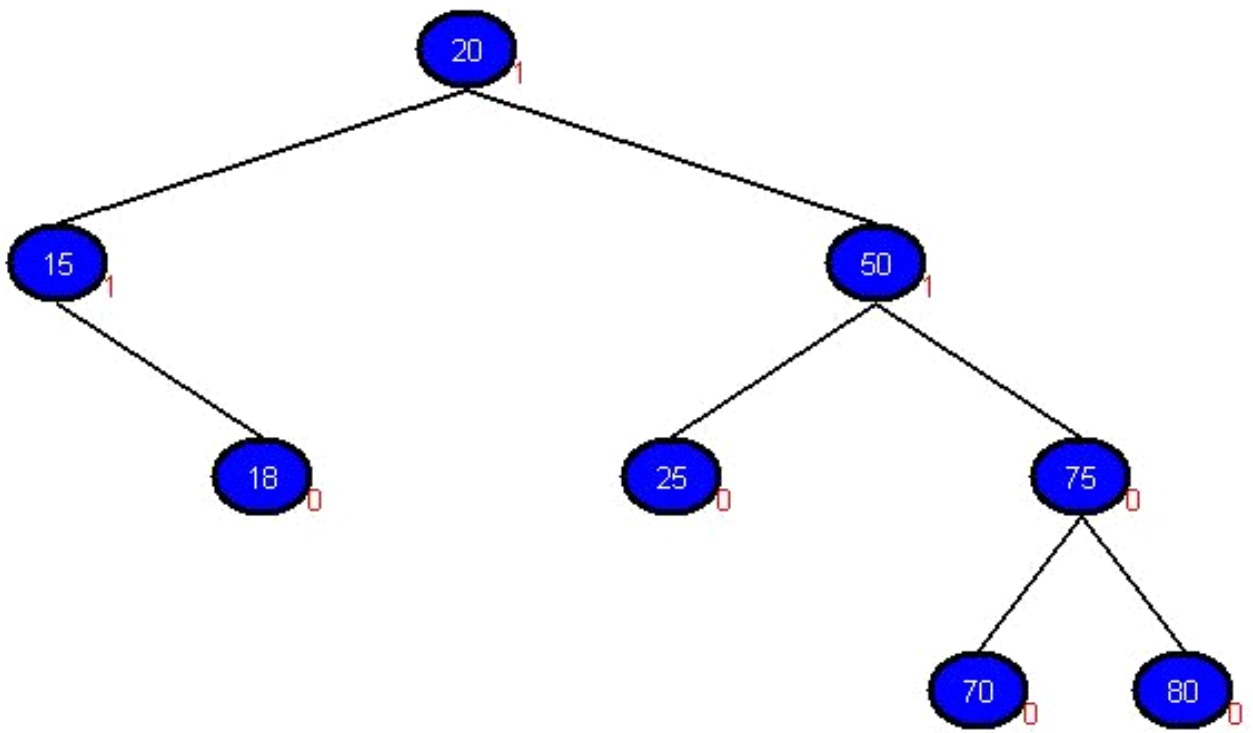


Ejercicio 4

Dado el árbol balanceado de la figura:

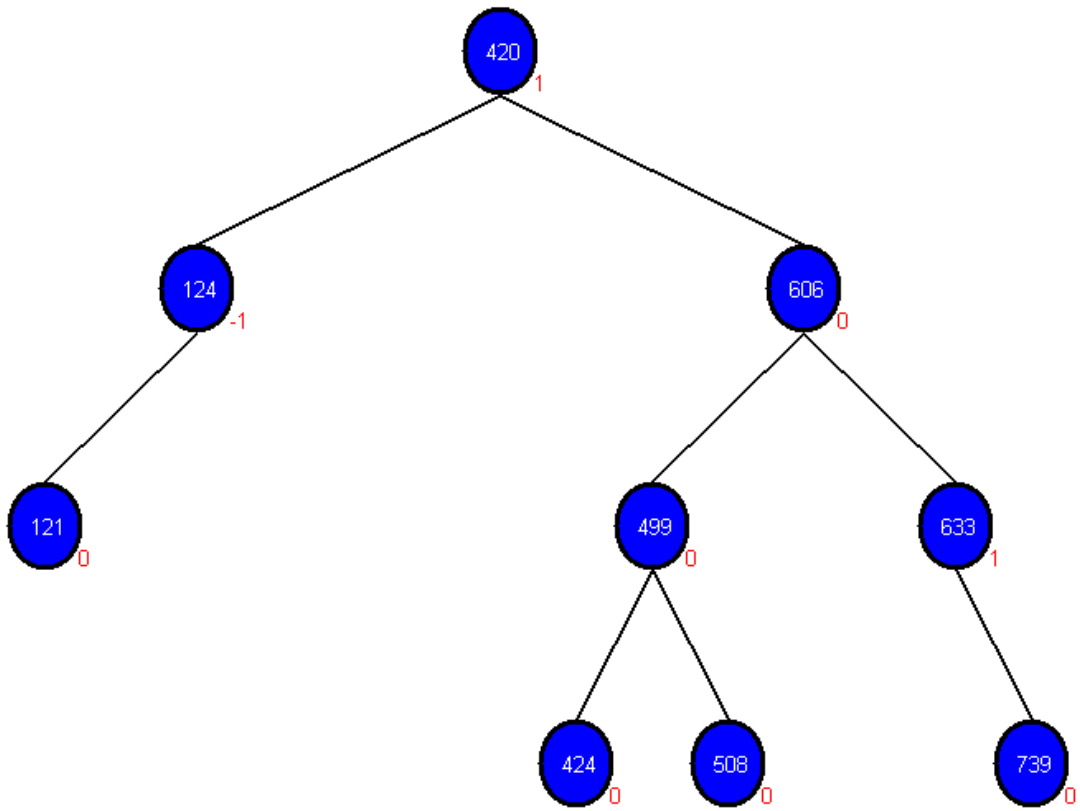
a) Insertar un nodo con valor 90
b) Insertar un nodo con valor 60

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



Ejercicio 5

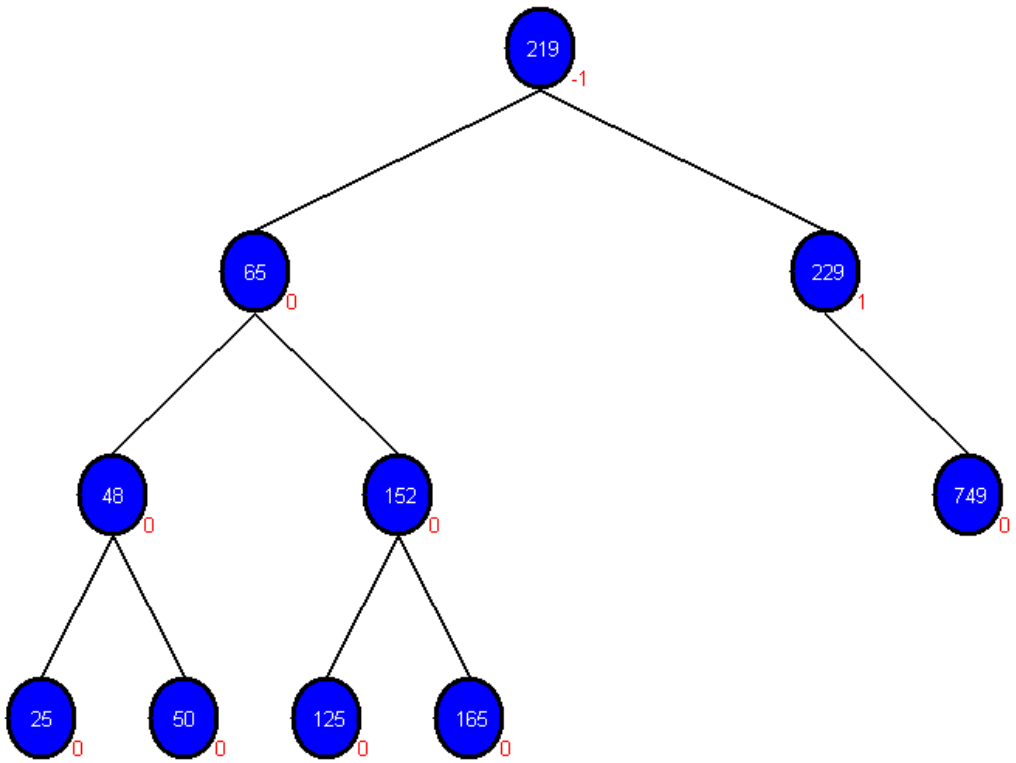
Dado el árbol balanceado de la figura y, teniendo en cuenta la tabla que muestra todos los casos de rotaciones que se pueden presentar ante una eliminación en un árbol balanceado, indicar que tipo de rotación se produce y mostrar cómo queda el árbol después de eliminar el nodo con valor 121



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

Ejercicio 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