

# Fundamentos de análisis y diseño de algoritmos

Árboles rojinegros

Árboles rojinegros

Propiedades de un árbol rojinegro

Rotaciones

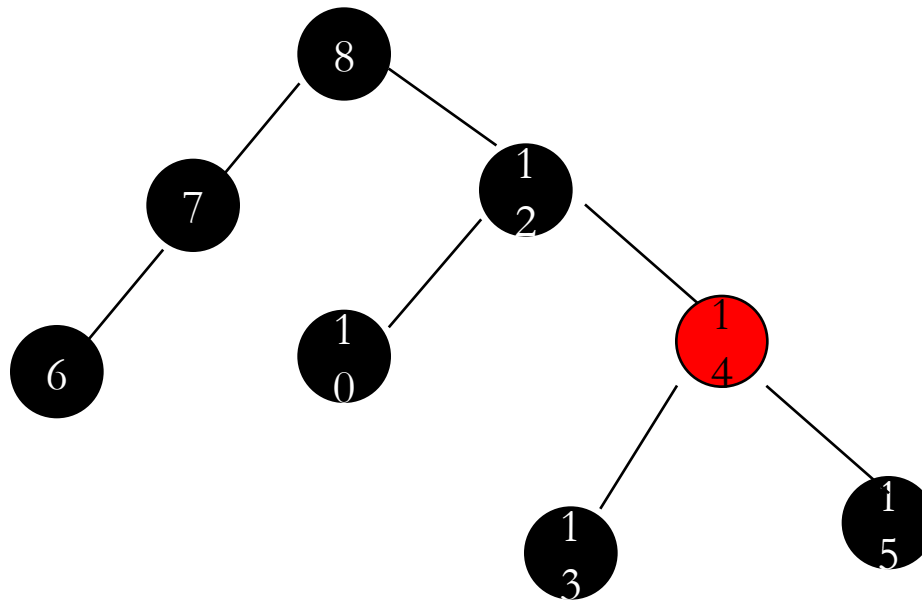
Insertar borrar elementos de un árbol rojinegro

# Árboles rojinegros

---

## Árboles rojinegros

Un árbol rojinegro es un árbol de búsqueda binario en el que cada nodo tiene un bit extra para almacenar su color.

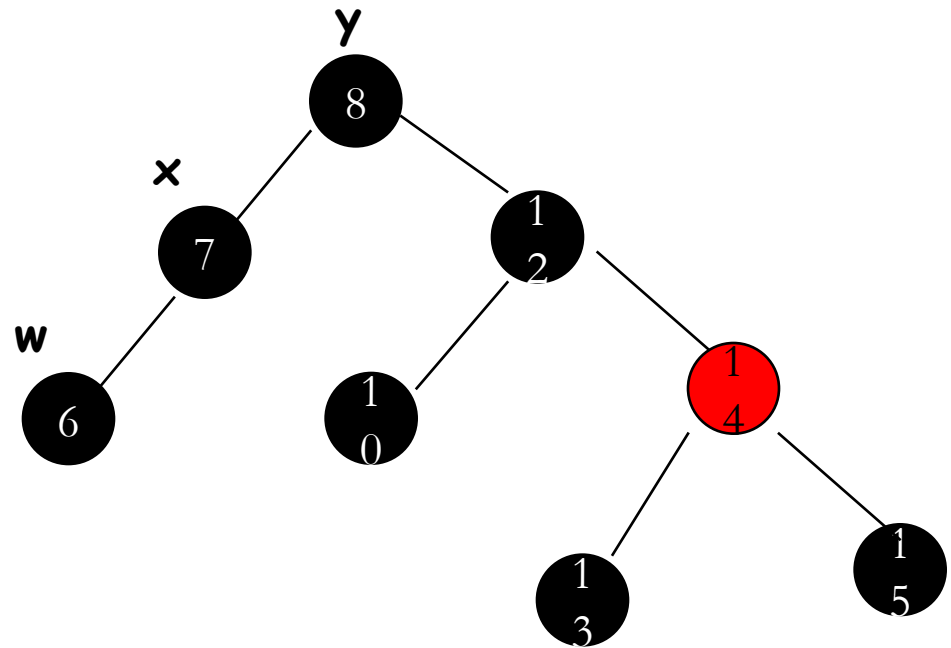


# Árboles rojinegros

## Árboles rojinegros

Un árbol rojinegro es un árbol de búsqueda binario en el que cada nodo tiene un campo extra para almacenar su color

key[x]=7  
p[x]=y  
left[x]=w  
right[x]=nil  
color[x]=black

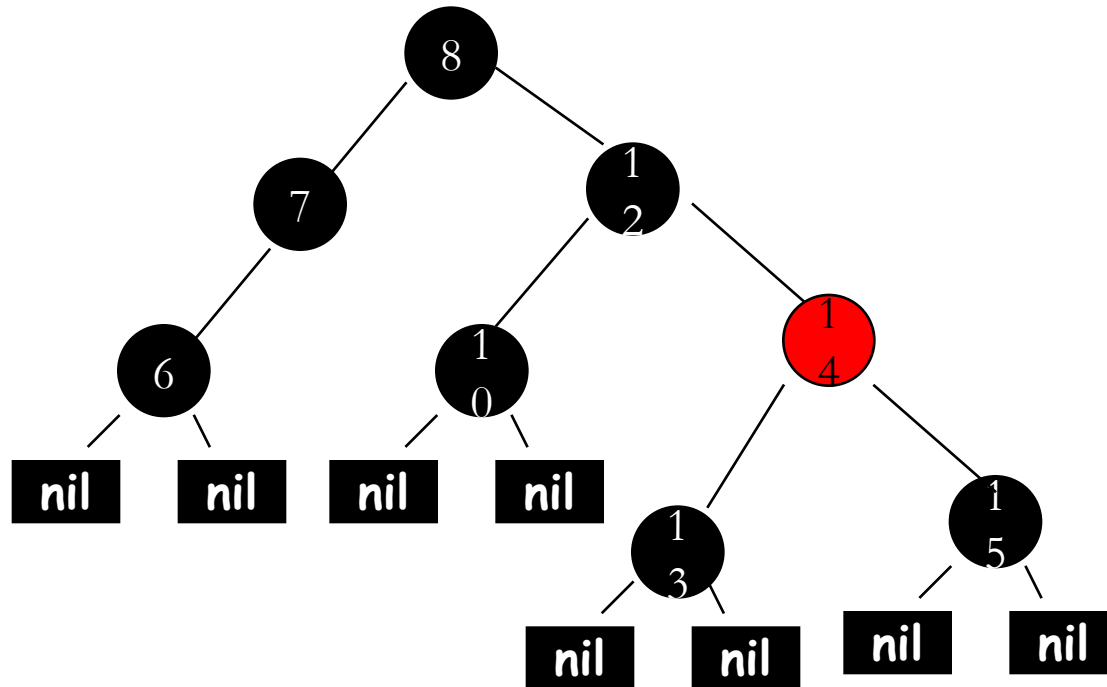


# Árboles rojinegros

---

## Árboles rojinegros

En los árboles rojinegros se colocan las referencias a nil como nodos de color negro



# Árboles rojinegros

---

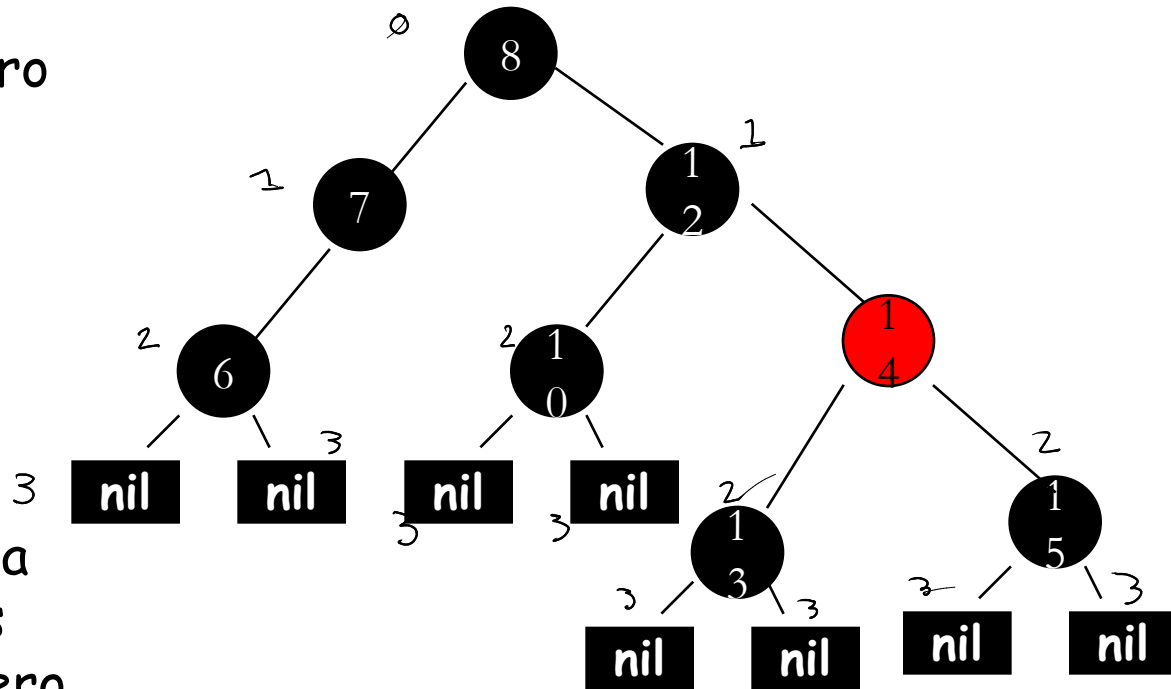
## Propiedades de un árbol rojinegro

1. Todo nodo es rojo o negro
2. Toda hoja (nil) es negra
3. La raíz es negra
4. Si un nodo es rojo, entonces sus hijos son negros
5. Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros

# Árboles rojinegros

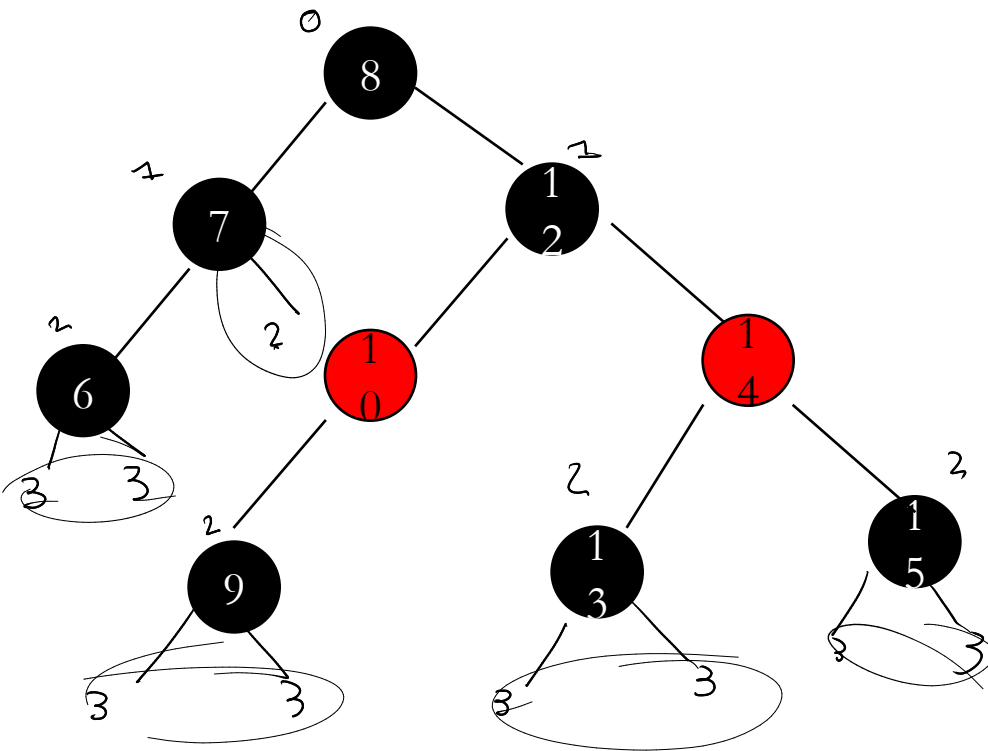
## Propiedades de un árbol rojinegro

1. Todo nodo es rojo o negro
2. Toda hoja (nil) es negra
3. La raíz es negra
4. Si un nodo es rojo, entonces sus hijos son negros
5. Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros



# Árboles rojinegros

Indique si el siguiente árbol es rojinegro



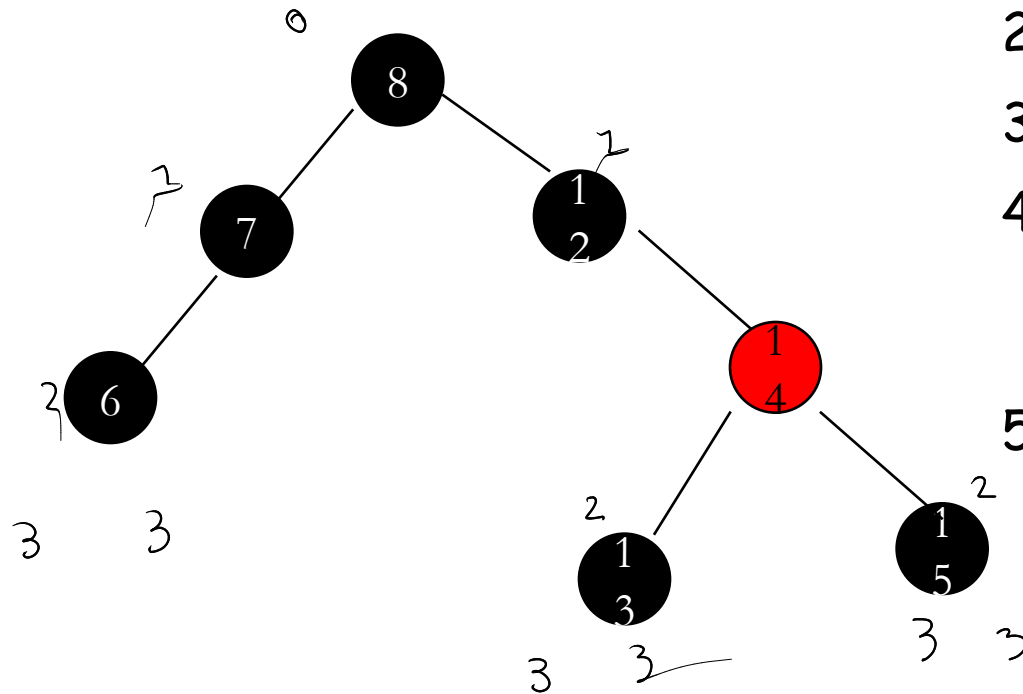
Propiedades de un árbol rojinegro

1. Todo nodo es rojo o negro
2. Toda hoja (nil) es negra
3. La raíz es negra
4. Si un nodo es rojo, entonces sus hijos son negros
5. Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros



# Árboles rojinegros

Indique si el siguiente árbol es rojinegro

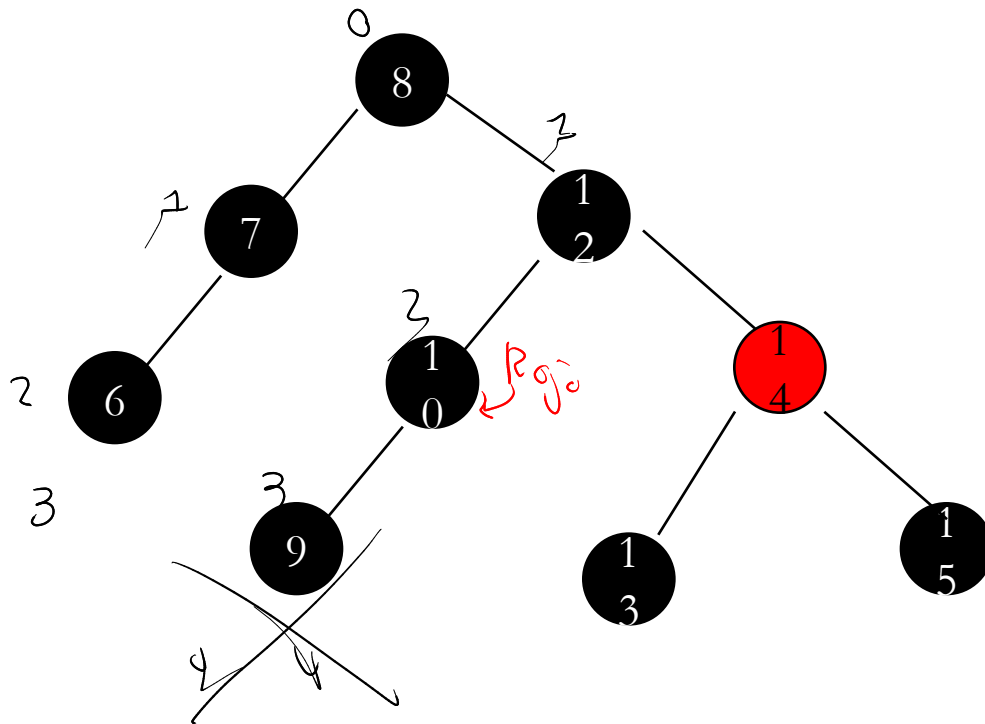


Propiedades de un árbol rojinegro

1. Todo nodo es rojo o negro
2. Toda hoja (nil) es negra
3. La raíz es negra
4. Si un nodo es rojo, entonces sus hijos son negros
5. Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros

# Árboles rojinegros

Indique si el siguiente árbol es rojinegro

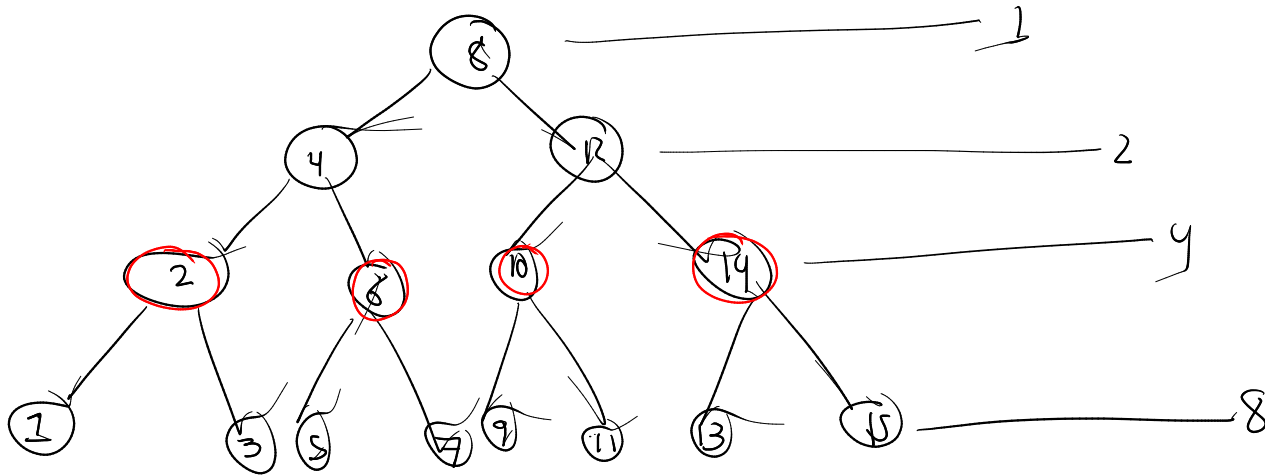


Propiedades de un árbol rojinegro

1. Todo nodo es rojo o negro
2. Toda hoja (nil) es negra
3. La raíz es negra
4. Si un nodo es rojo, entonces sus hijos son negros
5. Cada camino de un nodo a sus hojas descendientes contienen el mismo número de nodos negros

# Árboles rojinegros

- Dibuje el árbol rojinegro completo de altura 3, dadas las llaves {1,2, ..., 15}
- Suponga que la raíz de un árbol rojinegro es de color rojo, Si se cambia a color negro, el árbol será rojinegro?

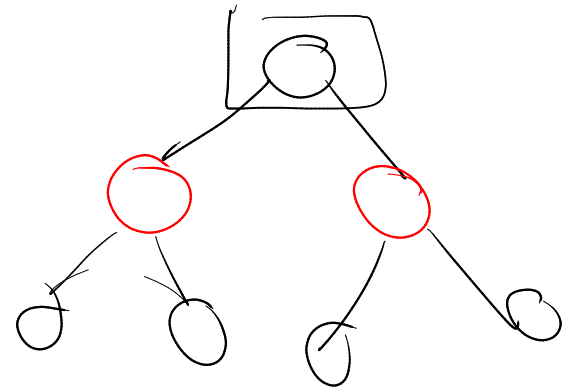
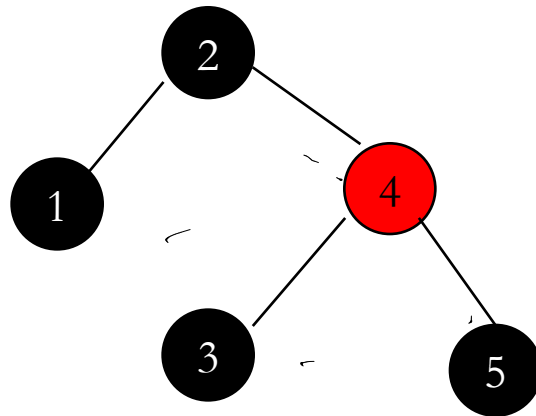
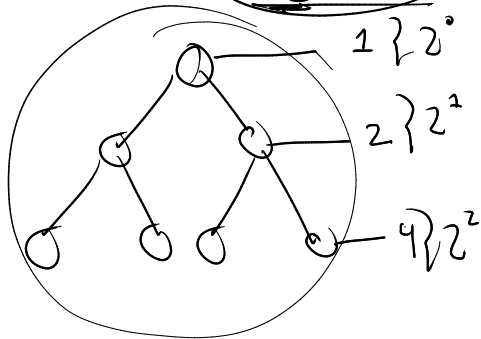


# Árboles rojinegros

## Black-height (bh)

La altura negra de un nodo  $x$ ,  $bh(x)$ , es el número de nodos negros en cualquier camino desde el nodo  $x$  (no incluido) hasta una hoja

Una árbol rojinegro con  $n$  nodos internos tiene altura, a lo más, de  $2\lg(n+1) \rightarrow O(\lg(n))$



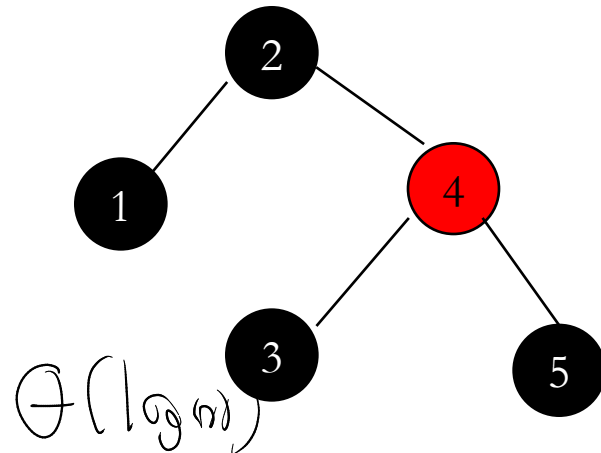
# Árboles rojinegros

## Black-height (bh)

La altura negra de un nodo  $x$ ,  $bh(x)$ , es el número de nodos negros en cualquier camino desde el nodo  $x$  (no incluido) hasta una hoja

Una árbol rojinegro con  $n$  nodos internos tiene altura, a lo más, de  $2\lg(n+1)$

Las operaciones SEARCH, MINIMUM, MAXIMUM, SUCCESSOR, INSERT Y DELETE se pueden realizar en tiempo  $O(h)$ , esto es, en el caso de árboles rojinegros,  $O(\lg n)$



# Árboles rojinegros

---

## Acerca de INSERT y DELETE

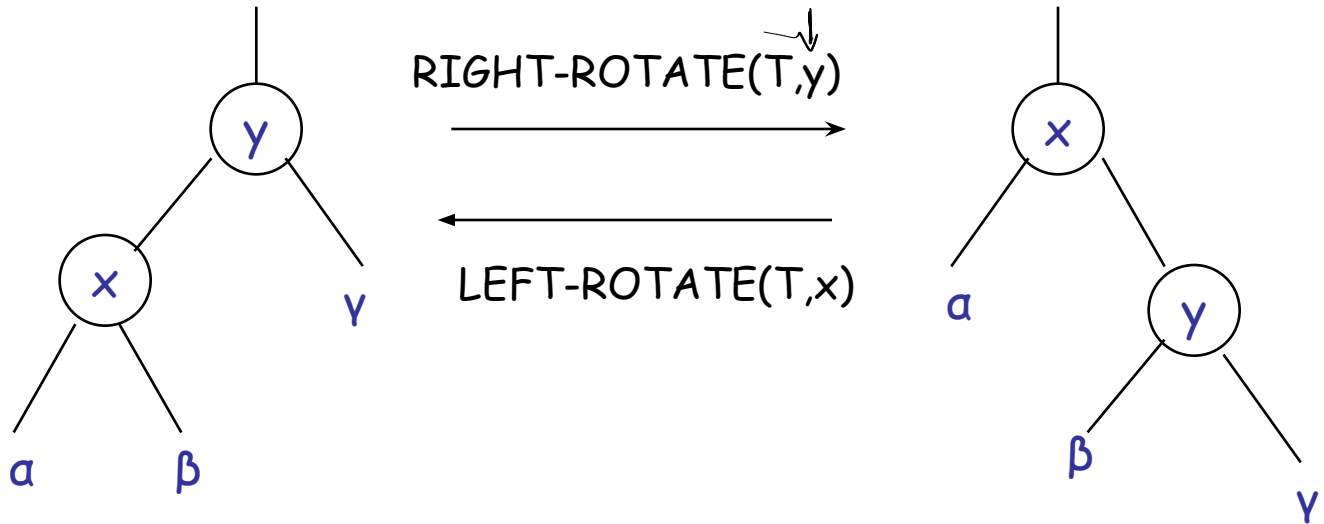
Si se utilizan los procedimientos definidos para los árboles de búsqueda binario se podría violar alguna de las reglas de los árboles rojinegros

Es necesario definir una operación adicional para rotar los nodos

# Árboles rojinegros

## Rotaciones

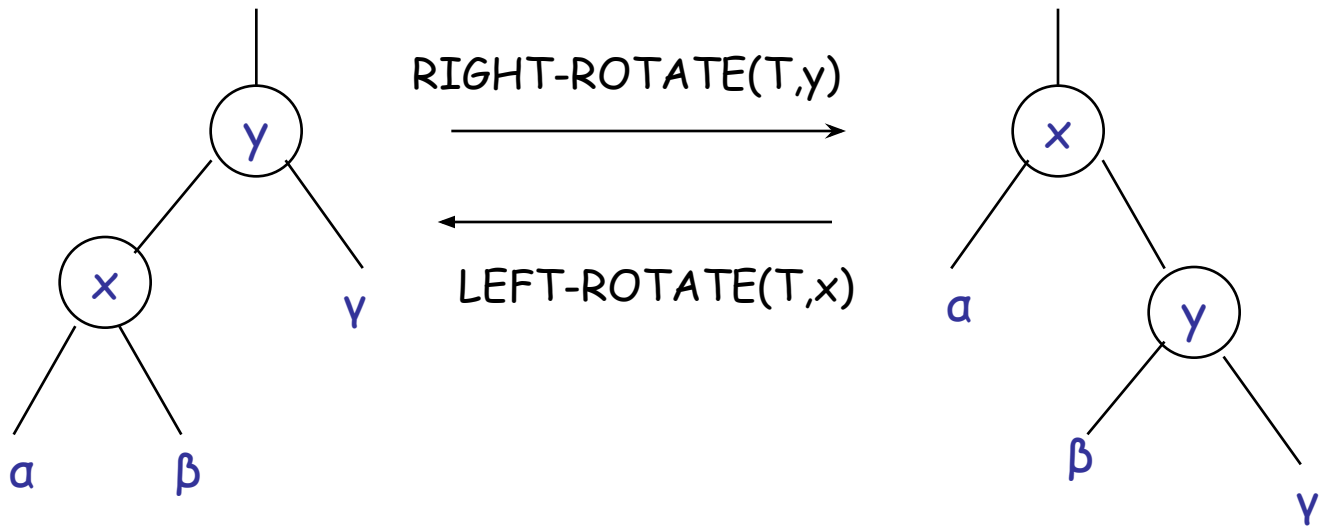
Existen dos procedimientos, uno para rotar a la izquierda y otro a la derecha



# Árboles rojinegros

## Rotaciones

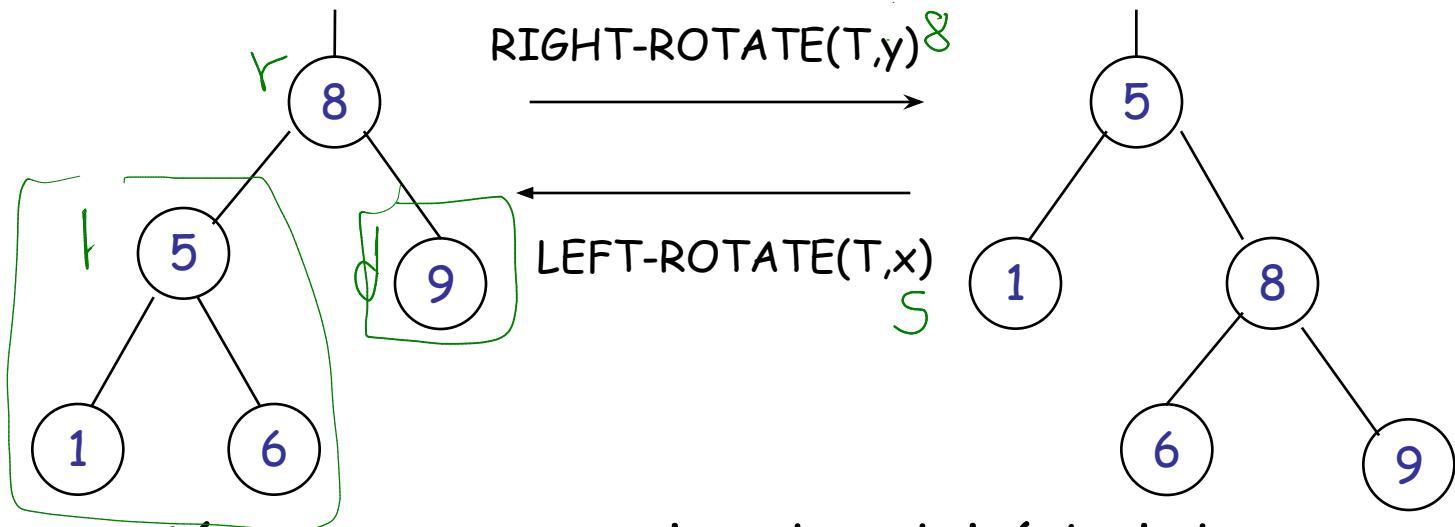
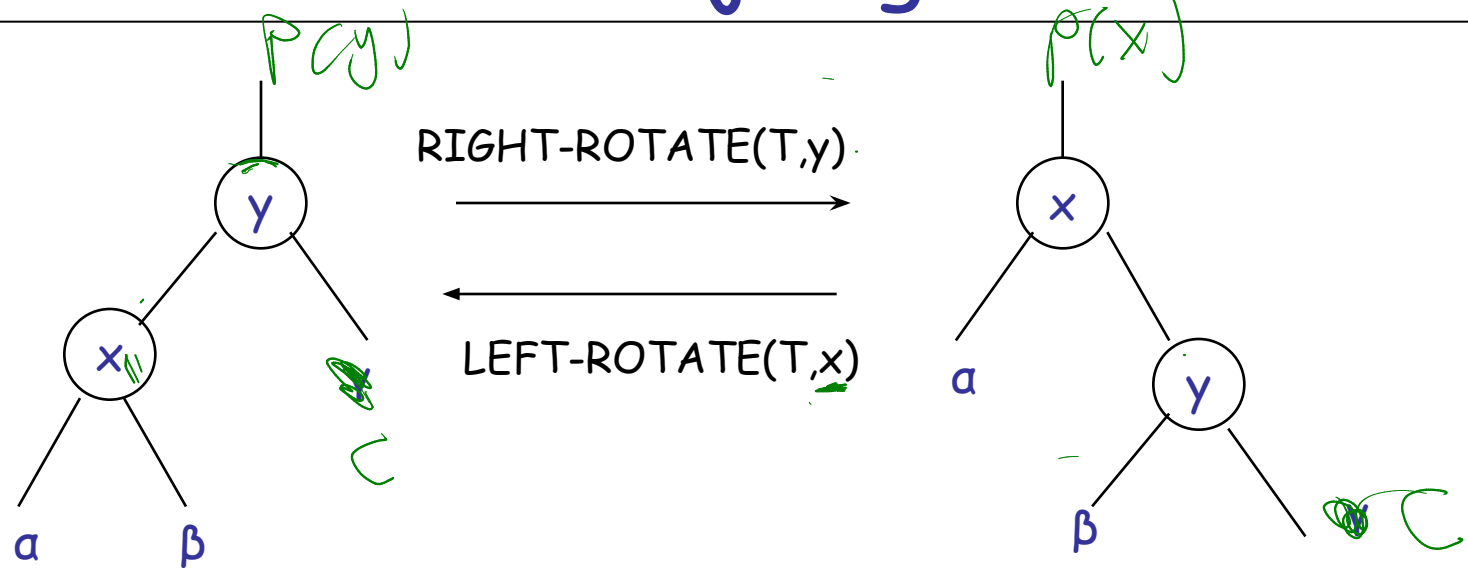
Existen dos procedimientos, uno para rotar a la izquierda y otro a la derecha



Con la rotación se preserva el orden del árbol de búsqueda binaria



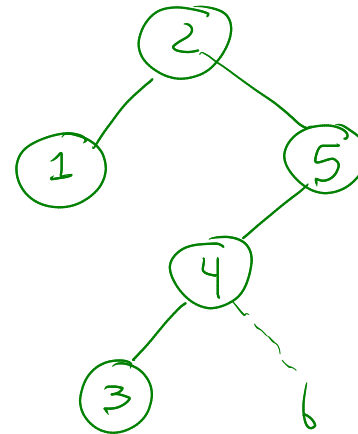
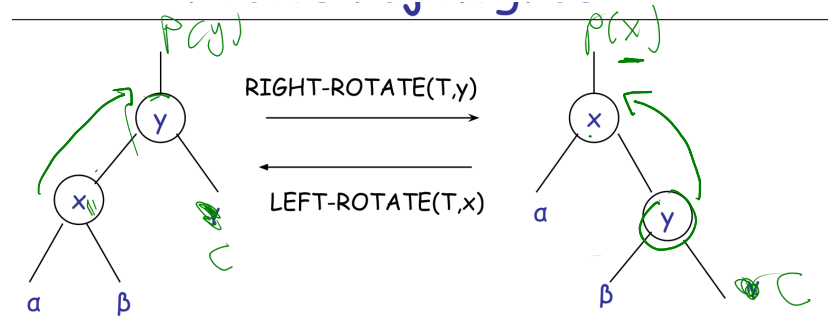
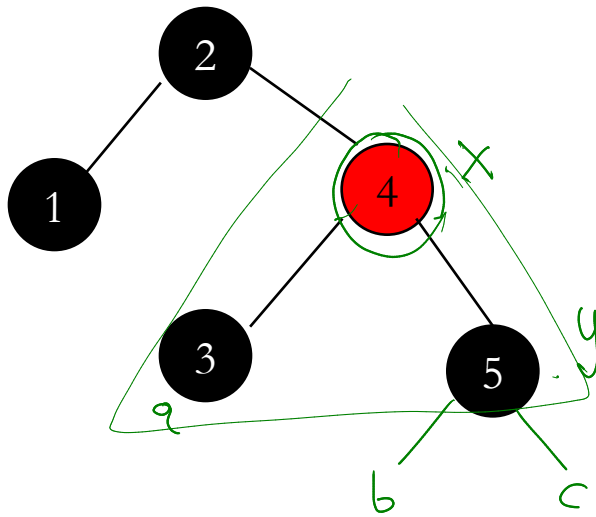
# Árboles rojinegros



Con la rotación se preserva el orden del árbol de búsqueda binaria

# Árboles rojinegros

Indique el resultado de  $\text{LEFT-ROTATE}(T, 4)$



# Árboles rojinegros

---

LEFT-ROTATE( $T, x$ )

$y \leftarrow \text{right}[x]$

$\text{right}[x] \leftarrow \text{left}[y]$

$p[\text{left}[y]] \leftarrow x$

$p[y] \leftarrow p[x]$

if  $p[x] = \text{nil}$

    then  $\text{root}[T] \leftarrow y$

    else if  $x = \text{left}[p[x]]$

        then  $\text{left}[p[x]] \leftarrow y$

        else  $\text{right}[p[x]] \leftarrow y$

$\text{left}[y] \leftarrow x$

$p[x] \leftarrow y$

*Se asume que  $\text{right}[x] \neq \text{nil}$*

# Árboles rojinegros

---

LEFT-ROTATE( $T, x$ )

$y \leftarrow \text{right}[x]$

$\text{right}[x] \leftarrow \text{left}[y]$

$p[\text{left}[y]] \leftarrow x$

$p[y] \leftarrow p[x]$

if  $p[x] = \text{nil}$

    then  $\text{root}[T] \leftarrow y$

    else if  $x = \text{left}[p[x]]$

        then  $\text{left}[p[x]] \leftarrow y$

        else  $\text{right}[p[x]] \leftarrow y$

$\text{left}[y] \leftarrow x$

$p[x] \leftarrow y$

¿Cuál es la complejidad del algoritmo?

$O(1)$

# Árboles rojinegros

LEFT-ROTATE( $T, x$ )

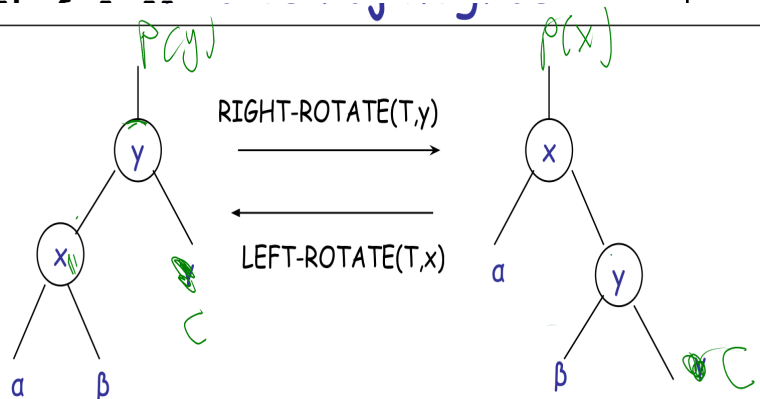
$y \leftarrow \text{right}[x]$

$\text{right}[x] \leftarrow \text{left}[y]$

$p$

$p$

if



then  $\text{left}[p[x]] \leftarrow y$

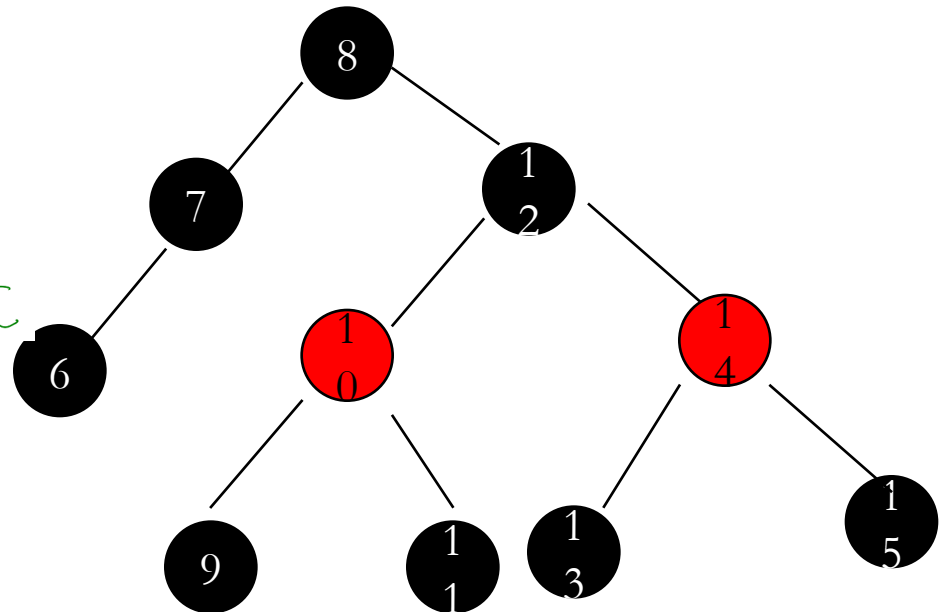
else  $\text{right}[p[x]] \leftarrow y$

$\text{left}[y] \leftarrow x$

$p[x] \leftarrow y$

Siga el algoritmo

LEFT-ROTATE( $T, 12$ )

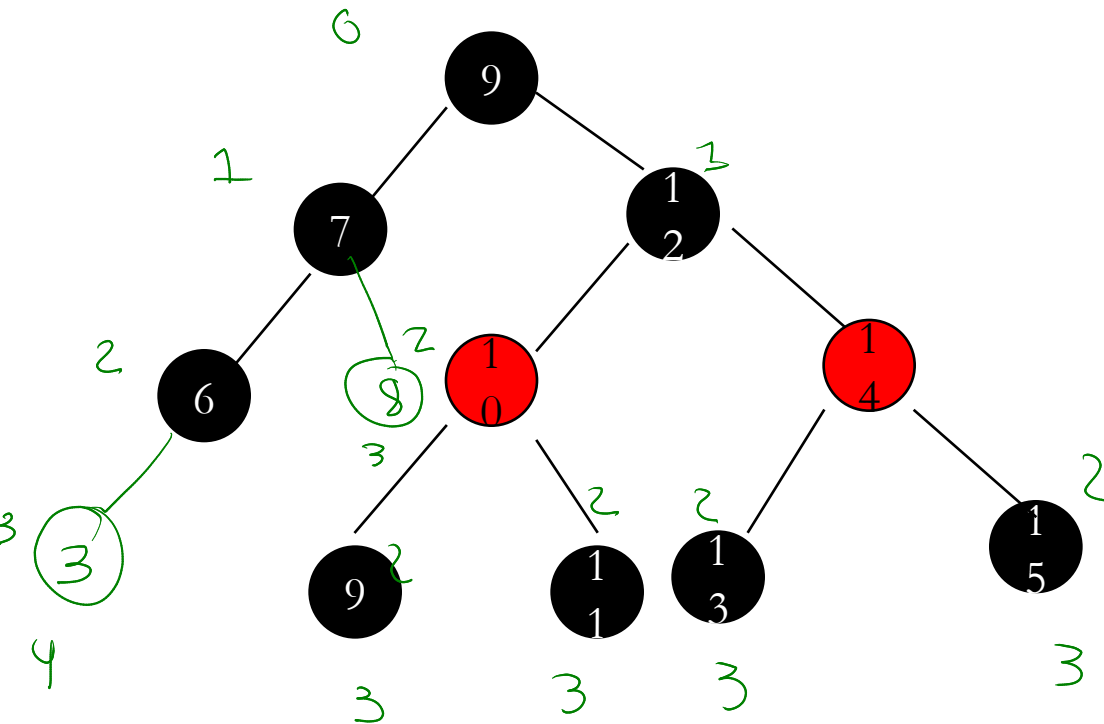


# Árboles rojinegros

## Insertar un nodo en el árbol

Se usa el procedimiento TREE-INSERT y se colorea x de rojo

Luego se modifica el árbol recoloreando nodos y haciendo rotaciones



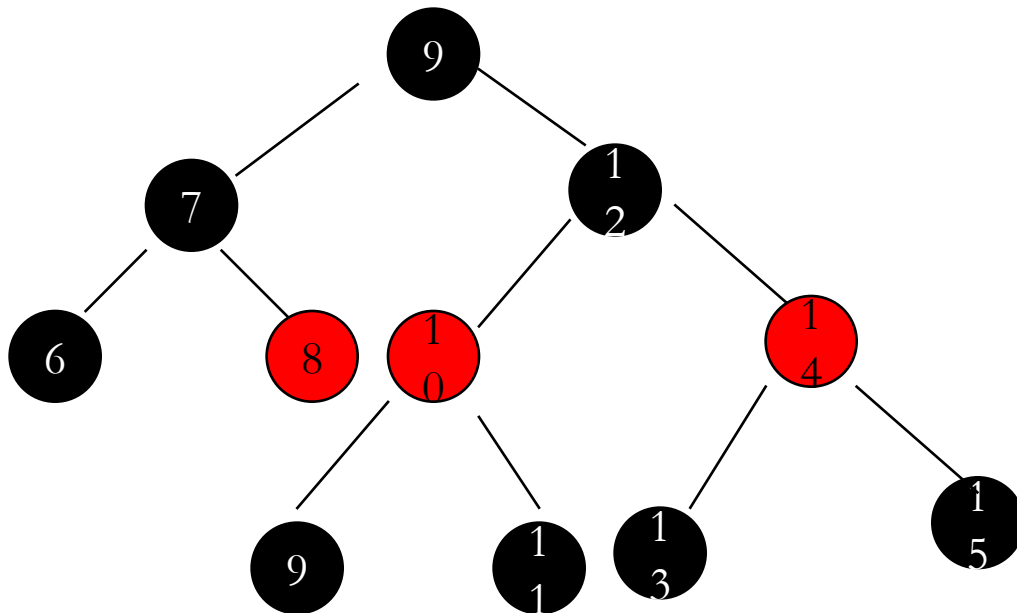
Qué pasa si se inserta el  
nodo con llave 8

# Árboles rojinegros

## Insertar un nodo en el árbol

Se usa el procedimiento TREE-INSERT y se colorea x de rojo

Luego se modifica el árbol recoloreando nodos y haciendo rotaciones



Qué pasa si se inserta el  
nodo con llave 8

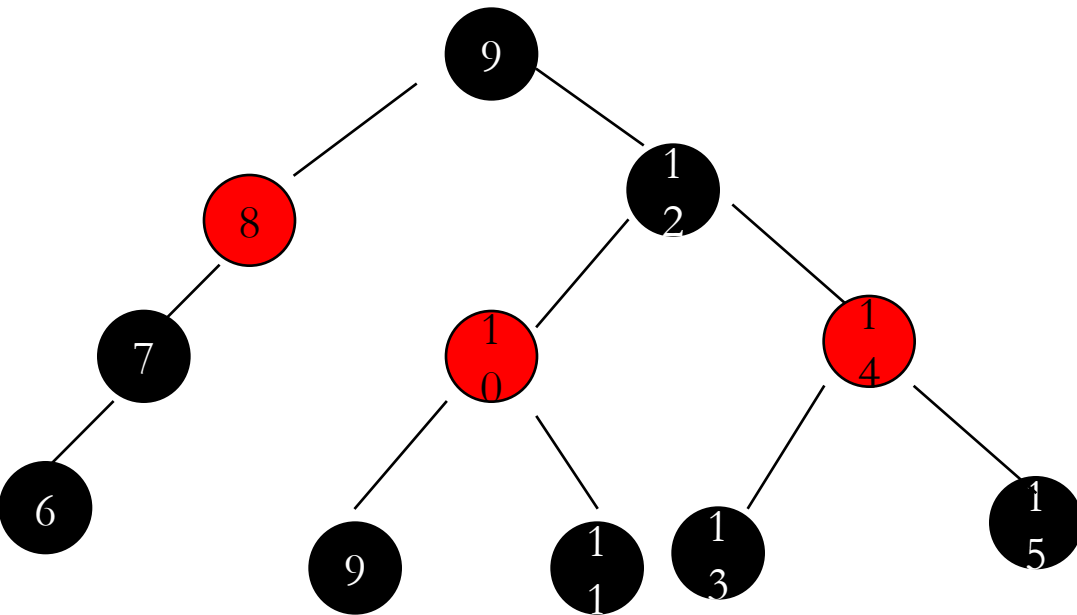
8 no puede ser rojo porque  
no tienes hijos negros

# Árboles rojinegros

## Insertar un nodo en el árbol

Se usa el procedimiento TREE-INSERT y se colorea x de rojo

Luego se modifica el árbol recoloreando nodos y haciendo rotaciones



Qué pasa si se inserta el  
nodo con llave 8

Rotación a la izquierda, aun  
no es rojinegro ...



RB-INSERT(T, x)

TREE-INSERT(T,x)

color[z]←RED

while x≠root[T] and color[p[x]]=RED

do if p[x]=left[p[p[x]]]

then y←right[p[p[z]]]

if color[y]=RED

then color[p[x]]←BLACK #Caso1

color[y]←BLACK #Caso1

color[p[p[x]]]←RED #Caso1

x←p[p[x]] #Caso1

else if x=right[p[x]]

then x←p[x] #Caso2

LEFT-ROTATE(T,x) #Caso2

color[p[x]]←BLACK #Caso3

color[p[p[x]]]←RED #Caso3

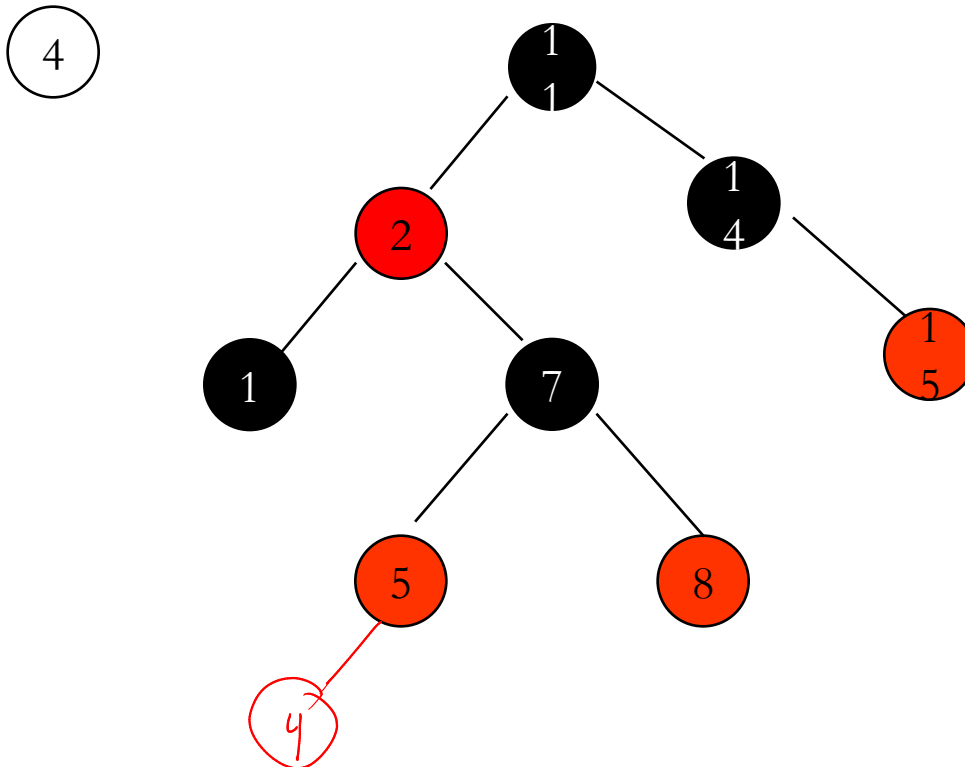
RIGHT-ROTATE(T,p[p[x]]) #Caso3

else procedimiento simétrico cambiando "right" por "left"

color[root[T]]←BLACK

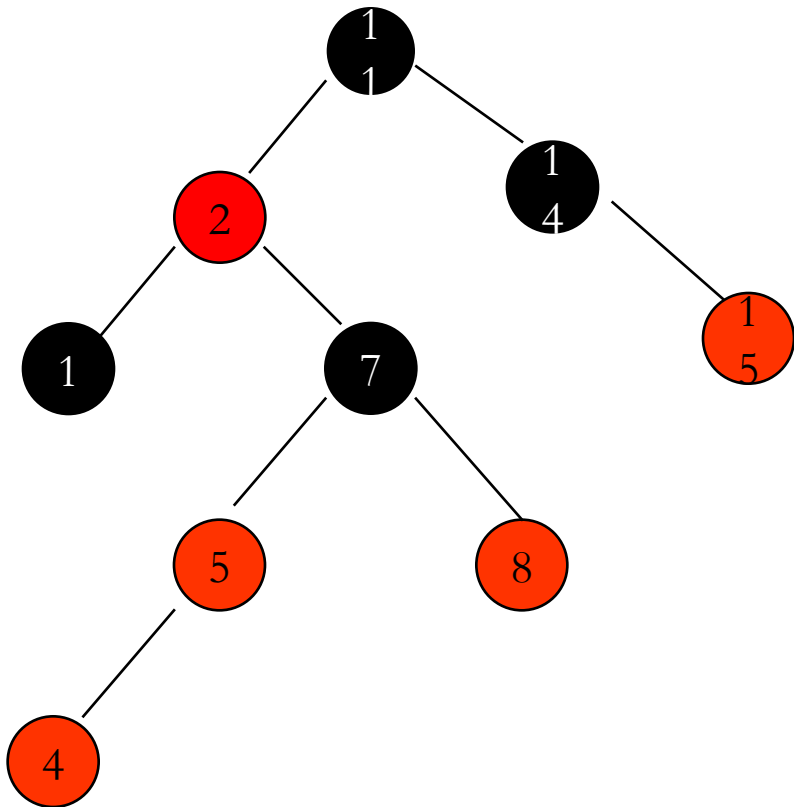
# Árboles rojinegros

Dado el árbol  $T$ , se desea insertar  $x$ ,  $\text{key}[x]=4$



# Árboles rojinegros

Dado el árbol  $T$ , se desea insertar  $x$ ,  $\text{key}[x]=4$

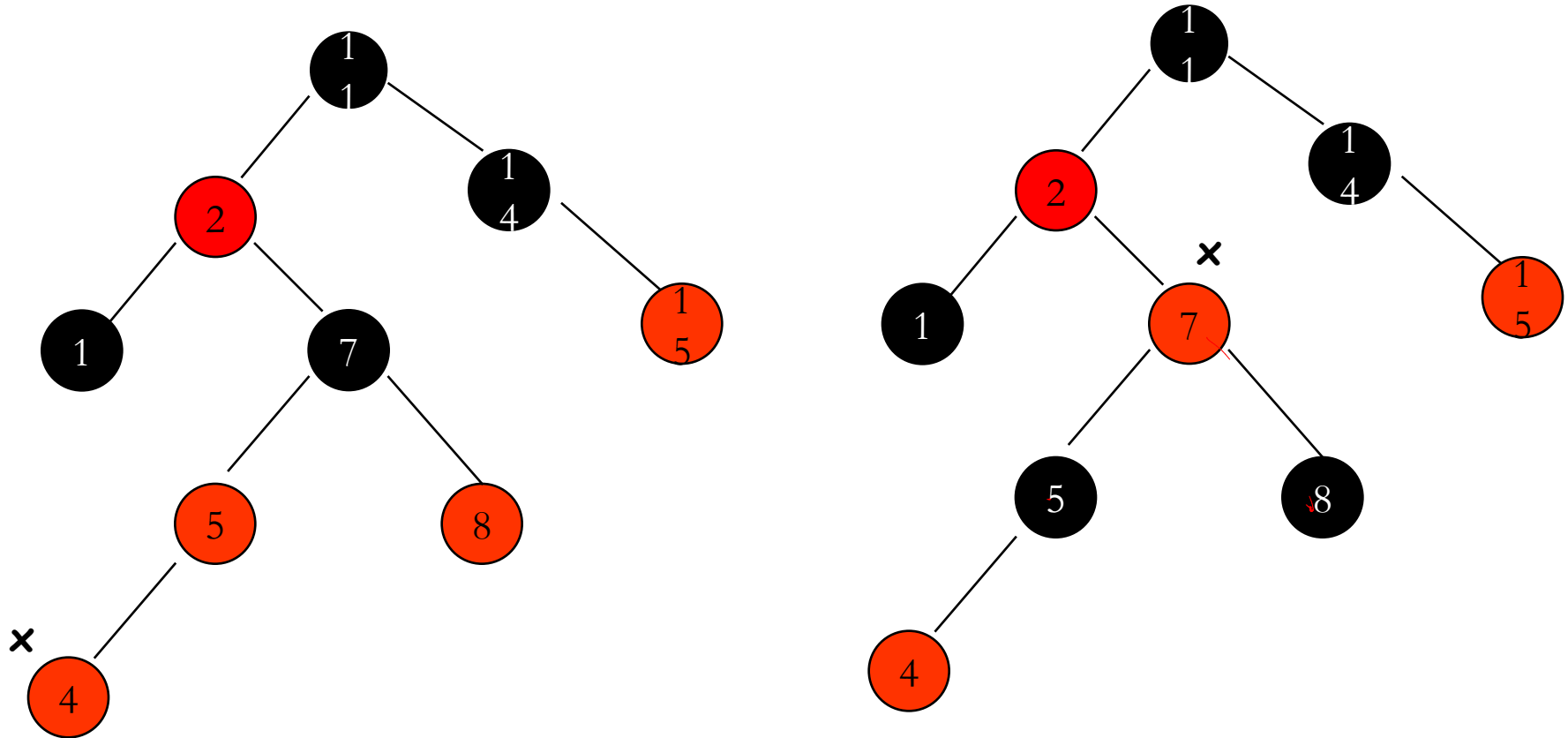


Se inserta y se colorea de rojo

El árbol resultante no es rojinegro

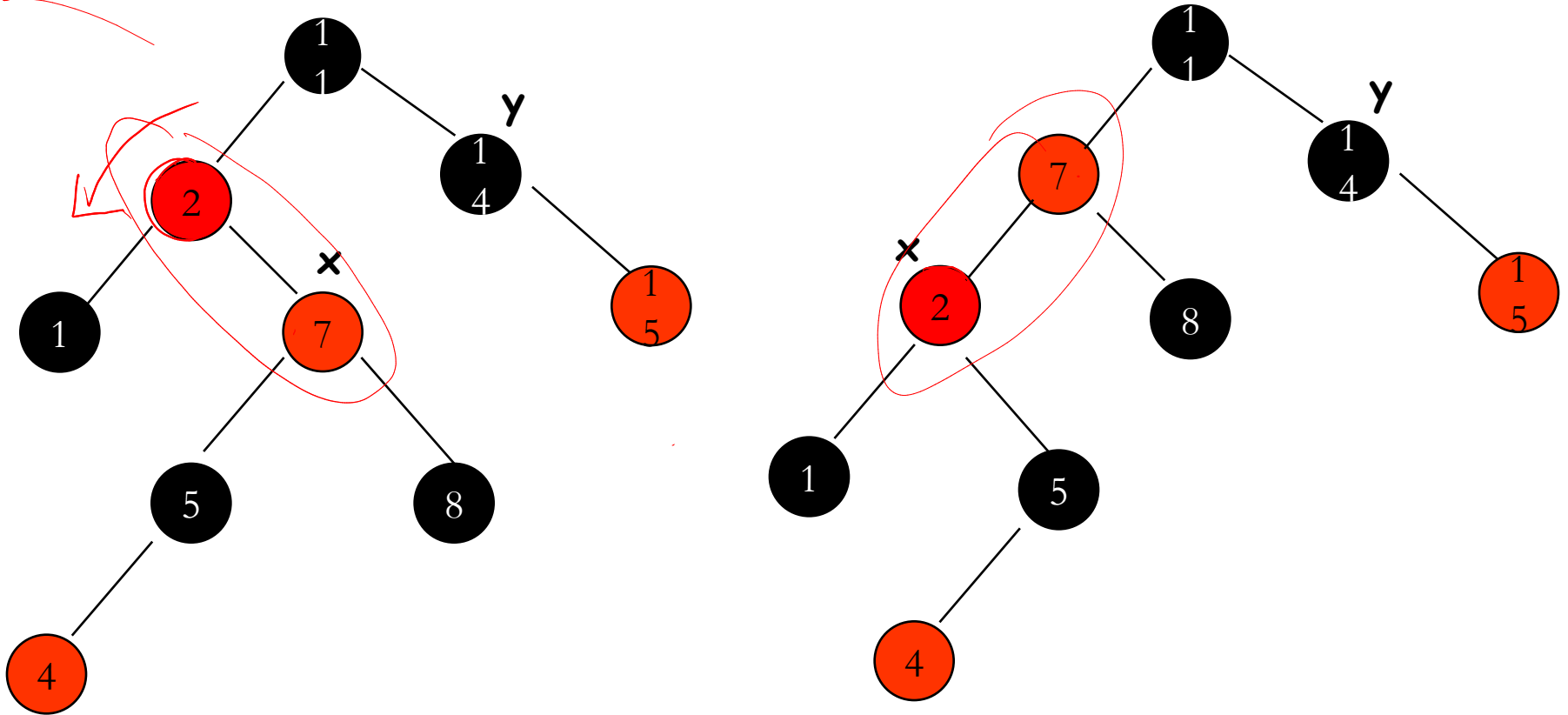
# Árboles rojinegros

**Caso 1:** El tío de  $x$  es rojo, se pintan de negro padre y tío de  $x$ , el abuelo de  $x$  queda entonces de rojo



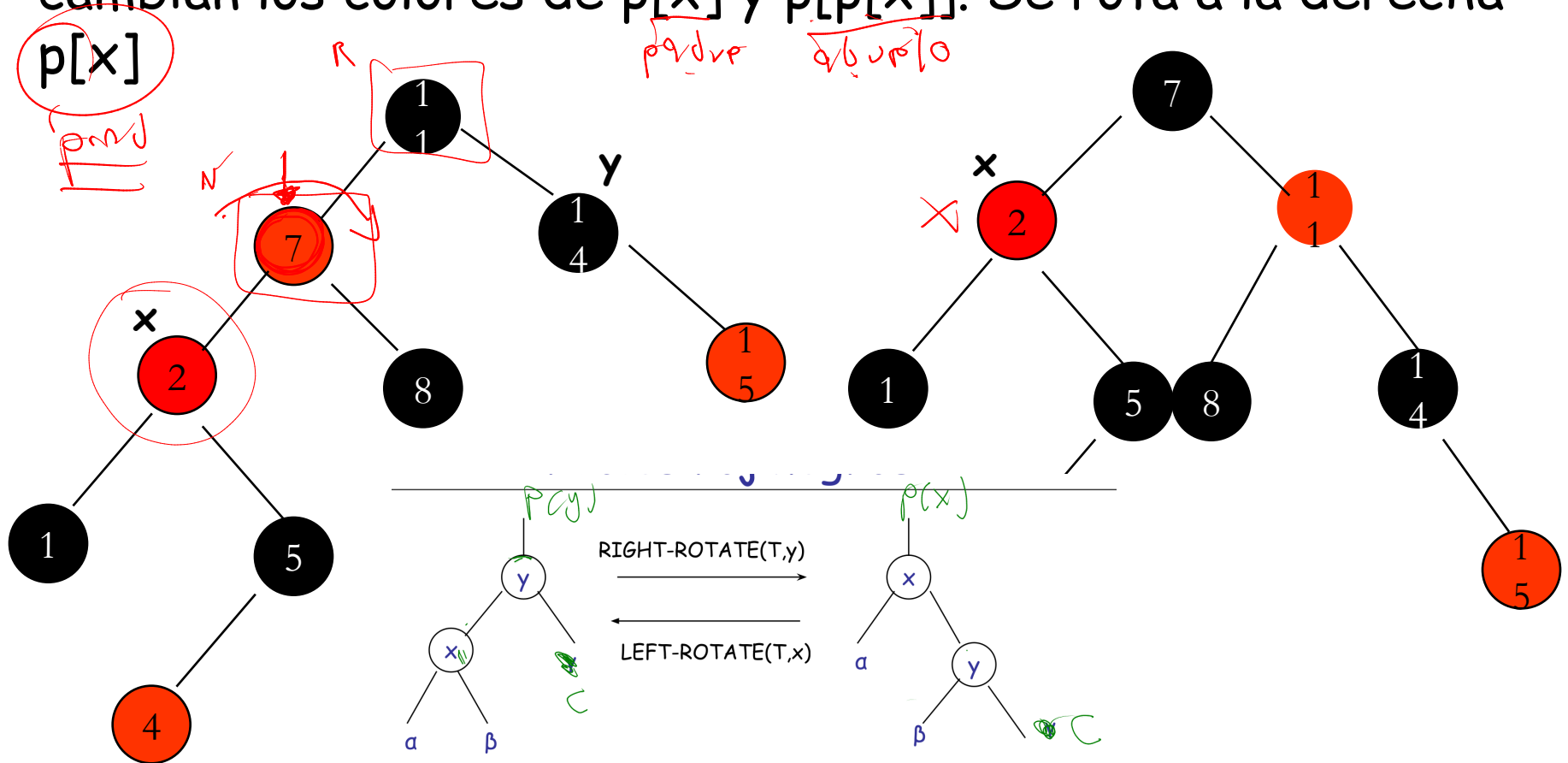
# Árboles rojinegros

**Caso 2:** El tío de  $x$ ,  $y$ , es ahora negro. Se rota a la izquierda  $p[x]$



# Árboles rojinegros

**Caso 3:**  $x$  (rojo) es el hijo izquierdo de un padre rojo. Se cambian los colores de  $p[x]$  y  $p[p[x]]$ . Se rota a la derecha



RB-INSERT(T, x)

TREE-INSERT(T,x)

color[z]←RED

while x≠root[T] and color[p[x]]=RED

do if p[x]=left[p[p[x]]]

then y←right[p[p[z]]]

if color[y]=RED

then color[p[x]]←BLACK #Caso1

color[y]←BLACK #Caso1

color[p[p[x]]]←RED #Caso1

x←p[p[x]] #Caso1

else if x=right[p[x]]

then x←p[x] #Caso2

LEFT-ROTATE(T,x) #Caso2

color[p[x]]←BLACK #Caso3

color[p[p[x]]]←RED #Caso3

RIGHT-ROTATE(T,p[p[x]]) #Caso3

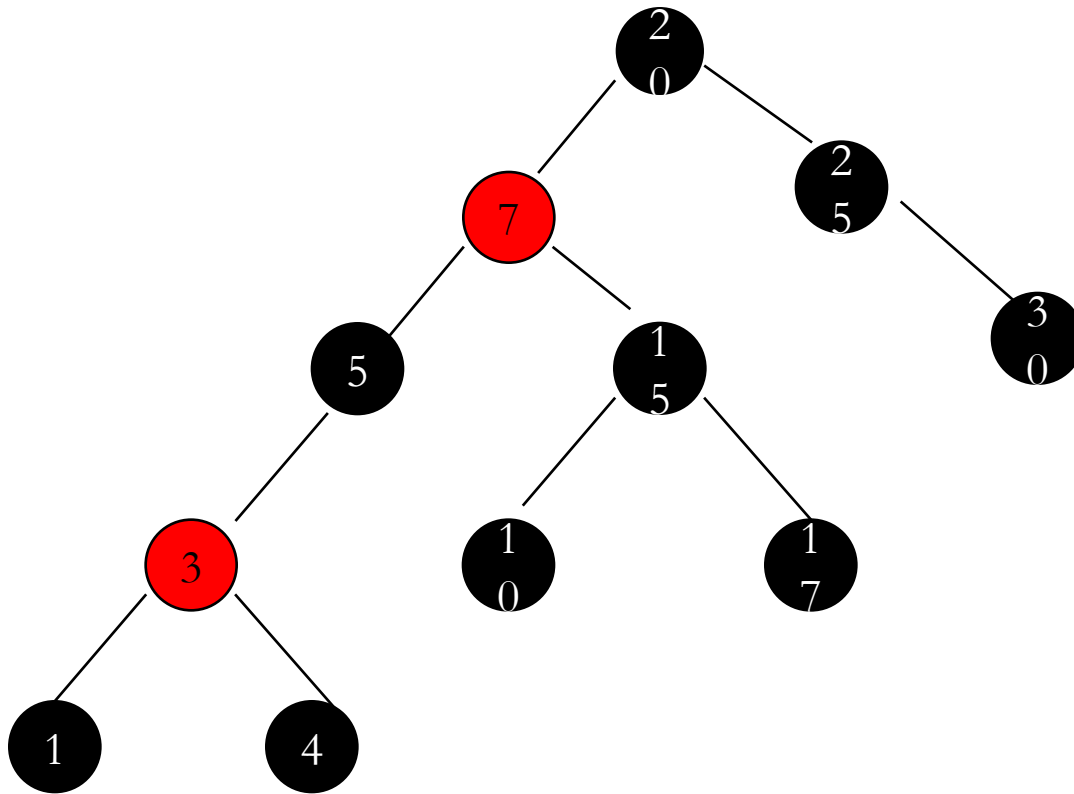
else procedimiento simétrico cambiando "right" por "left"

color[root[T]]←BLACK

# Árboles rojinegros

---

Siga el algoritmo RB-INSERT( $T, x$ ), donde  $\text{key}[x]=11$

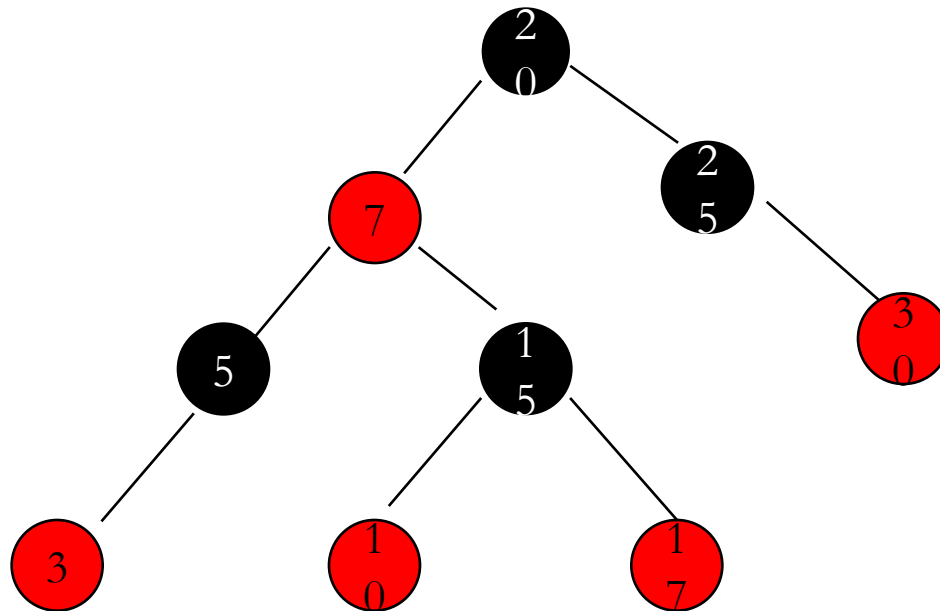




# Árboles rojinegros

---

Siga el algoritmo RB-INSERT( $T, x$ ),  $\text{key}[x]=12$



# Árboles rojinegros

---

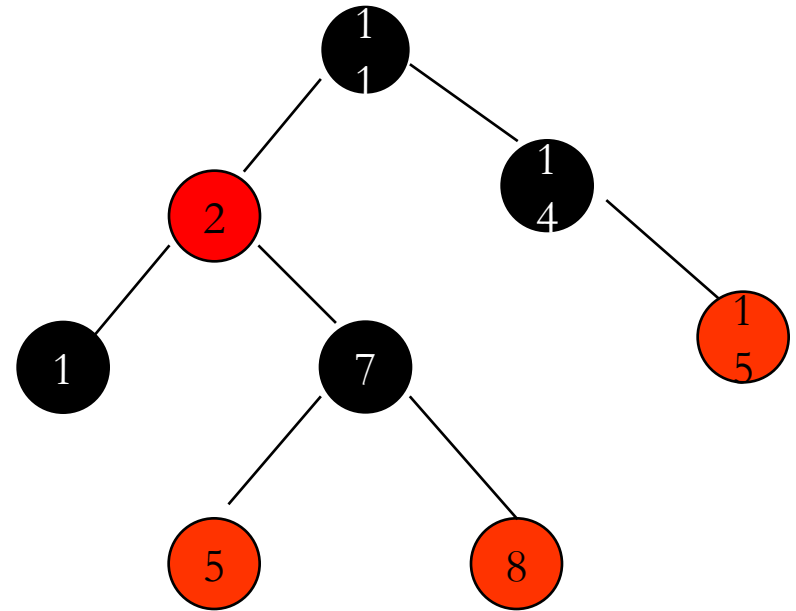
## Eliminación

Es una modificación de TREE-DELETE:

- Todas las referencias a nil se reemplazan por un nodo centinela nil[T]. De esta forma, un nodo nil tiene un puntero a su padre
- Se debe hacer un llamado al procedimiento TREE-DELETE-FIXUP para cumplir con las condiciones de los árboles rojinegros

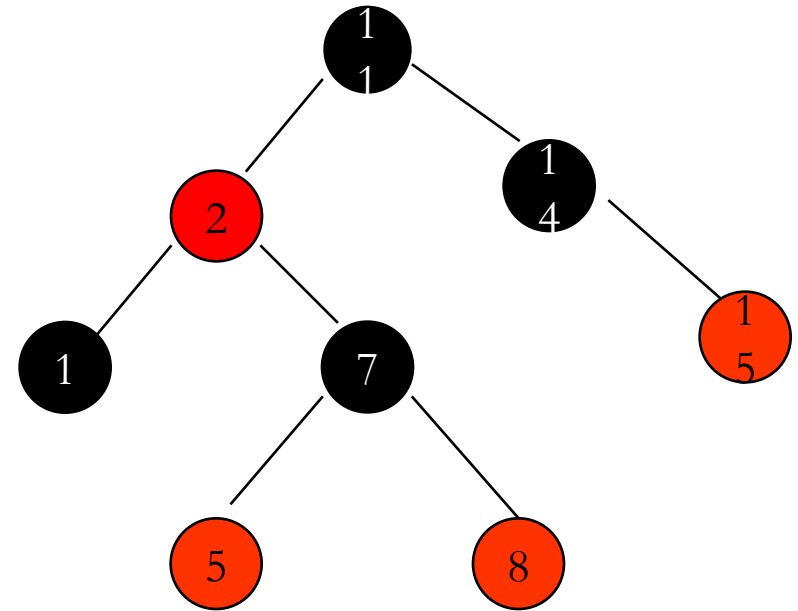
## RB-DELETE( $T, z$ )

```
if left[z]=nil or right[z]=nil[T]
  then  $y \leftarrow z$ 
  else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
if left[y]  $\neq$  nil[T]
  then  $x \leftarrow \text{left}[y]$ 
  else  $x \leftarrow \text{right}[y]$ 
 $p[x] \leftarrow p[y]$ 
if  $p[y] = \text{nil}[T]$ 
  then  $\text{root}[T] \leftarrow x$ 
  else if  $y = \text{left}[p[y]]$ 
    then  $\text{left}[p[y]] \leftarrow x$ 
    else  $\text{right}[p[y]] \leftarrow x$ 
if  $y \neq z$ 
  then  $\text{key}[z] \leftarrow \text{key}[y]$ 
if  $\text{color}[y] = \text{BLACK}$ 
  then RB-DELETE-FIXUP( $T, x$ )
return  $y$ 
```



## RB-DELETE( $T, z$ )

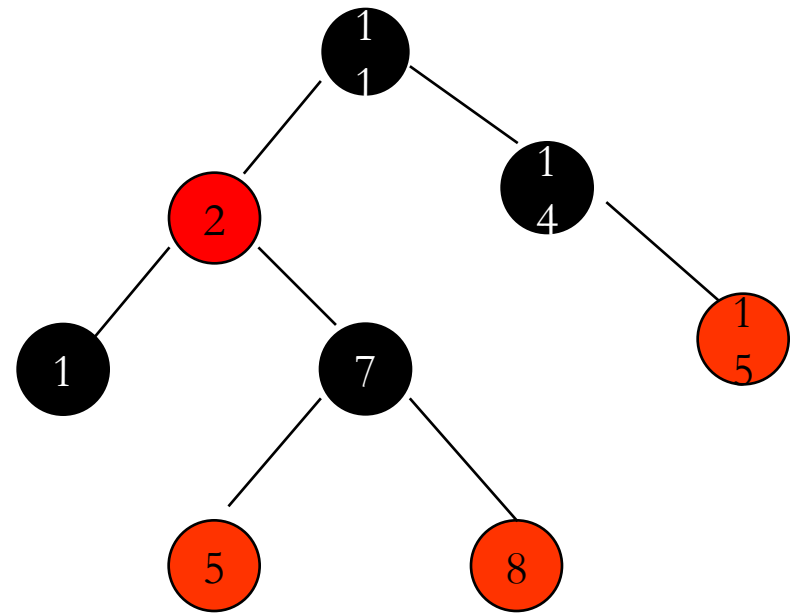
```
if left[z]=nil or right[z]=nil[T]
  then  $y \leftarrow z$ 
  else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
if left[y]  $\neq$  nil[T]
  then  $x \leftarrow \text{left}[y]$ 
  else  $x \leftarrow \text{right}[y]$ 
 $p[x] \leftarrow p[y]$ 
if  $p[y] = \text{nil}[T]$ 
  then  $\text{root}[T] \leftarrow x$ 
  else if  $y = \text{left}[p[y]]$ 
    then  $\text{left}[p[y]] \leftarrow x$ 
    else  $\text{right}[p[y]] \leftarrow x$ 
if  $y \neq z$ 
  then  $\text{key}[z] \leftarrow \text{key}[y]$ 
if  $\text{color}[y] = \text{BLACK}$ 
  then RB-DELETE-FIXUP( $T, x$ )
return  $y$ 
```



RB-DELETE( $T, z$ ), donde  
 $\text{key}[z] = 5$

## RB-DELETE( $T, z$ )

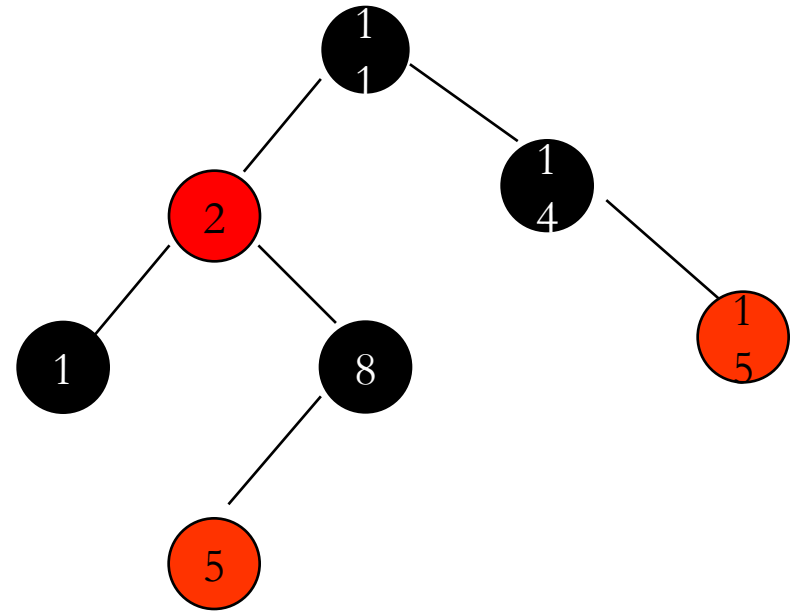
```
if left[z]=nil or right[z]=nil[T]
  then  $y \leftarrow z$ 
  else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
if left[y]  $\neq$  nil[T]
  then  $x \leftarrow \text{left}[y]$ 
  else  $x \leftarrow \text{right}[y]$ 
 $p[x] \leftarrow p[y]$ 
if  $p[y] = \text{nil}[T]$ 
  then  $\text{root}[T] \leftarrow x$ 
  else if  $y = \text{left}[p[y]]$ 
    then  $\text{left}[p[y]] \leftarrow x$ 
    else  $\text{right}[p[y]] \leftarrow x$ 
if  $y \neq z$ 
  then  $\text{key}[z] \leftarrow \text{key}[y]$ 
if  $\text{color}[y] = \text{BLACK}$ 
  then RB-DELETE-FIXUP( $T, x$ )
return  $y$ 
```



RB-DELETE( $T, z$ ), donde  
 $\text{key}[z] = 7$

## RB-DELETE( $T, z$ )

```
if left[z]=nil or right[z]=nil[T]
  then  $y \leftarrow z$ 
  else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
if left[y]  $\neq$  nil[T]
  then  $x \leftarrow \text{left}[y]$ 
  else  $x \leftarrow \text{right}[y]$ 
 $p[x] \leftarrow p[y]$ 
if  $p[y] = \text{nil}[T]$ 
  then  $\text{root}[T] \leftarrow x$ 
  else if  $y = \text{left}[p[y]]$ 
    then  $\text{left}[p[y]] \leftarrow x$ 
    else  $\text{right}[p[y]] \leftarrow x$ 
if  $y \neq z$ 
  then  $\text{key}[z] \leftarrow \text{key}[y]$ 
if  $\text{color}[y] = \text{BLACK}$ 
  then RB-DELETE-FIXUP( $T, x$ )
return  $y$ 
```

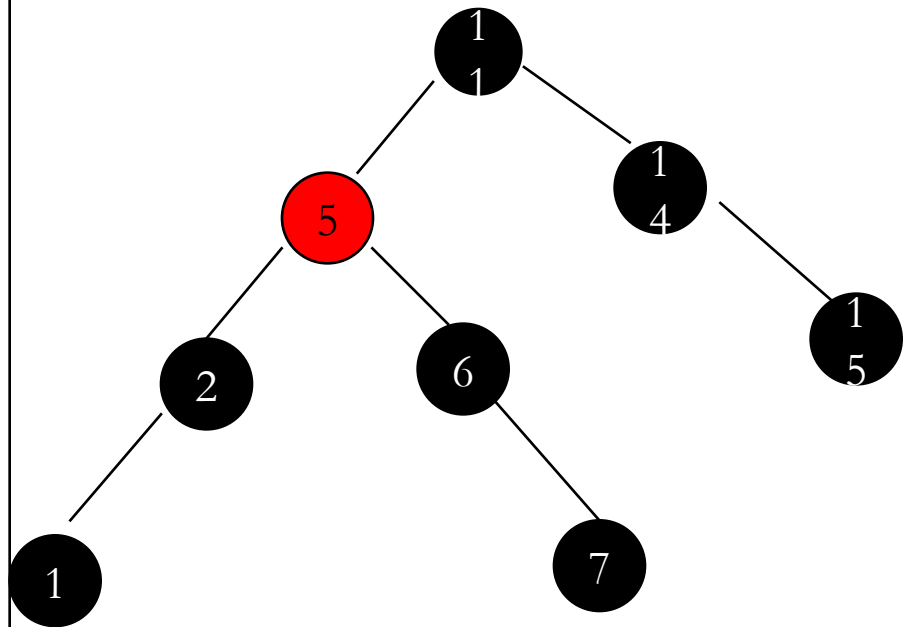


RB-DELETE( $T, z$ ), donde  
 $\text{key}[z] = 7$

Se cambian las llaves entre  
7 y 8, y se deja de color  
negro el nodo (que ahora  
tiene el valor 8)

## RB-DELETE( $T, z$ )

```
if left[z]=nil or right[z]=nil[T]
  then  $y \leftarrow z$ 
  else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
if left[y]  $\neq$  nil[T]
  then  $x \leftarrow \text{left}[y]$ 
  else  $x \leftarrow \text{right}[y]$ 
 $p[x] \leftarrow p[y]$ 
if  $p[y] = \text{nil}[T]$ 
  then  $\text{root}[T] \leftarrow x$ 
  else if  $y = \text{left}[p[y]]$ 
    then  $\text{left}[p[y]] \leftarrow x$ 
    else  $\text{right}[p[y]] \leftarrow x$ 
if  $y \neq z$ 
  then  $\text{key}[z] \leftarrow \text{key}[y]$ 
if  $\text{color}[y] = \text{BLACK}$ 
  then RB-DELETE-FIXUP( $T, x$ )
return  $y$ 
```



RB-DELETE( $T, z$ ), donde  
 $\text{key}[z]=1$

Es necesario un ajuste

RB-DELETE-FIXUP(T, x)

while  $x \neq \text{root}[T]$  and  $\text{color}[x] = \text{BLACK}$

do if  $x = \text{left}[p[x]]$

then  $w \leftarrow \text{right}[p[x]]$

if  $\text{color}[w] = \text{RED}$

then  $\text{color}[w] \leftarrow \text{BLACK}$

$\text{color}[p[x]] \leftarrow \text{RED}$

LEFT-ROTATE(T, p[x])

$w \leftarrow \text{right}[p[x]]$

if  $\text{color}[\text{left}[w]] = \text{BLACK}$  and  $\text{color}[\text{right}[w]] = \text{BLACK}$

then  $\text{color}[w] \leftarrow \text{RED}$

$x \leftarrow p[x]$

else if  $\text{color}[\text{right}[w]] = \text{BLACK}$

then  $\text{color}[\text{left}[w]] \leftarrow \text{BLACK}$

$\text{color}[w] \leftarrow \text{RED}$

RIGHT-ROTATE(T, w)

$w \leftarrow \text{right}[p[x]]$

$\text{color}[w] \leftarrow \text{color}[p[x]]$

$\text{color}[p[x]] \leftarrow \text{BLACK}$

$\text{color}[\text{right}[w]] \leftarrow \text{BLACK}$

LEFT-ROTATE(T, p[x])

$x \leftarrow \text{root}[T]$

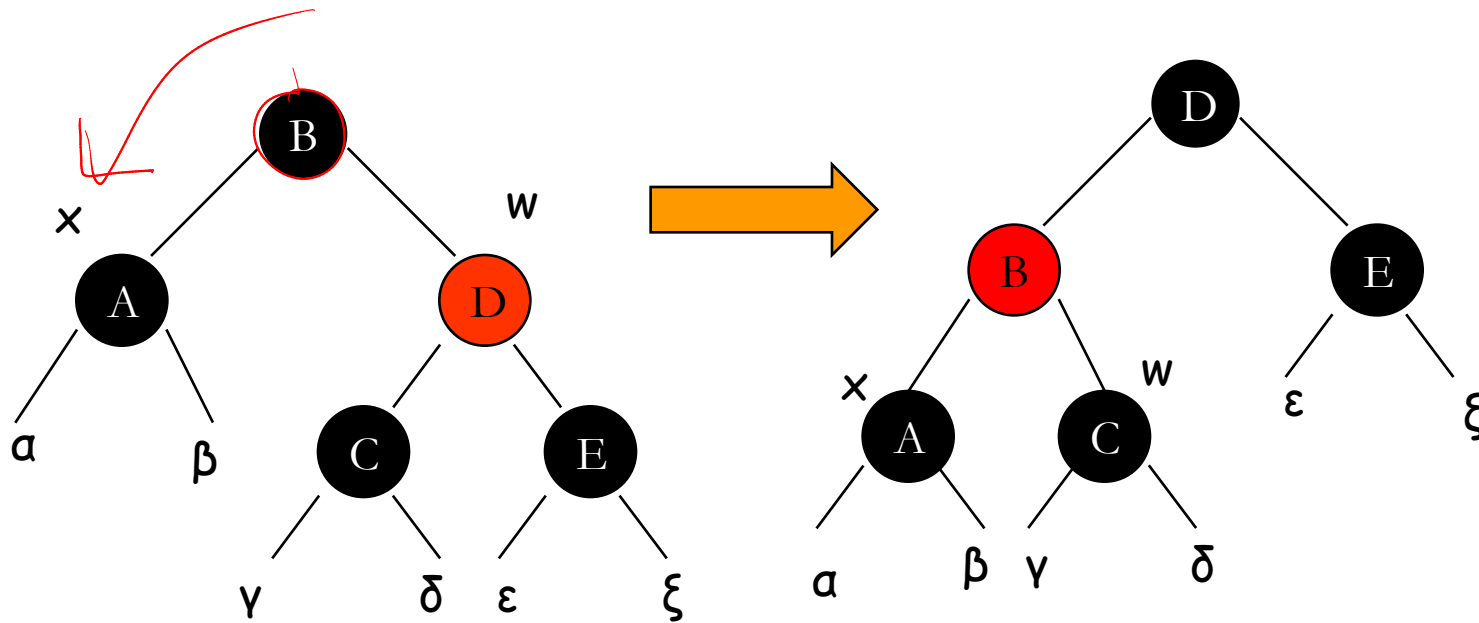
else #código simétrico intercambiando right y left

$\text{color}[x] \leftarrow \text{BLACK}$



# Árboles rojinegros

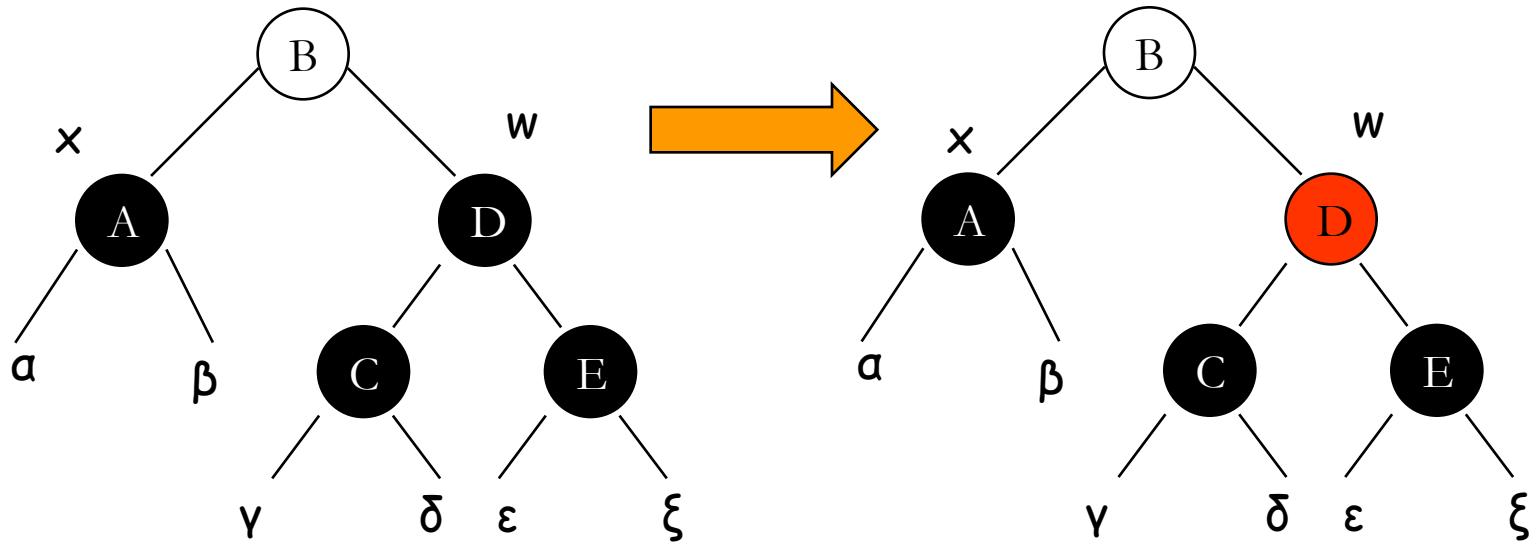
## Caso 1:



Se cambian los colores de B y D, y se realiza rotación a la izquierda  
Este cambio genera uno de los 3 casos siguientes

# Árboles rojinegros

## Caso 2:

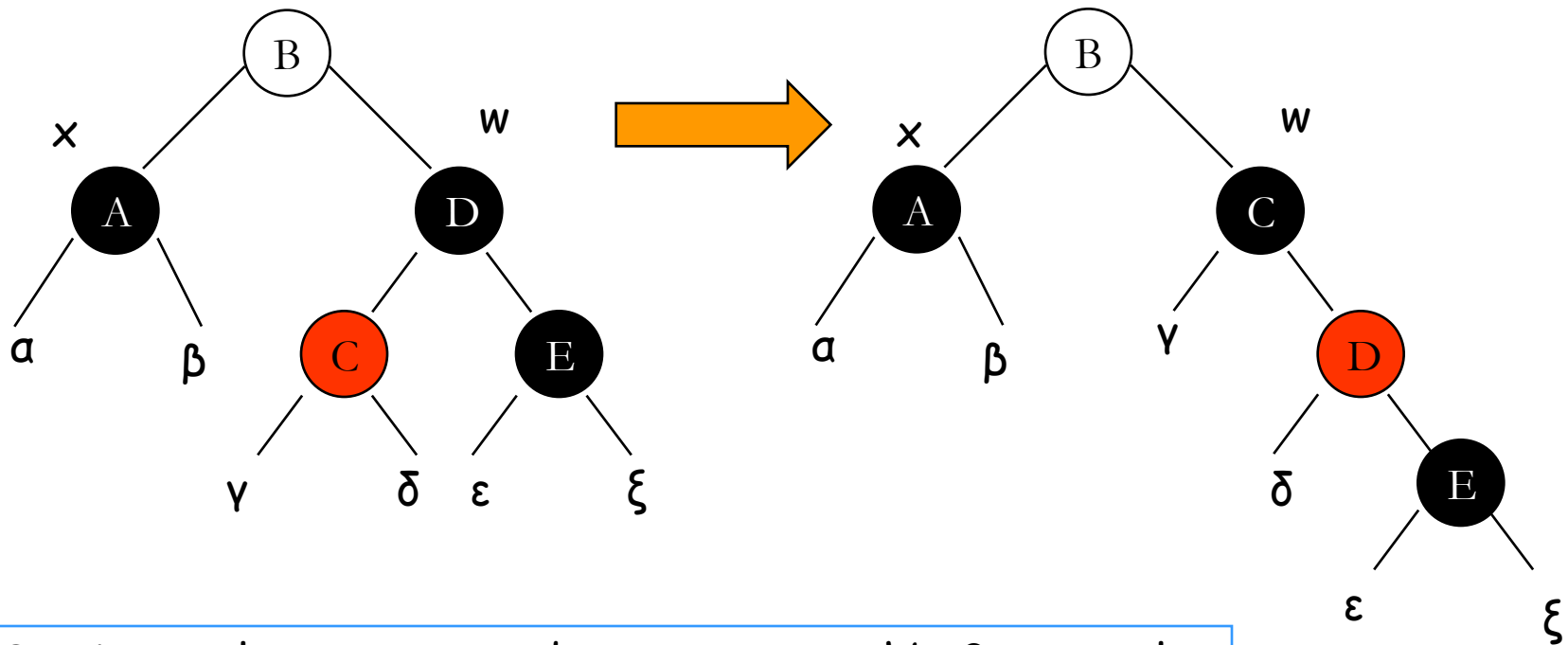


La estrategia del algoritmo consiste en considerar el nodo  $x$  como si tuviera un nodo extra, aumenta el conteo de negro en 2

En la rama de  $A$ , como se eliminó un nodo, el conteo de nodos negros se debe disminuir al lado derecho de  $B$ , para esto, se colorea de rojo el nodo con dos hijos negros

# Árboles rojinegros

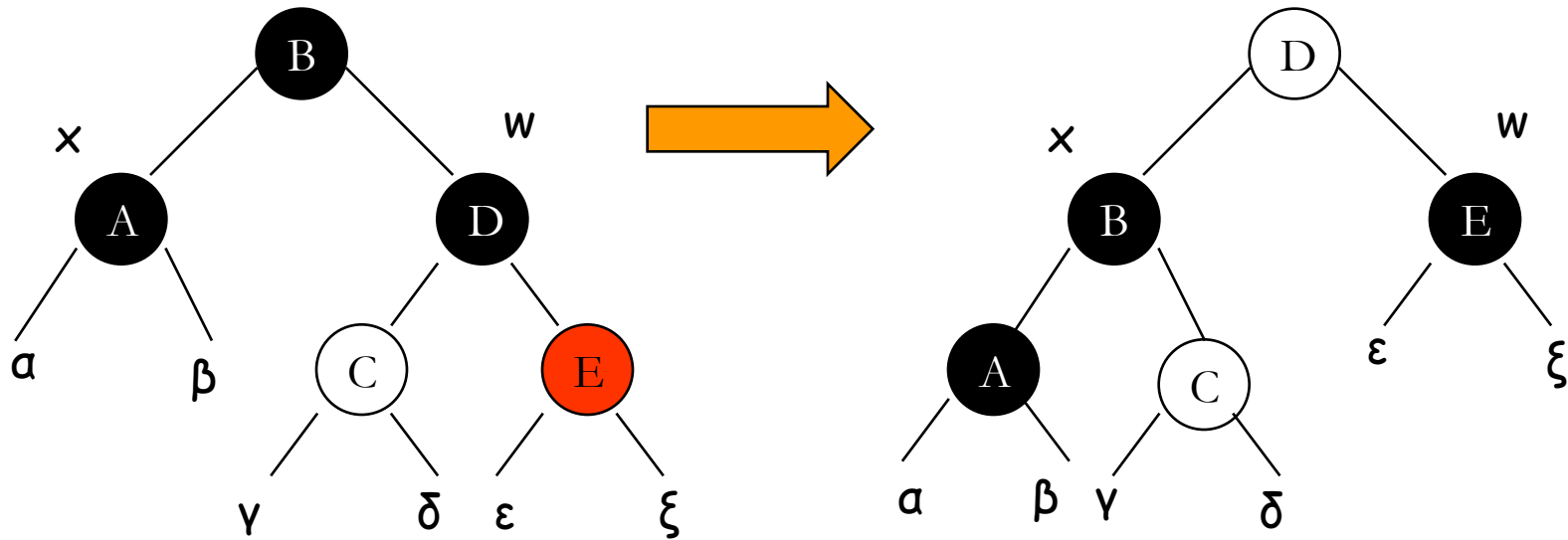
## Caso 3:



Se ajusta el conteo creando una rama con  $hb=2$ , antes, la rama B-D-C no tenía conteo 2

# Árboles rojinegros

## Caso 4:



Para completar el conteo por la rama de A, se rota a la izquierda B, el nodo E se cambia de color

**RB-DELETE-FIXUP**(T, x)

while  $x \neq \text{root}[T]$  and  $\text{color}[x] = \text{BLACK}$

do if  $x = \text{left}[p[x]]$

then  $w \leftarrow \text{right}[p[x]]$

if  $\text{color}[w] = \text{RED}$

then  $\text{color}[w] \leftarrow \text{BLACK}$

$\text{color}[p[x]] \leftarrow \text{RED}$

**LEFT-ROTATE**(T, p[x])

$w \leftarrow \text{right}[p[x]]$

if  $\text{color}[\text{left}[w]] = \text{BLACK}$  and  $\text{color}[\text{right}[w]] = \text{BLACK}$

then  $\text{color}[w] \leftarrow \text{RED}$

$x \leftarrow p[x]$

else if  $\text{color}[\text{right}[w]] = \text{BLACK}$

then  $\text{color}[\text{left}[w]] \leftarrow \text{BLACK}$

$\text{color}[w] \leftarrow \text{RED}$

**RIGHT-ROTATE**(T, w)

$w \leftarrow \text{right}[p[x]]$

$\text{color}[w] \leftarrow \text{color}[p[x]]$

$\text{color}[p[x]] \leftarrow \text{BLACK}$

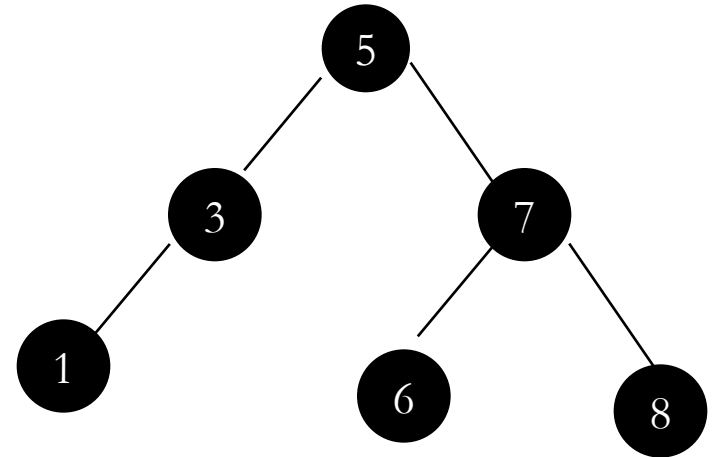
$\text{color}[\text{right}[w]] \leftarrow \text{BLACK}$

**LEFT-ROTATE**(T, p[x])

$x \leftarrow \text{root}[T]$

else #código simétrico intercambiando right y left

$\text{color}[x] \leftarrow \text{BLACK}$



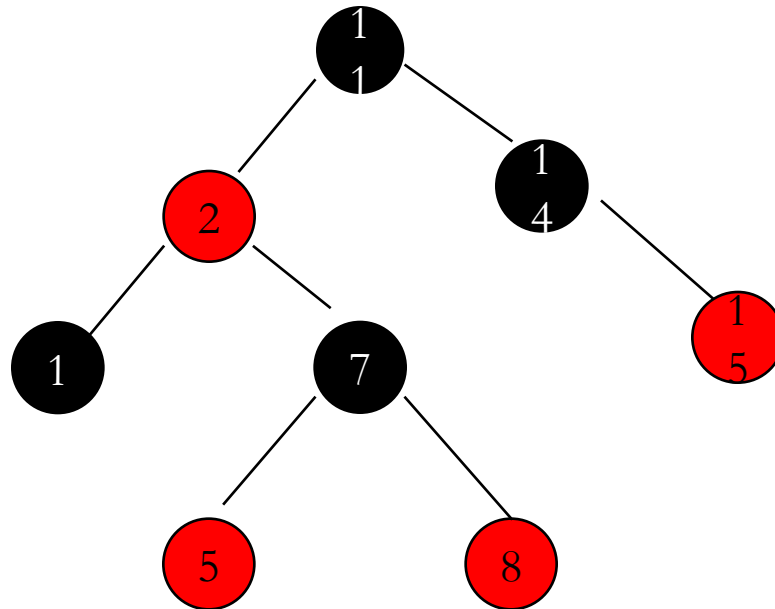
Siga el algoritmo  
**RB-DELETE**(T, 1)

# Árboles rojinegros

---

## Eliminación

Dado T, siga el algoritmo  
RB-DELETE(T, 7)

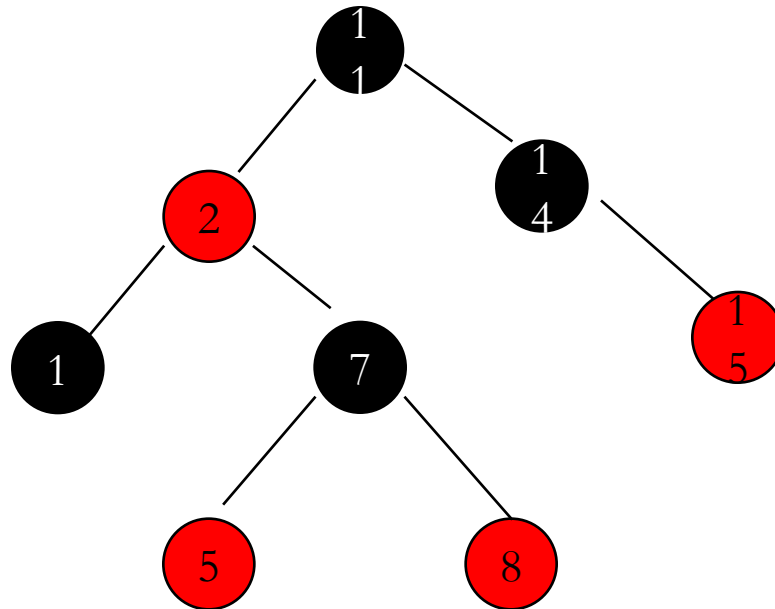


# Árboles rojinegros

---

## Eliminación

Dado T, siga el algoritmo  
RB-DELETE(T, 7)



# Árboles rojinegros

---

## Eliminación

Dado  $T$ , siga el algoritmo  $RB\text{-}DELETE(T, 4)$

