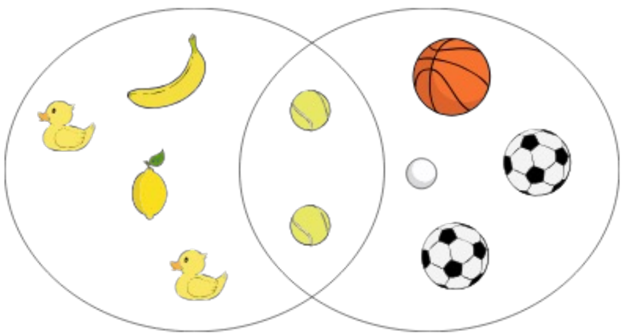
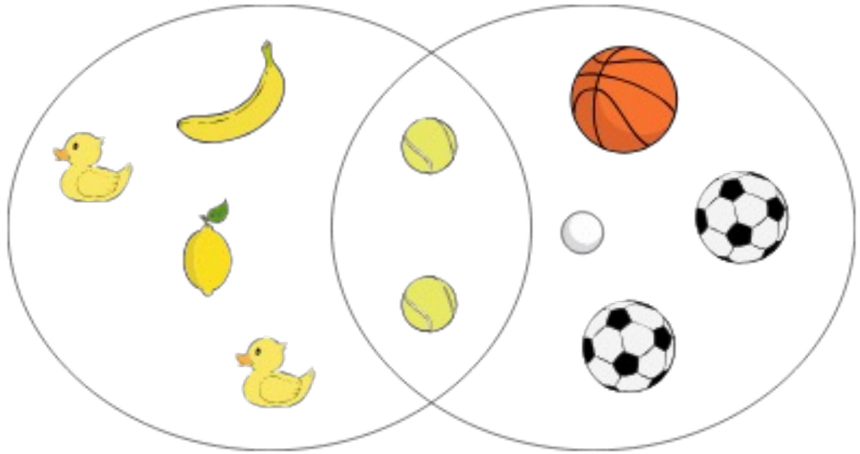
Diseño de Conjuntos y Diccionarios

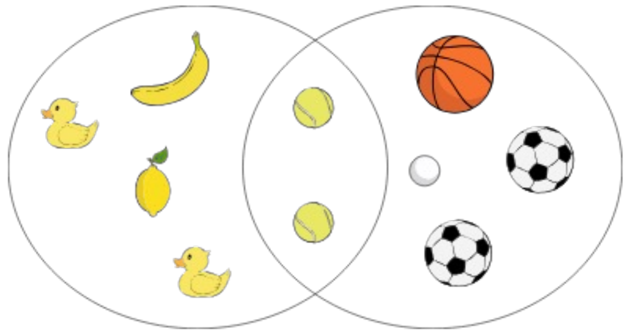
Balanceando los ABB con AVL



Diseño de Conjuntos y Diccionarios

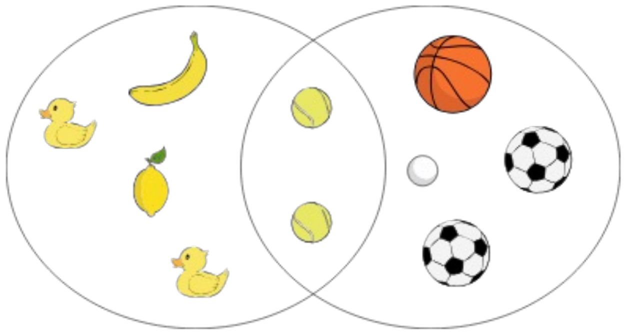


Definición del TAD Conjunto



```
TAD Conjunto<T> {
  obs elems: conj<T>

proc conjVacio(): Conjunto<T>
  asegura res.elems = {}
proc pertenece(in c: Conjunto<T>, in T e): bool
  asegura res = true <==> e in c.elems
proc agregar(input c: Conjunto<T>, in e: T)
  asegura c.elems = old(c).elems + {e}
proc sacar(inout c: Conjunto<T>, in e: T)
  asegura c.elems = old(c).elems - {e}
proc unir(inout c: Conjunto<T>, in c’: Conjunto<T>)
  asegura c.elems = old(c).elems + c’.elems
proc restar(inout c: Conjunto<T>, in c’: Conjunto<T>)
  asegura c.elems = old(c).elems - c’.elems
}
```

Definición del TAD Conjunto



```
TAD Conjunto<T> {
  obs elems: conj<T>

proc conjVacio(): Conjunto<T>
  asegura res.elems = {}
proc pertenece(in c: Conjunto<T>, in T e): bool
  asegura res = true <==> e in c.elems
proc agregar(input c: Conjunto<T>, in e: T)
  asegura c.elems = old(c).elems + {e}
proc sacar(inout c: Conjunto<T>, in e: T)
  asegura c.elems = old(c).elems - {e}
proc unir(inout c: Conjunto<T>, in c’: Conjunto<T>)
  asegura c.elems = old(c).elems + c’.elems
proc restar(inout c: Conjunto<T>, in c’: Conjunto<T>)
  asegura c.elems = old(c).elems - c’.elems
}
```

Y del TAD Diccionario

```
TAD Diccionario<K, V> {
  obs data: dict<K, V>

proc diccionarioVacio(): Diccionario<K, V>
  asegura res.data = {}
proc esta(in d: Diccionario<K, V>, in k: K): bool
  asegura res = true <==> k in d.data
proc definir(inout d: Diccionario<K, V>, in k: K, in v: V)
  asegura d.data = setKey(old(d).data, k, v)
proc obtener(in d: Diccionario<K, V>, in k: K): V
  requiere k in d.data
  asegura res = d.data[k]
proc borrar(inout d: Diccionario<K, V>, in k: K)
  requiere k in d.data
  asegura d.data == delKey(old(d).data, k)
}
```

Arboles Binarios de Búsqueda (ABB)

Que es un árbol binario de busqueda?

Es un árbol binario que satisfice la siguiente propiedad:

- Para todo nodo, los valores de su subarbol **izquierdo** son **menores** que el valor del nodo y los valores del subarbol **derecho** son **mayores**

O sea, un arbol es ABB si los elementos del subarbolos izquierdoo son menores a la raíz, y los elementos del subarbol derecho son mayores a la raíz.

esABB(a): esArbolBin(a) && esABBN(a.raiz)

esABBN(r) : r = null || (∀x) x in elementos(r.izq) => x<=r.dato && (∀x) x in elementos(r.der) => x>r.dato && esABBN(r.izq) && esABBN(r.der)

Invariante de Representación

El invariante de representación de la representación de Conjuntos con Árboles Binarios que son de Búsqueda sería:

```
pred InvRepABB (e: AB)

  {esABB(e)=TRUE}
```

esABB(a) = esArbolBin(a) && esABBN(a.raiz)
esABBN(r) = r = null || (∀x) x in elementos(r.izq) => x<=r.dato && (∀x) x in elementos(r.der) => x>r.dato && esABBN(r.izq) && esABBN(r.der)

Y la función de abstracción?

```
FuncAbs(a:AB): Conjunto c|
  c.elems = { n:N | n in elementos(a.raiz) }
```

elementos(r) = if r = null then {} else {r.dato} U elementos(r.izq) U

Algoritmos para ABB

- Vacio
- Búsqueda
- Inserción
- Eliminar

```
Nodo = Struct <dato: N, izq: Nodo, der: Nodo>
o opcionalmente...
Nodo = Struct <dato: N, izq: Nodo, der: Nodo, padre:Nodo>

Módulo AB implementa Conjunto {
  var raíz: Nodo
}
```

Representación de conjuntos y diccionarios a través de AVL

- Todas las representaciones vistas hasta ahora tienen al menos una operación de costo linear en función de la altura
  - Pero el problema que esta puede ser similar a la cantidad de elementos
- En muchos casos, eso puede ser inaceptable
- ¿Habrá estructuras más eficientes?

Introducción al balanceo

¿Qué altura tiene un árbol completo?

- Pero…no podemos pretender tener siempre árboles completos
- Quizás con alguna propiedad más débil…

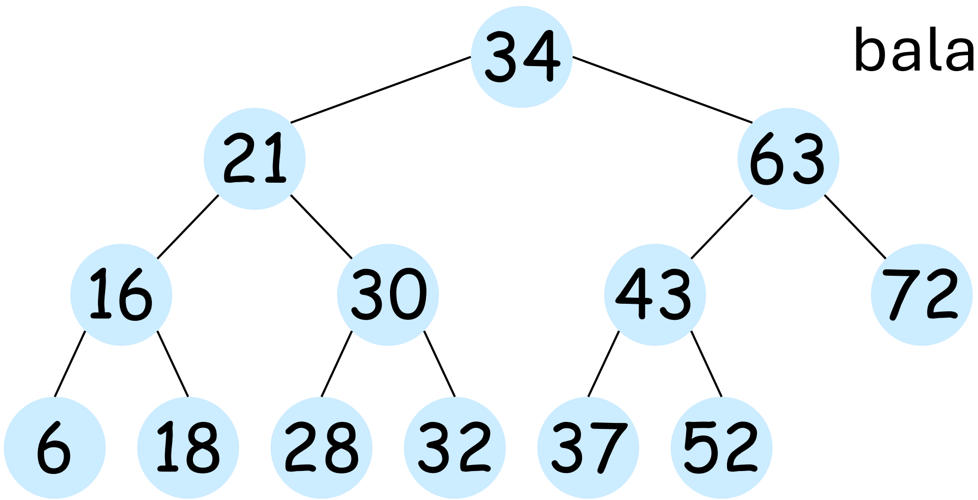
Noción intuitiva de balanceo

- Todas las ramas del árbol tienen “casi” la misma longitud
- Todos los nodos internos tienen “muchos” hijos

Caso ideal para un arbol *k*-ario

- Cada nodo tiene 0 o *k* hijos
- La longitud de dos ramas cualesquiera difiere a lo sumo en una unidad

balanceo perfecto



¡Las hojas son más del 50% de los nodos¡

- Teo: Un árbol binario perfectamente balanceado de *n* nodos tiene altura

$$\lfloor \lg_2 n \rfloor + 1$$

Nivel 1: 1 nodo
Nivel 2: 2 nodos
Nivel 3: 4 nodos
…
Nivel h: 2^{h-1} nodos

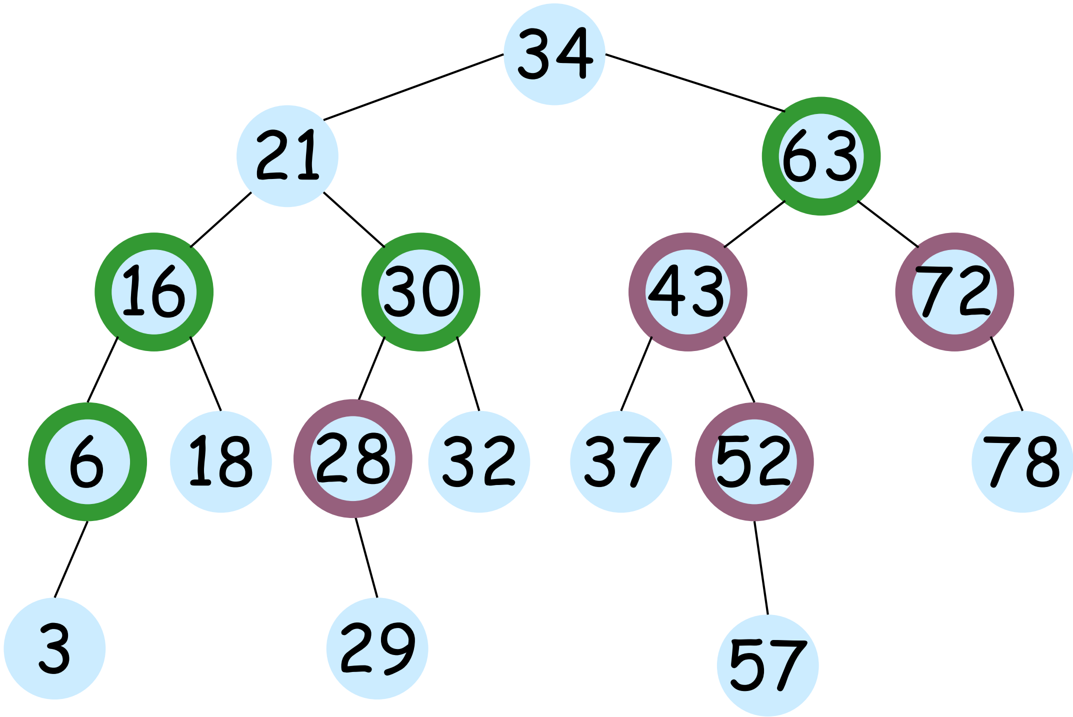
Si h= log n ➔ n/2 nodos

Balanceo en altura

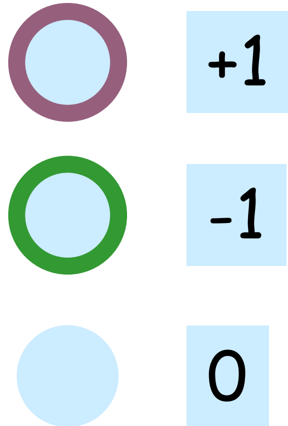
Un árbol se dice balanceado en altura si las alturas de los subárboles izquierdo y derecho de cada nodo difieren en a lo sumo una unidad

- Fueron propuestos en *1962*
- Se llaman árboles *AVL,* por sus creadores Гео́ргий Макси́мович Адельсо́н у Ве́льский у Евгений Михайлович Ландис
- También conocidos como Gueorgui Maksimovich Adelsón-Velski (1922-2014) y Yevgueni Mijáilovich Landis (1921-1997)

Factor de balanceo

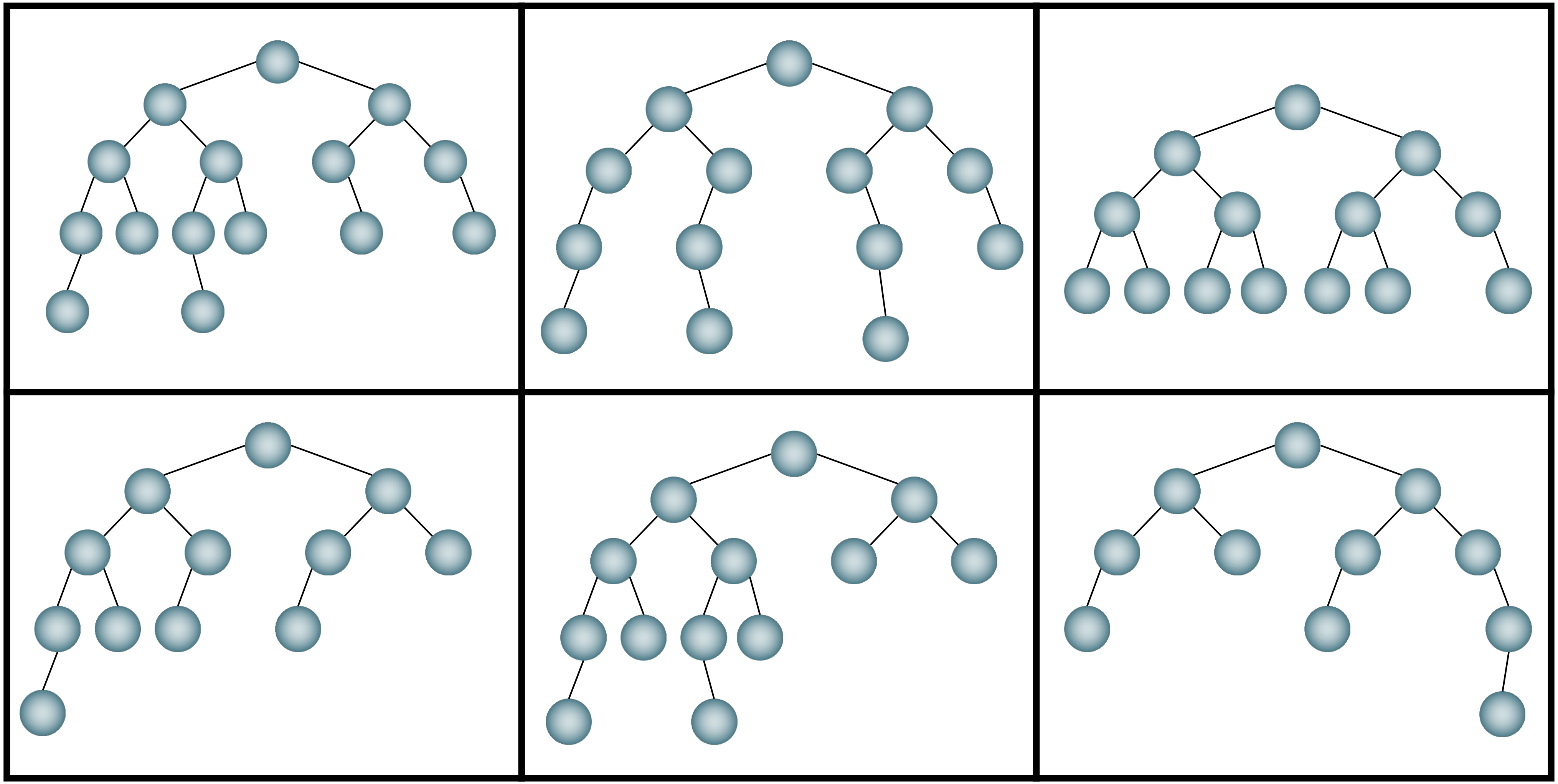


factor de balanceo (FDB):
Altura(Der) - altura(Izq)



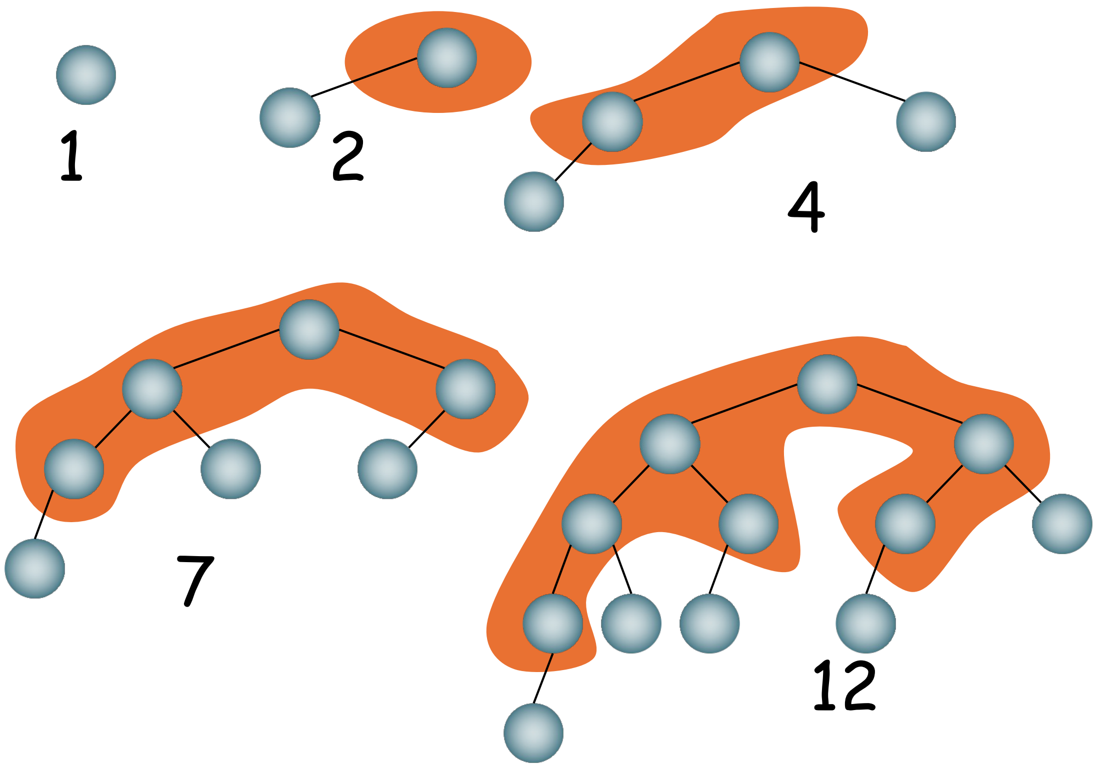
en un árbol balanceado en altura
$|FDB| \leq 1$, para cada nodo

Árboles AVL?



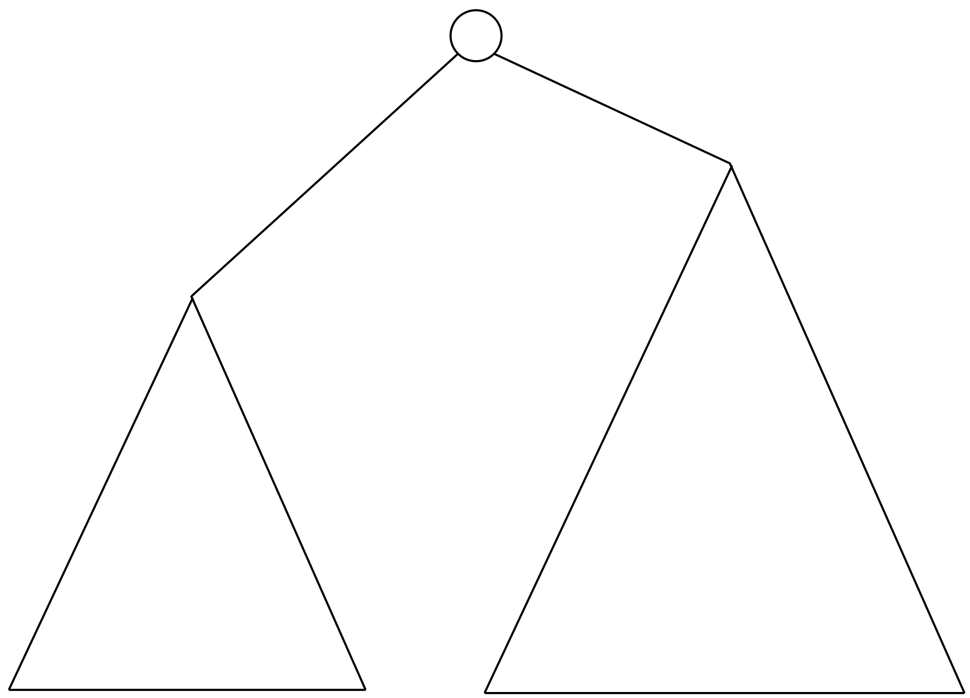
árboles de Fibonacci

- árboles AVL con el mínimo numero de nodos (dada la altura)



h	F_h	AVL_h
0	0	0
1	1	1
2	1	2
3	2	4
4	3	7
5	5	12
6	8	20
7	13	33

árboles de Fibonacci/2



AVL_i AVL_{i+1}

árboles de Fibonacci

árboles balanceados de altura *i* con mínimo numero de nodos

AVL_{i+2}

<u>Relaciones</u>

$AVL_{i+2} = AVL_i + AVL_{i+1} + 1$

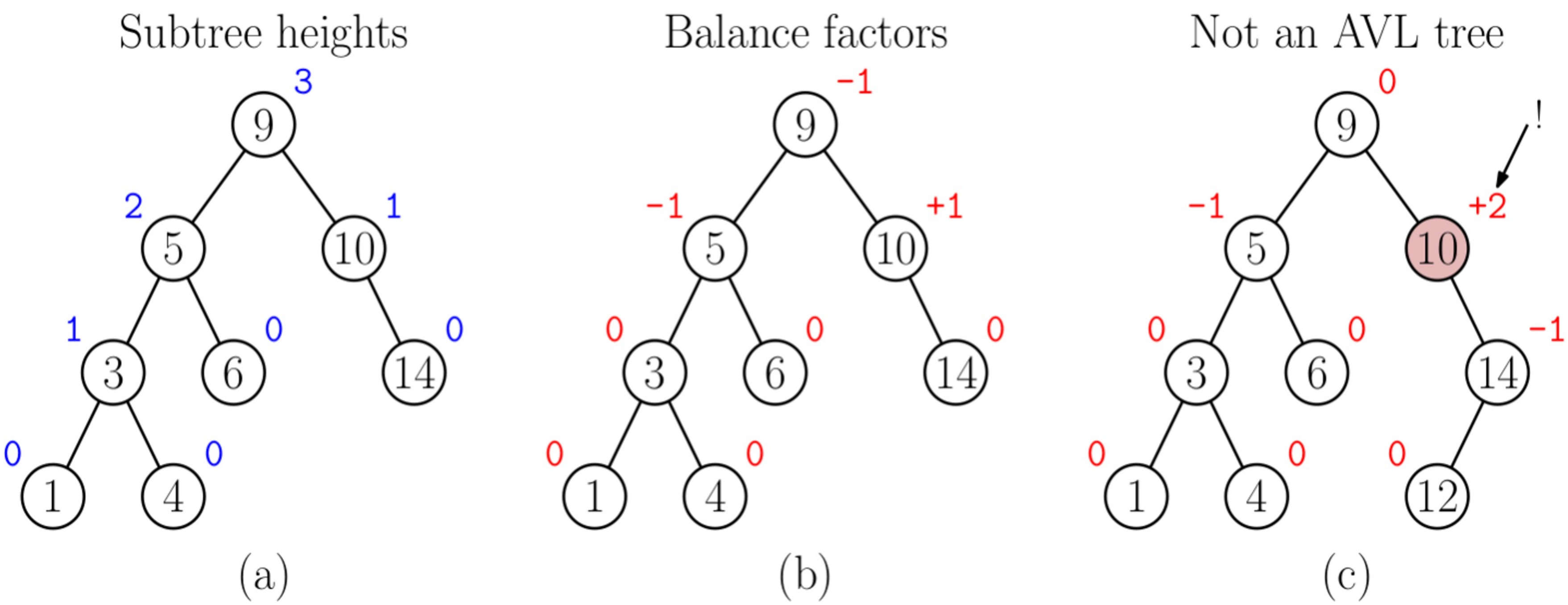
$F_{i+2} = F_i + F_{i+1}$

$AVL_i = F_{i+2} - 1$

árboles de Fibonacci/3

- un árbol de Fibonacci tiene todos los factores de balanceo de sus nodos internos ± 1
  - Es el árbol balanceado más cercano a la condición de no-balanceo
- un árbol de Fibonacci con n nodos tiene altura $< 1.44 \lg(n + 2) - 0.328$
  - demostrado por Adel'son-Vel'skii & Landis
  - $\Rightarrow$ un AVL de n nodos tiene altura $\Theta(\lg n)$

AVL Examples



Implementación de AVL

```
Módulo ArbolAVL implementa conjunto = {
   var raíz: NodoAVL
}

Struct NodoAVL x {
    x.izq,
    x.der,
    x.dato
    x.altura
}
```

`new NodoAvl(x)`: crea un nodo AVL con x como dato

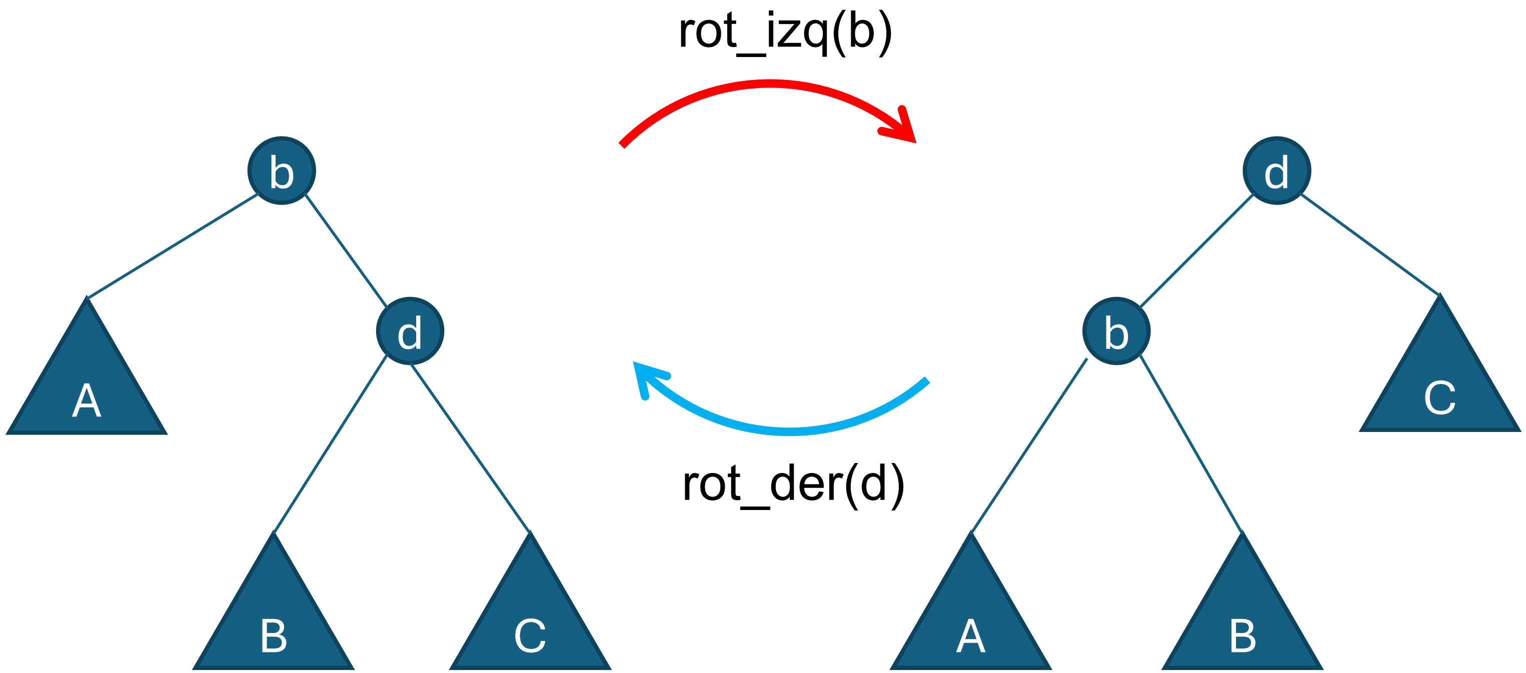
Funciones útiles

```
impl aux altura(p:NodoAVL): int
   if p == null
      return -1
   else
      return p.altura


impl aux actualizarAltura(p:NodoAVL)
    p. altura = 1 + max(altura(p.izq), altura(p.der))


impl aux factorBalanceo(p:NodoAVL): int
   return (altura(p.der) - altura(p.izq))
```

inserción en AVL

1. Insertar el nuevo nodo como en un ABB “clásico”
   - el nuevo nodo es una hoja
2. Recalcular los factores de balanceo que cambiaron por la inserción
   - sólo en la rama en la que ocurrió la inserción (los otros factores no pueden cambiar!), de abajo hacia arriba
3. Si en la rama aparece un factor de balanceo de ±2 hay que rebalancear
   - A través de “rotaciones”

Rotaciones

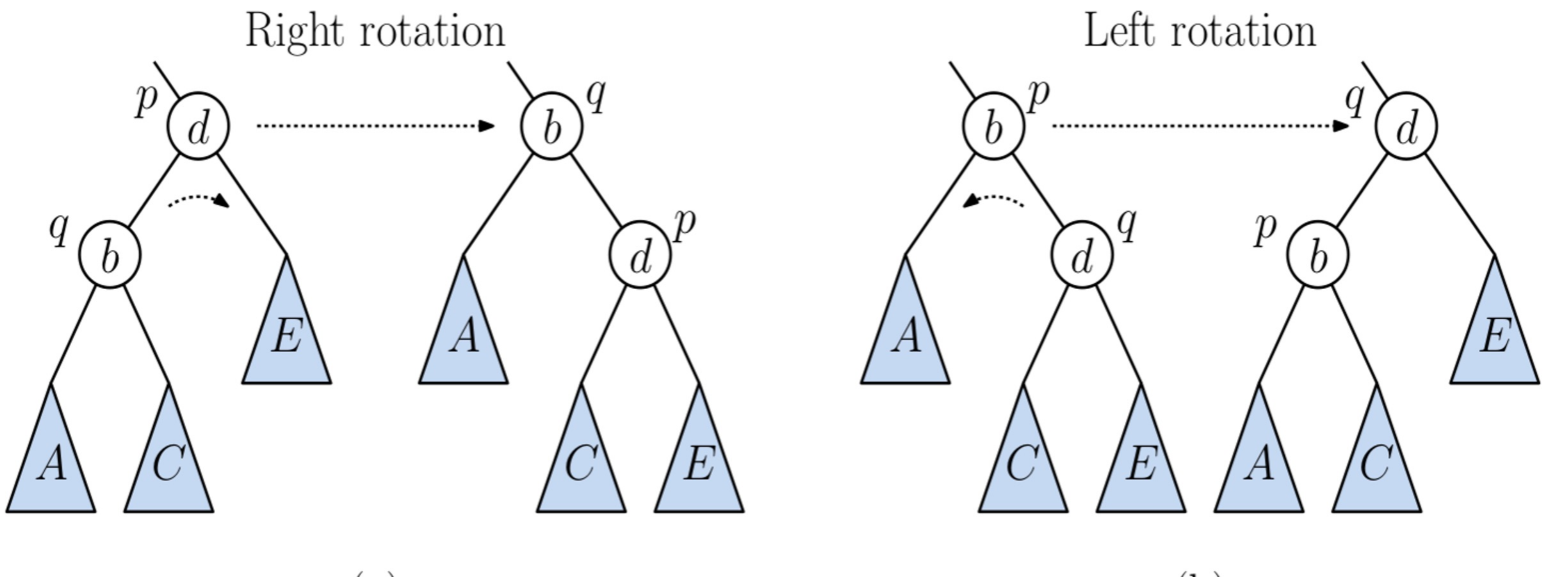


AbBdC

AbBdC

Rotaciones

Rotación: modificación local que modifica las alturas de los subárboles, pero conserva el orden

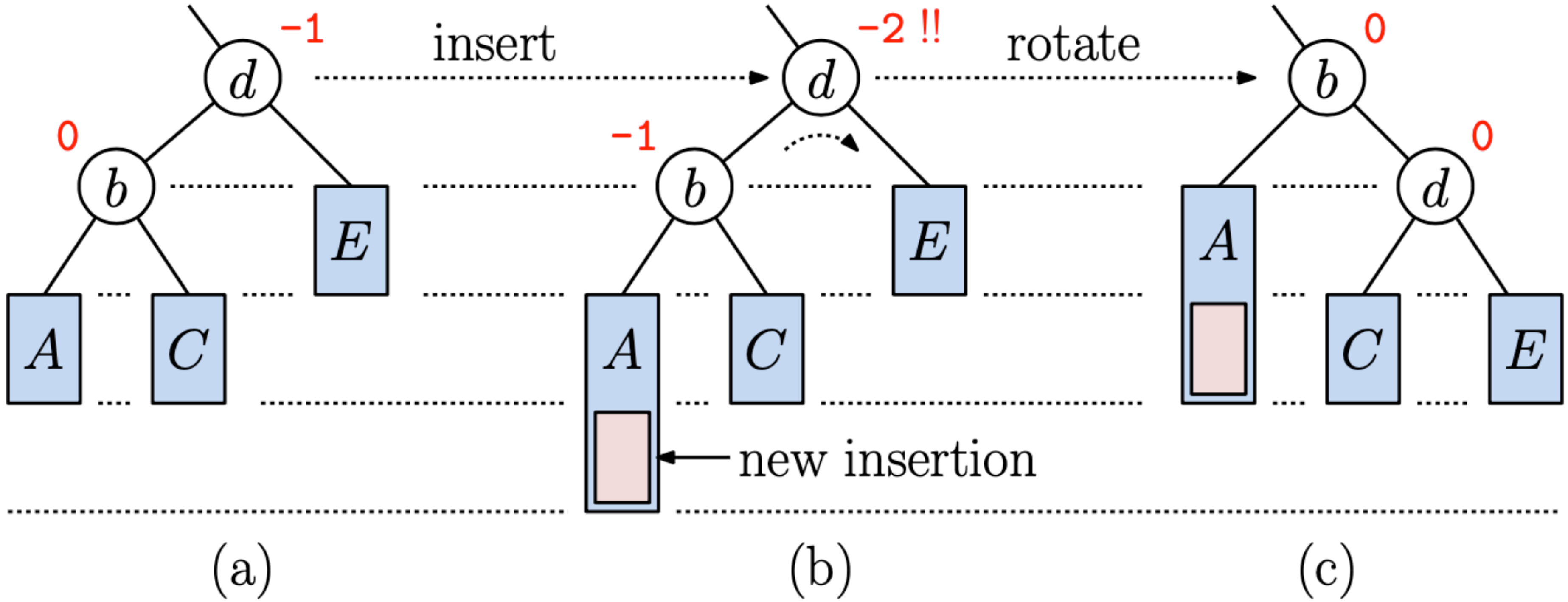


Una rotación no siempre es suficiente para rectificar un nodo que está desequilibrado (por ejemplo, el subárbol C)

Rotaciones en los AVL (Casos)

- II: inserción en el subárbol **izquierdo** de un hijo **izquierdo** (del nodo que se desbalancea)
  - Rotar Derecha
- DD: inserción en el subárbol **derecho** de un hijo **derecho** (del nodo que se desbalancea)
  - Rotar Izquierda

Insertar con rotación simple (II)



Notar que ademas de rotar temenos que llamar a actualizarAltura para b y d

Rotación doble

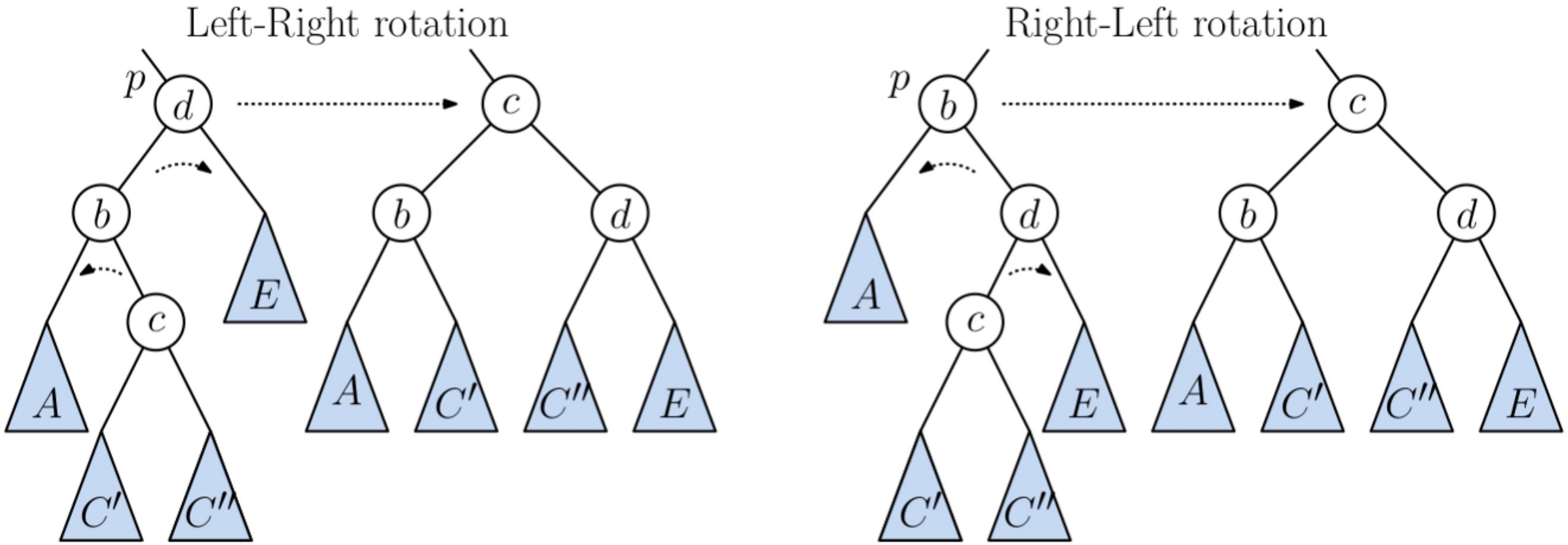
Una sola rotación no siempre es suficiente para rectificar un nodo que está desequilibrado. Por ejemplo: la rotación simple no altera la altura del subárbol C.

Doble rotación: combinación de dos rotaciones.

rotarIzquierdaDerecha(p): una rotación a la izquierda de p.izq seguida de una rotación a la derecha hacia p.

rotarDerechaIzquierda(p): una rotación a la derecha de p.der seguida de una rotación hacia la izquierda hacia p.

Rotación doble



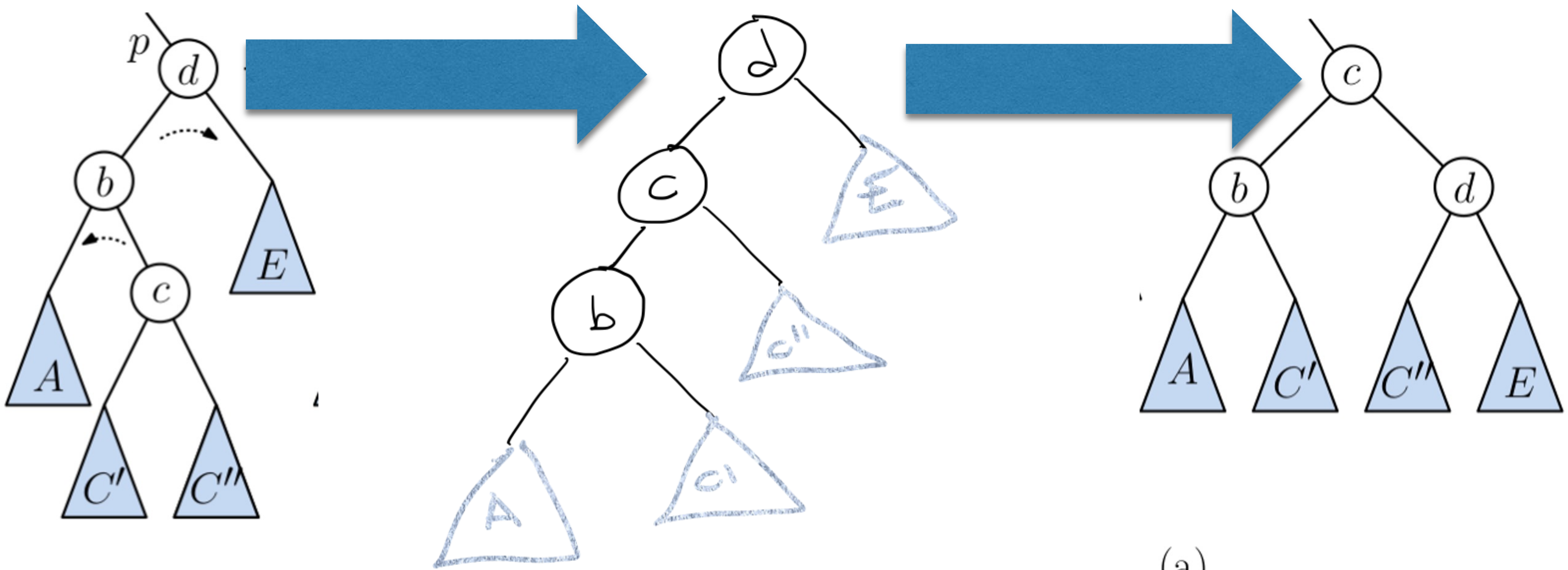
(a)

(b)

rotaciónIzquierdaDerecha(p):una rotación a la izquierda hacia la p.izq seguida de una rotación a la derecha hacia p.

rotaciónDerechaIzquierda(p): una rotación a la derecha hacia la p.der seguida de una rotación hacia la izquierda hacia p.

Rotación doble



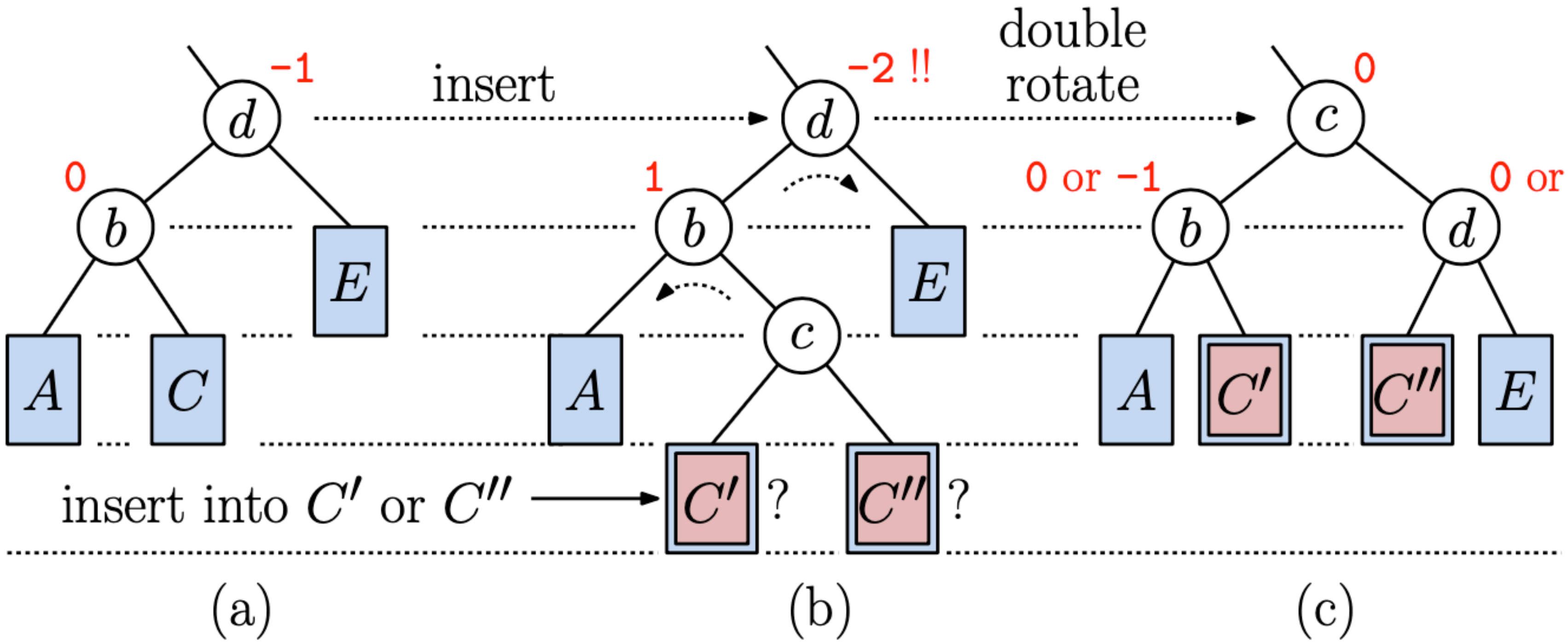
(a)

rotaciónIzquiedaDerecha(p): una rotación a la izquierda de p.izq seguida de una rotación a la derecha hacia p.

Rotaciones en los AVL (Casos)

- II: inserción en el subárbol **izquierdo** de un hijo **izquierdo** (del nodo que se desbalancea)
  - Rotar Derecha
- DD: inserción en el subárbol **derecho** de un hijo **derecho** (del nodo que se desbalancea)
  - Rotar Izquierda
- DI: inserción en el subárbol **derecho** de un hijo **izquierdo** (del nodo que se desbalancea)
  - Rotar Izquierda-Derecha
- ID: inserción en el subárbol **izquierdo** de un hijo **derecho** (del nodo que se desbalancea)
  - Rotar Derecha-Izquierda

Insertar con rotación doble (DI)



Notar que ademas de rotar temenos que llamar a actualizarAltura para b y d (y c)

Insertar doble

```
NodoAVL insertAVL(p:NodoAVL, x:int) {
 if (p == null)
    var p = new NodoAVL(x)
 else if (x < p.dato)
     p.izq = insertAVL(p.izq, x)
 else if (x > p.key)
     // x is larger - insert right
    p.der = insertAVL(p.der, x)
 else error 'Ya estaba el dato'
 return rebalance(p)
 }
```

Nota: el rebalanceo se invoca para todos los nodos de la rama desde p hasta la raíz

Rotaciones en los AVL (Como los detectamos?

- II: inserción en el subárbol **izquierdo** de un hijo **izquierdo** (del nodo que se desbalancea)
  - FB(n) <-1 y FB(n.izq)<=0
- DD: inserción en el subárbol **derecho** de un hijo **derecho** (del nodo que se desbalancea)
  - FB(n) >1 y FB(n.der)>=0
- DI: inserción en el subárbol **derecho** de un hijo **izquierdo** (del nodo que se desbalancea)
  - FB(n) <-1 y FB(n.izq)>0
- ID: inserción en el subárbol **izquierdo** de un hijo **derecho** (del nodo que se desbalancea)
  - FB(n) >1 y FB(n.der)<0

Rebalanceo

```
rebalancear(NodoAVL nodo): NodoAVL
   actualizarAltura(nodo);
   var fbd = balanceFactor(nodo);
    if (fbd < -1 && balanceFactor(nodo.izq) <= 0) // II
        nodo = rotacionDerecha(nodo);
    if (fbd > 1 && balanceFactor(nodo.der) >= 0)    // DD
           nodo = rotacionIzquierda(nodo);
    if (fbd < -1 && balanceFactor(nodo.izq) > 0)  // DI
         nodo = rotacionzquierdaDerecha(nodo);
    if (fbd > 1 && balanceFactor(nodo.der) < 0)   // ID
         nodo = rotacionDerechaIzquierda(nodo));
     actualizarAltura(nodo);
     return nodo;
}
```

inserción en AVL

1. Insertar el nuevo nodo como en un ABB “clásico”
   - el nuevo nodo es una hoja
2. Recalcular los factores de balanceo que cambiaron por la inserción
   - sólo en la rama en la que ocurrió la inserción (los otros factores no pueden cambiar!), de abajo hacia arriba
3. Si en la rama aparece un factor de balanceo de ±2 hay que rebalancear
   - A través de “rotaciones”

inserción en los AVL/costo

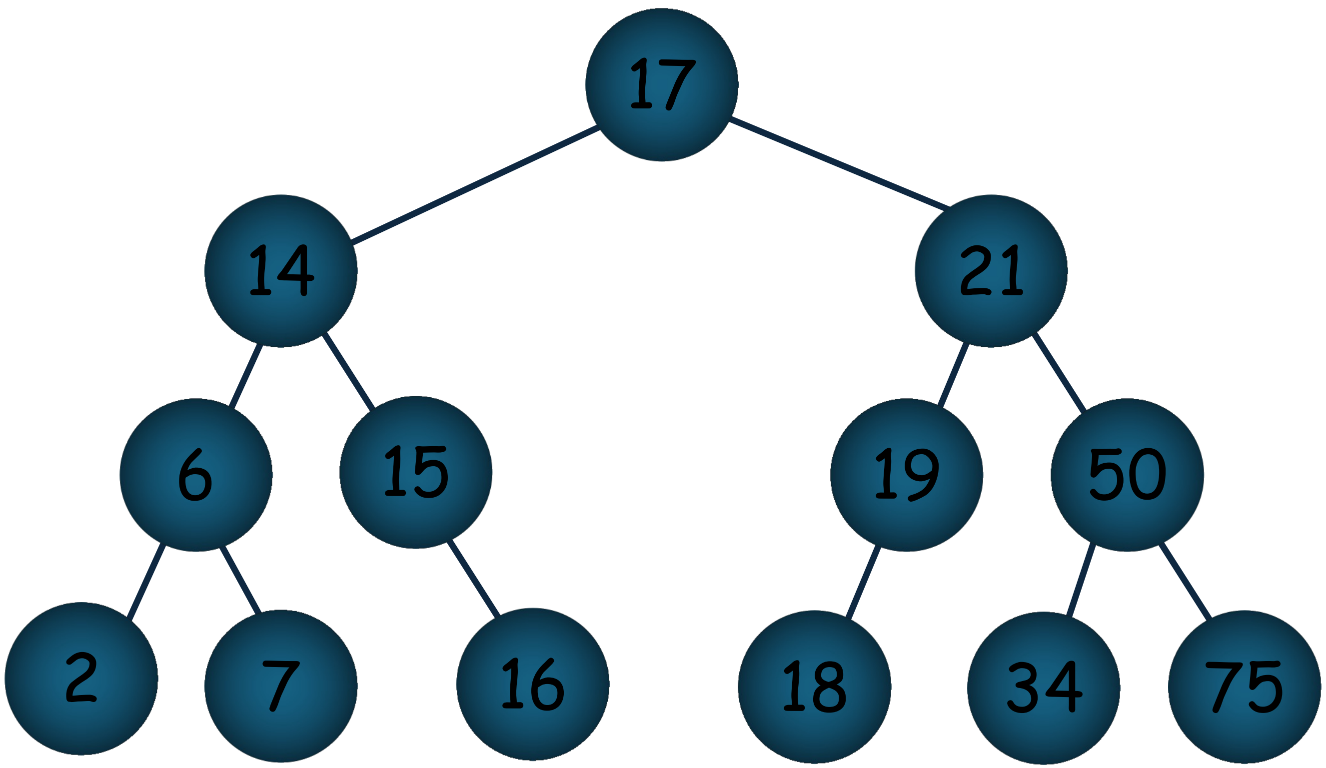
- paso 1: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 2: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 3: $O(1)$ (se hace una o dos rotaciones por inserción)

En total: $\Theta(\lg n)$

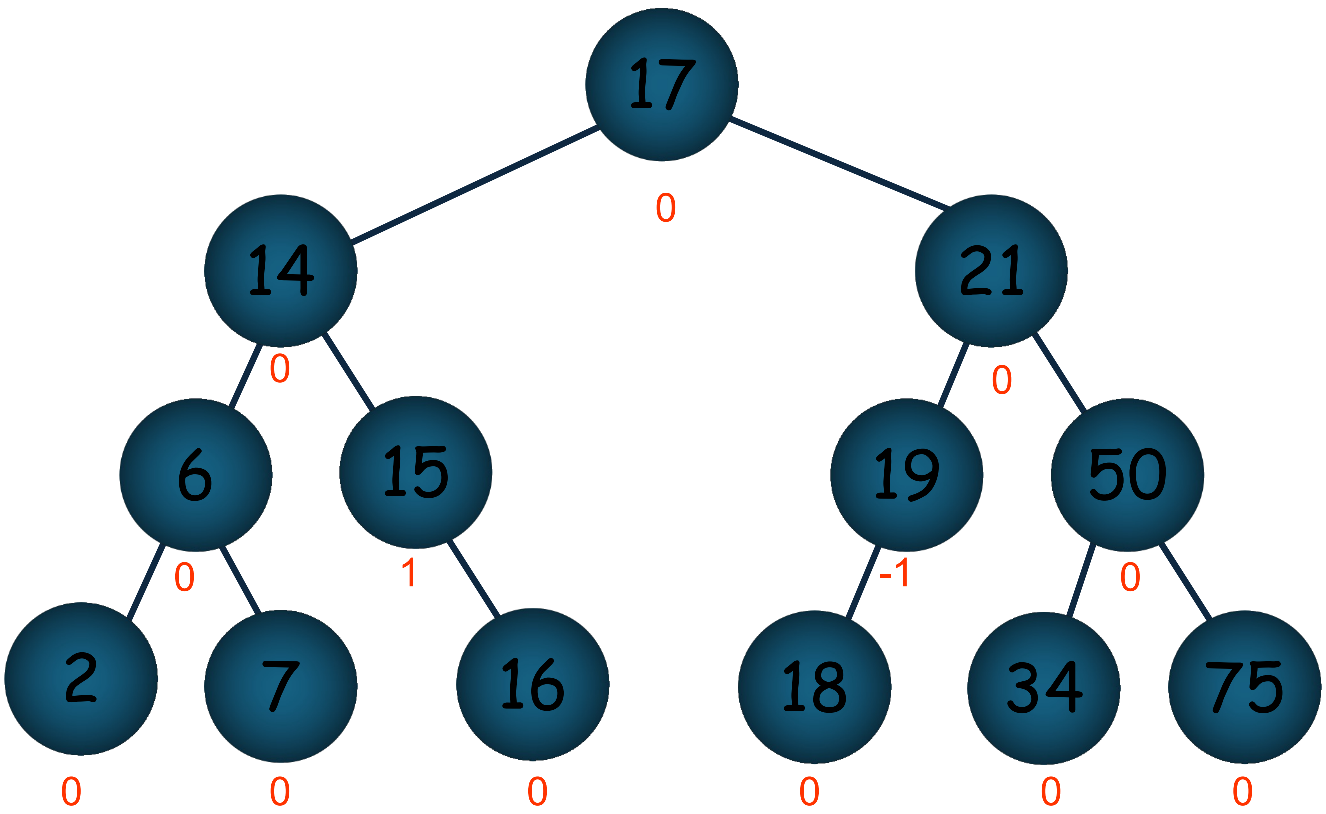
borrado en los AVL

1. Borrar el nodo como en un ABB “clásico”
2. recalcular los factores de balanceo que cambiaron por el borrado
   - sólo en la rama en que ocurrió el borrado, de abajo hacia arriba
3. <u>para cada nodo</u> con factor de balanceo ±2 hay que hacer una rotación simple o doble
   - O(lg *n)* rotaciones en el caso peor

borrado en los AVL

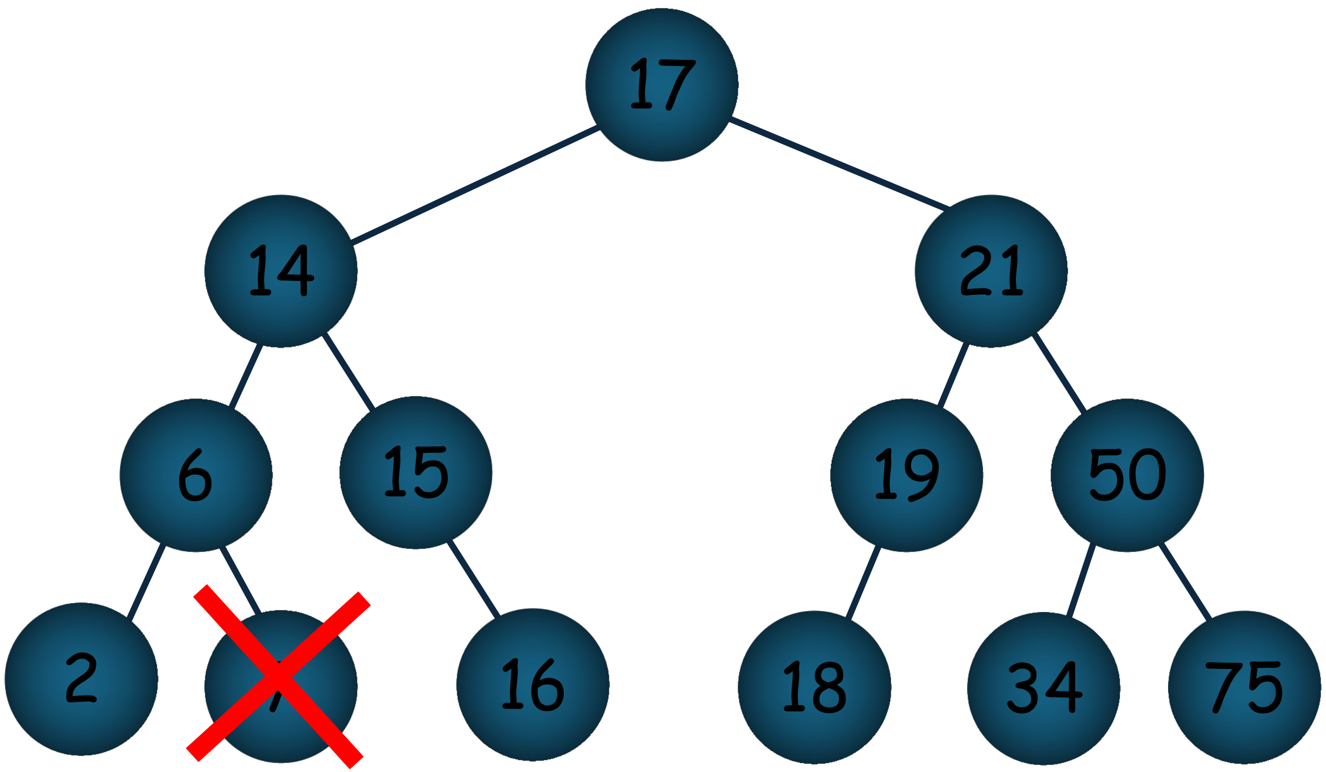


borrado en los AVL

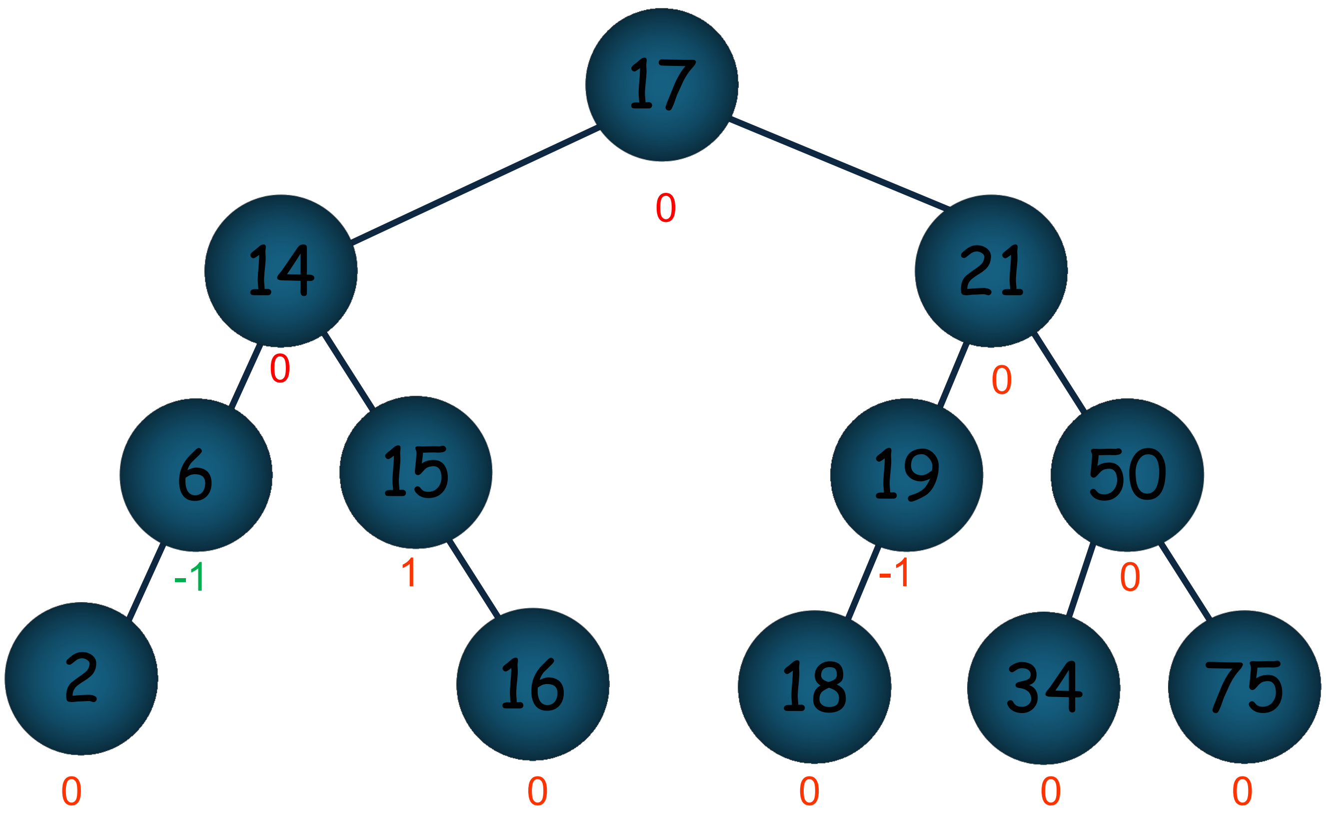


borrado en los AVL

Borrar 7

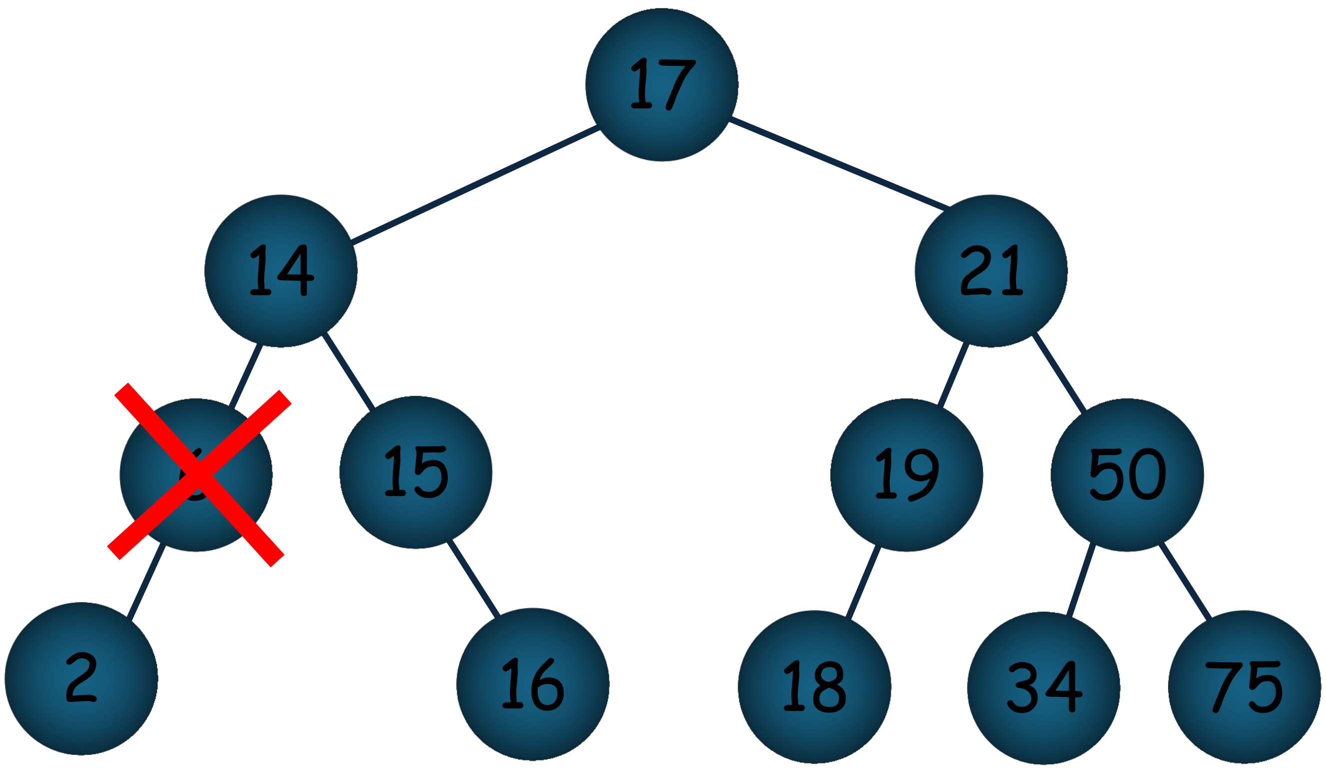


borrado en los AVL

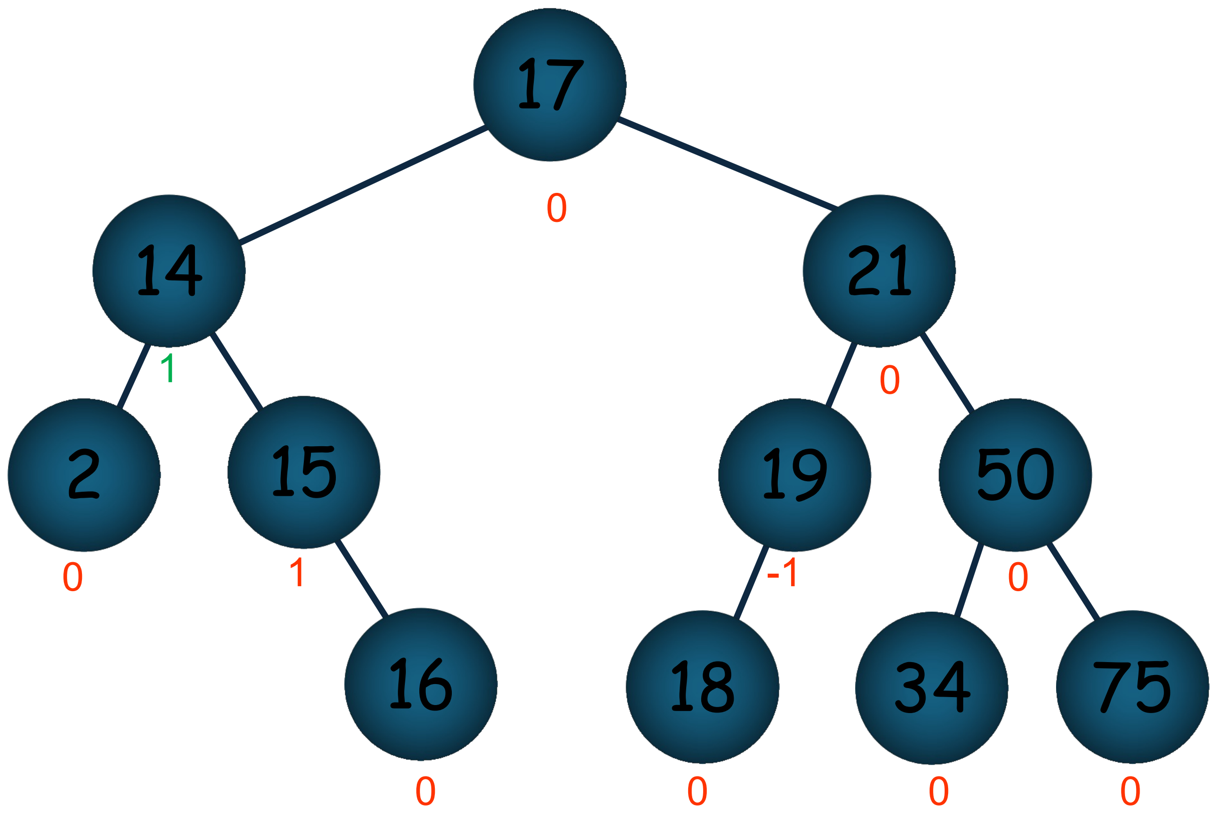


borrado en los AVL

Borrar 6

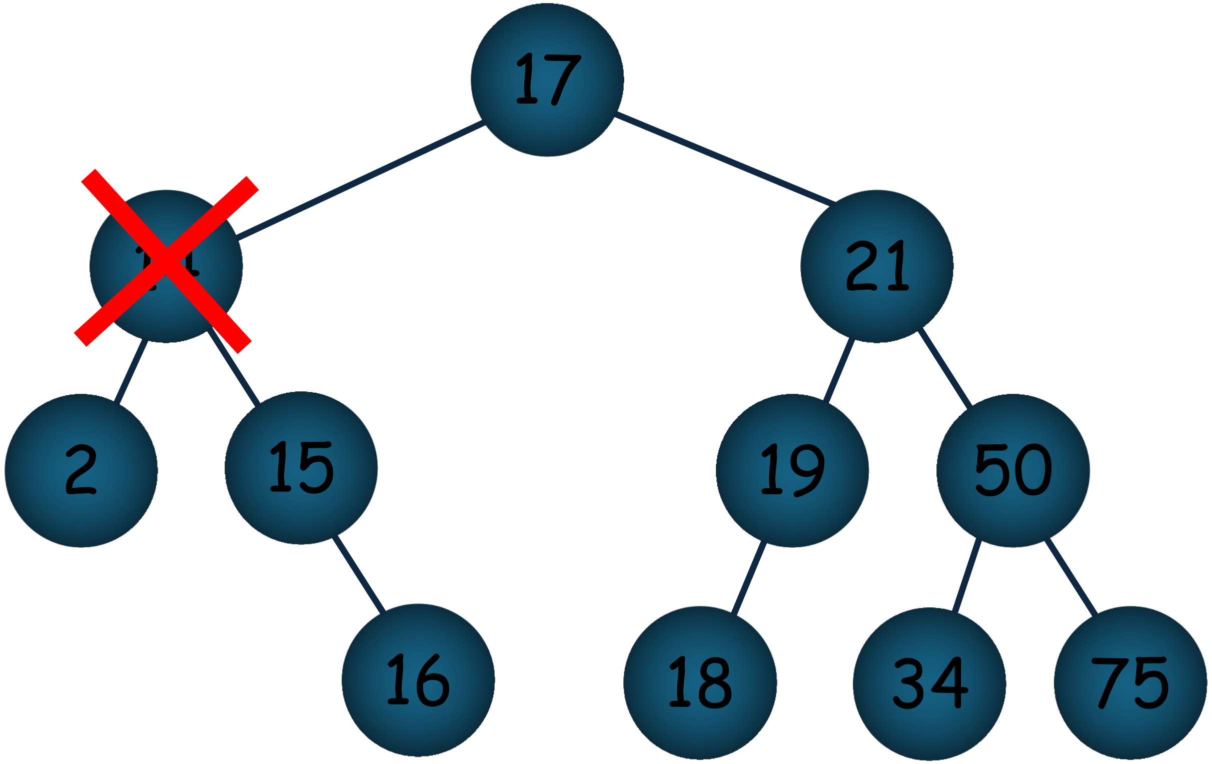


borrado en los AVL

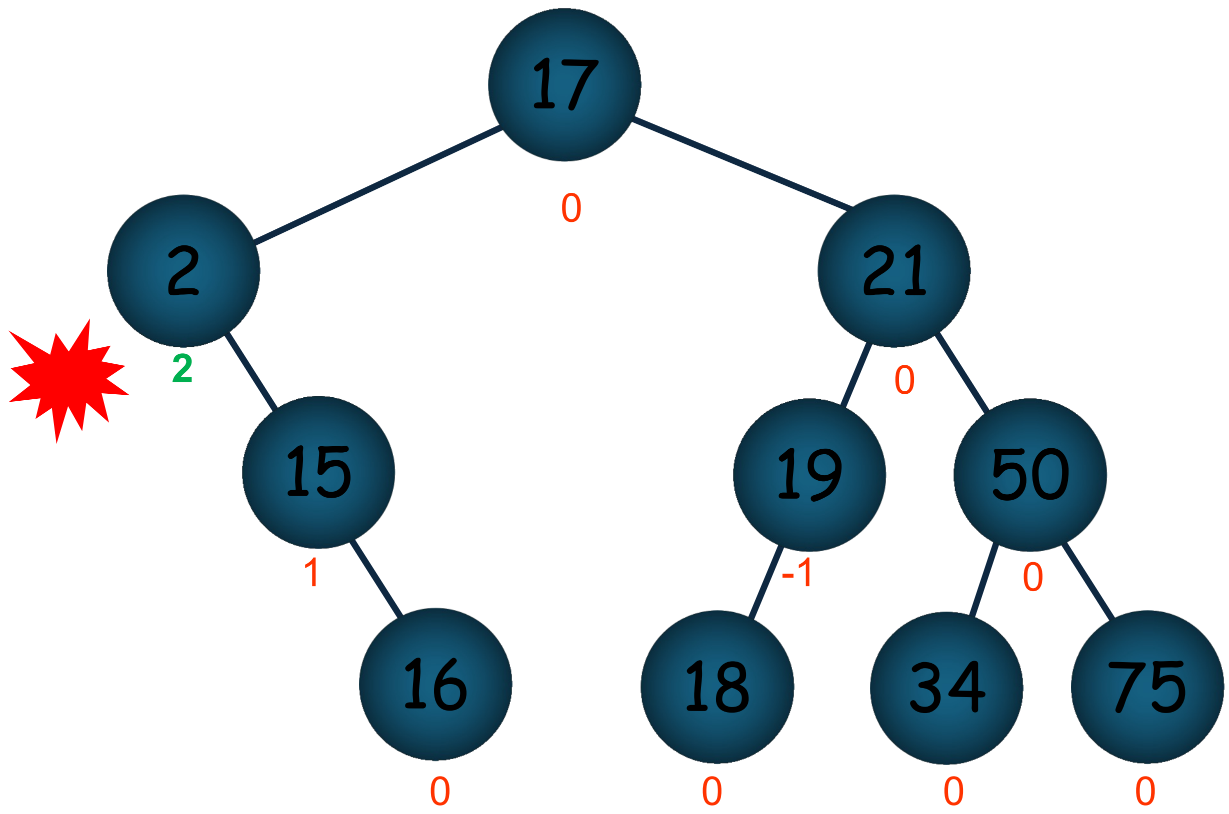


borrado en los AVL

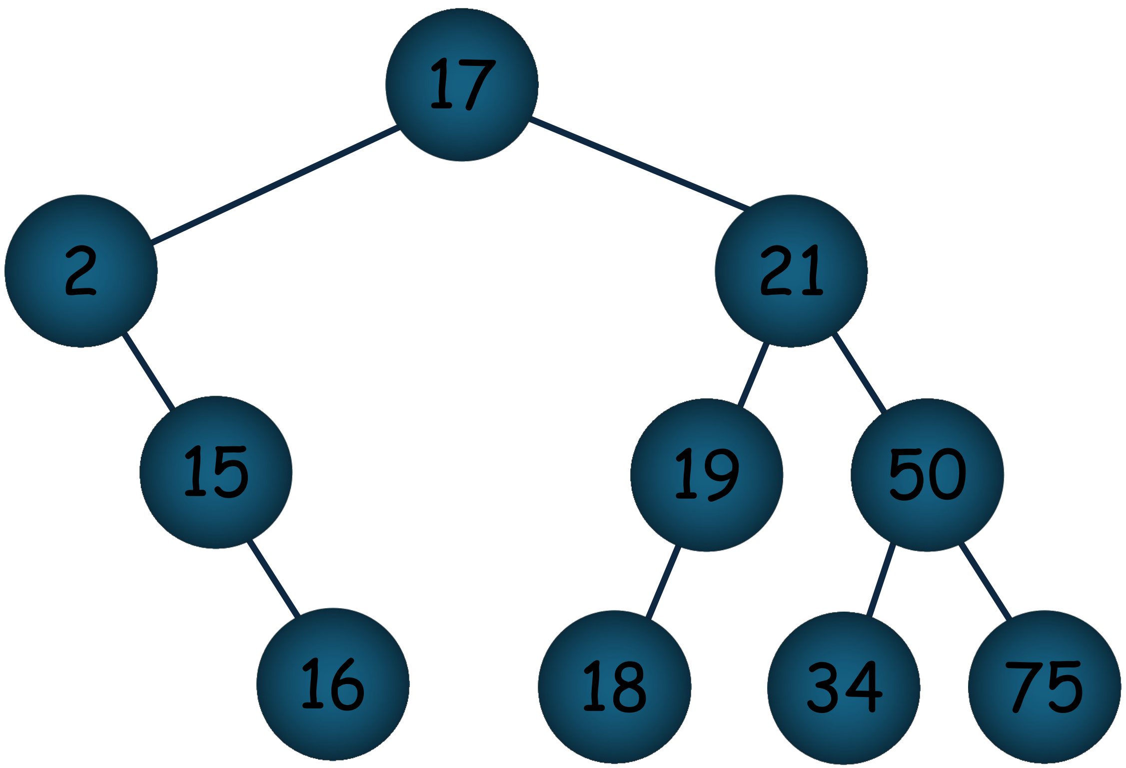
Borrar 14



borrado en los AVL

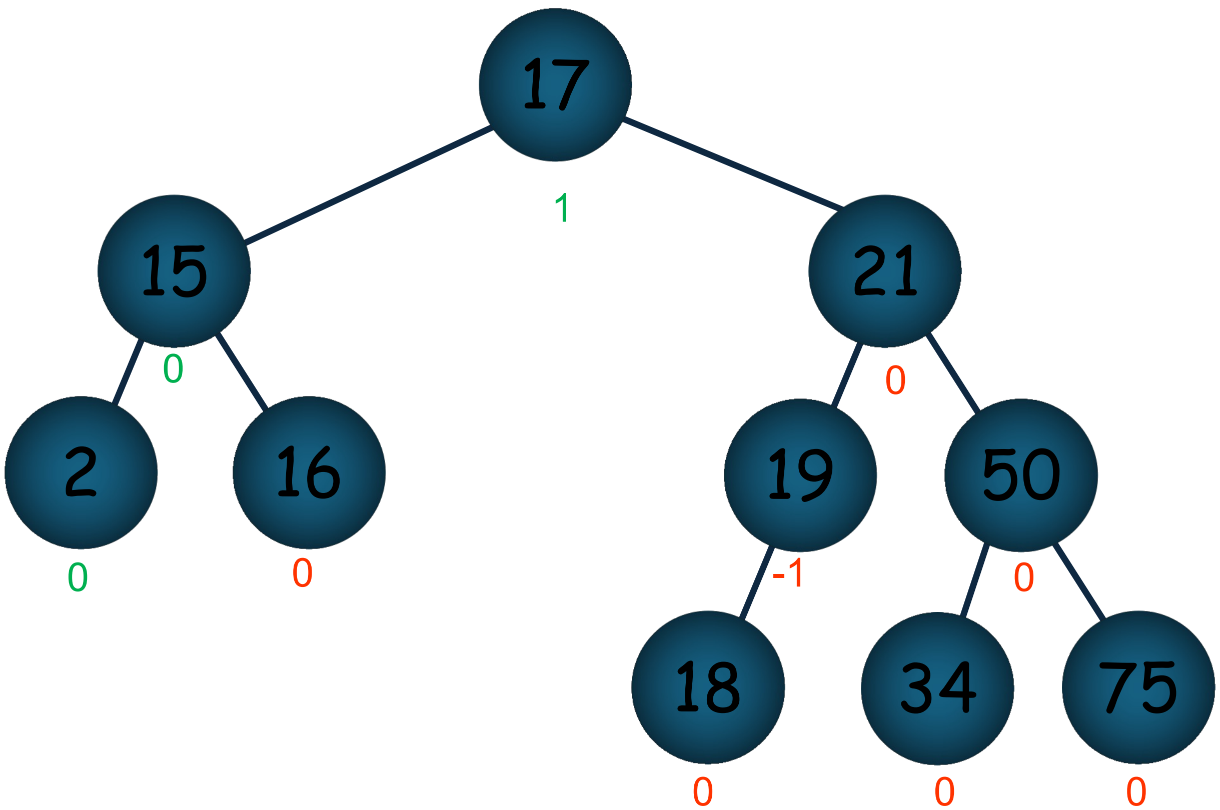


borrado en los AVL



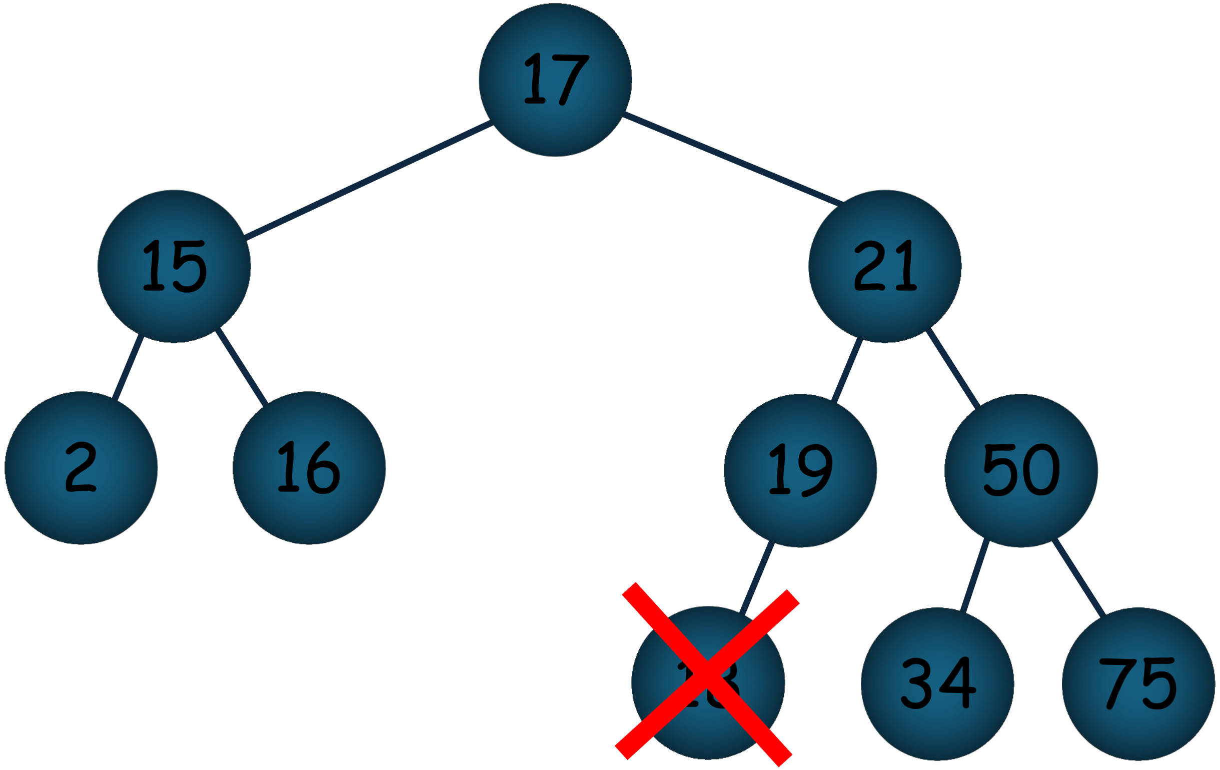
Balanceo DD:
rotación a izquierda(2)

borrado en los AVL

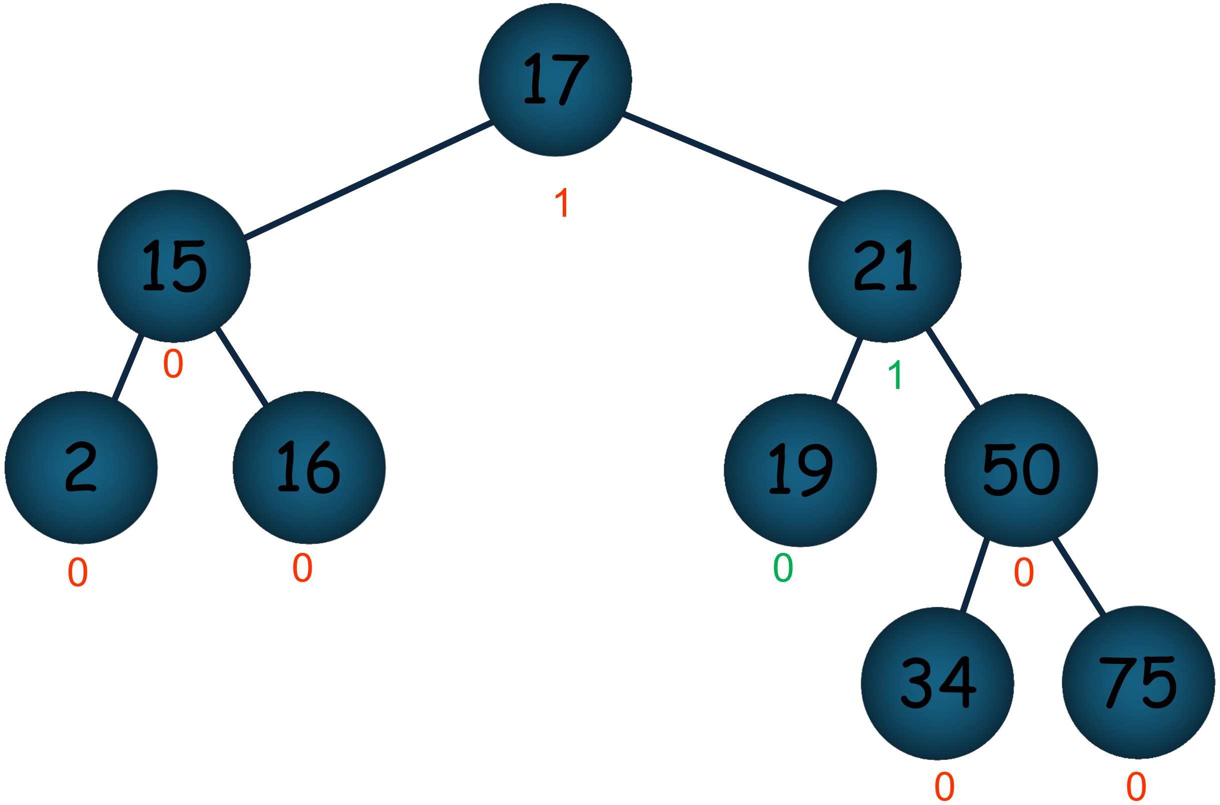


borrado en los AVL

Borrar 18

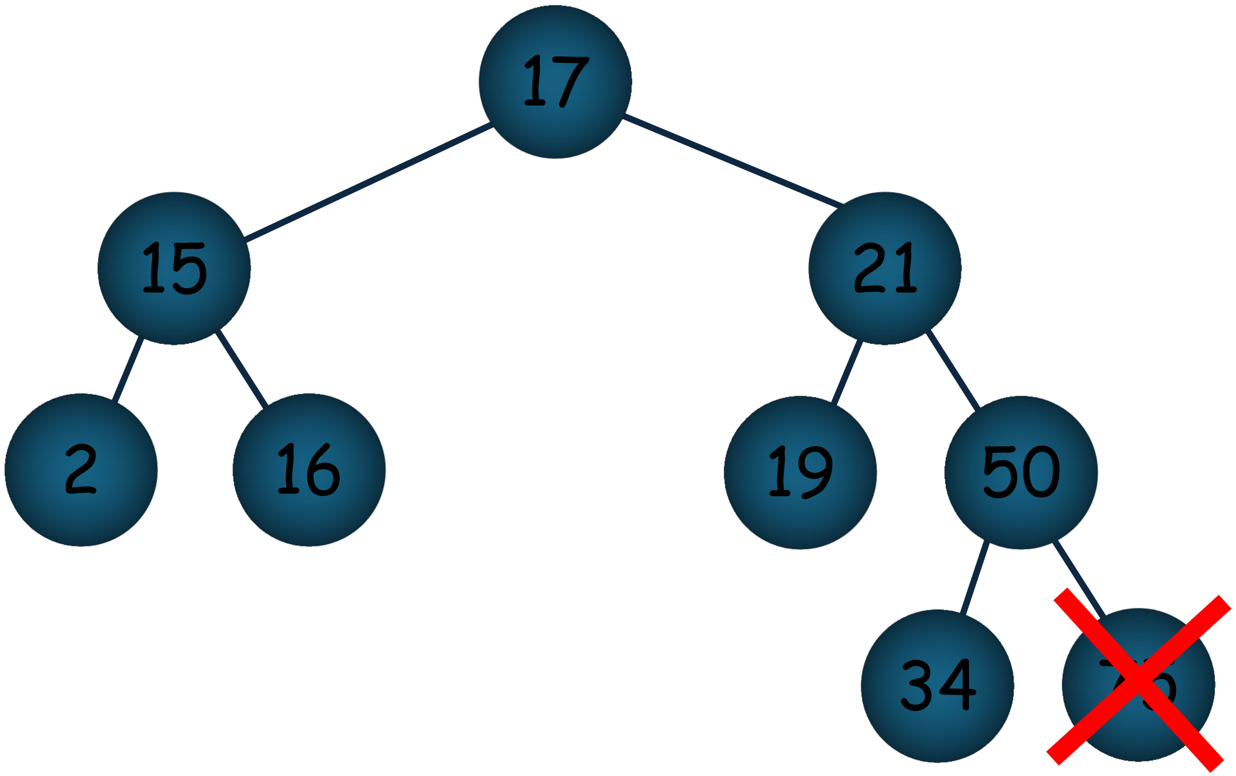


borrado en los AVL

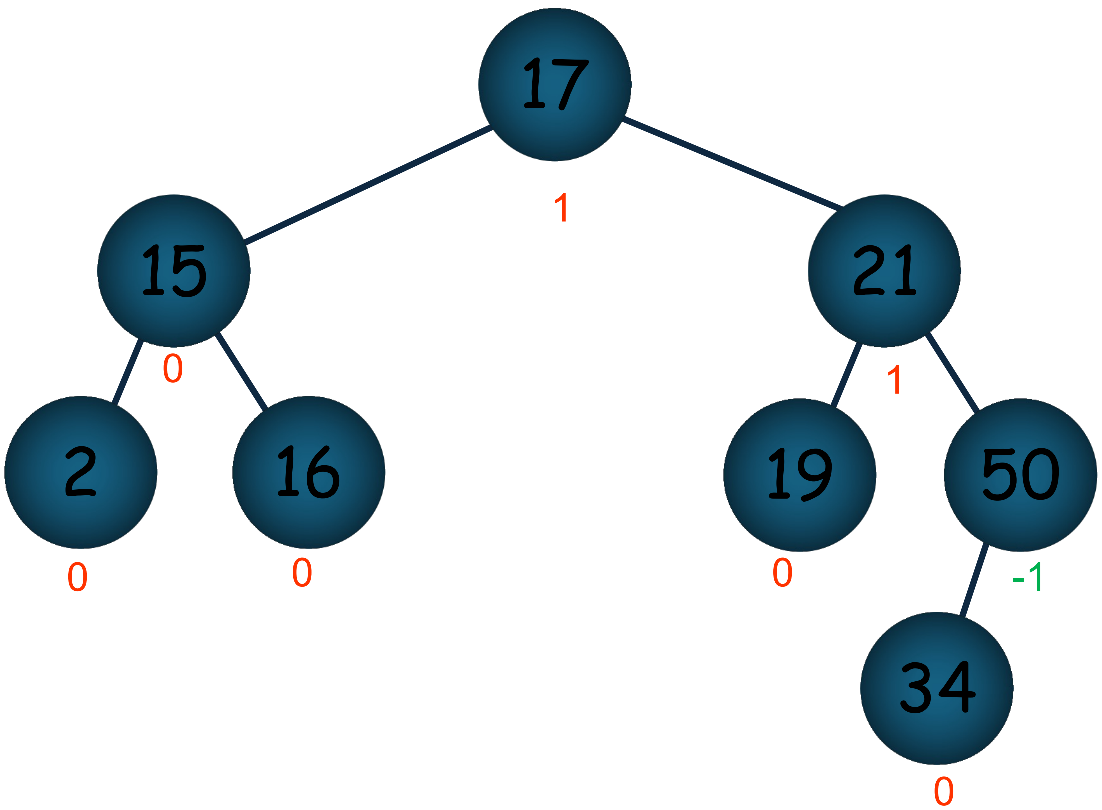


borrado en los AVL

Borrar 75

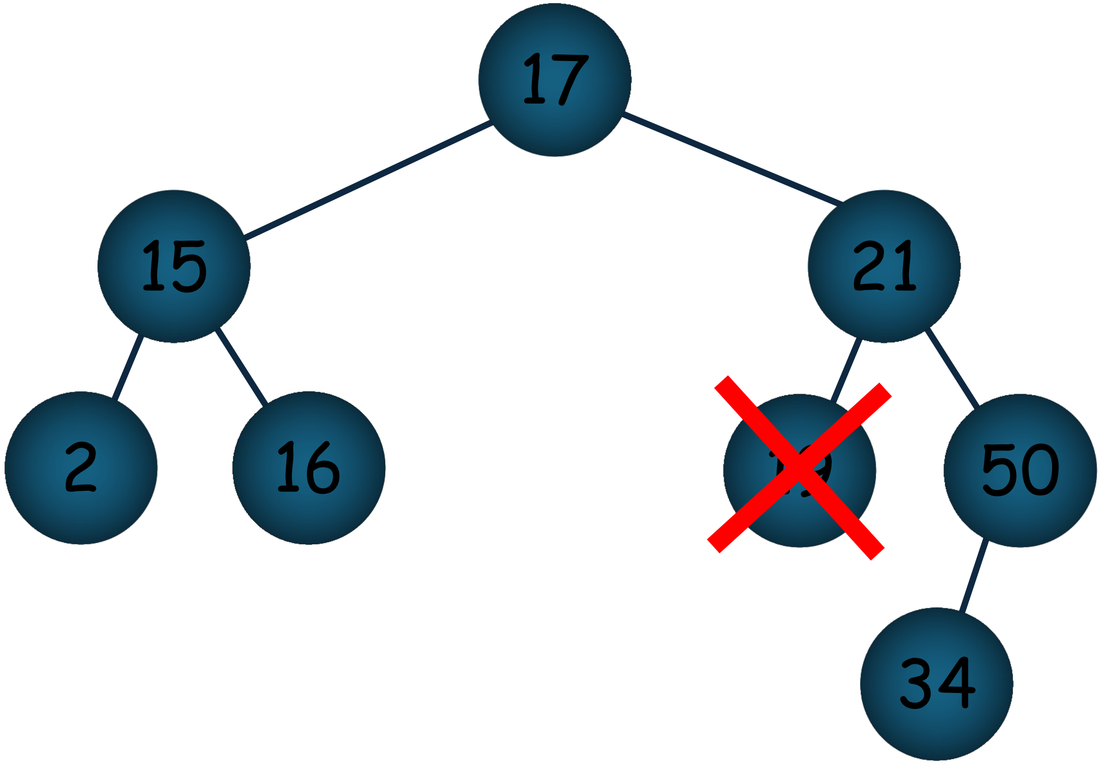


borrado en los AVL

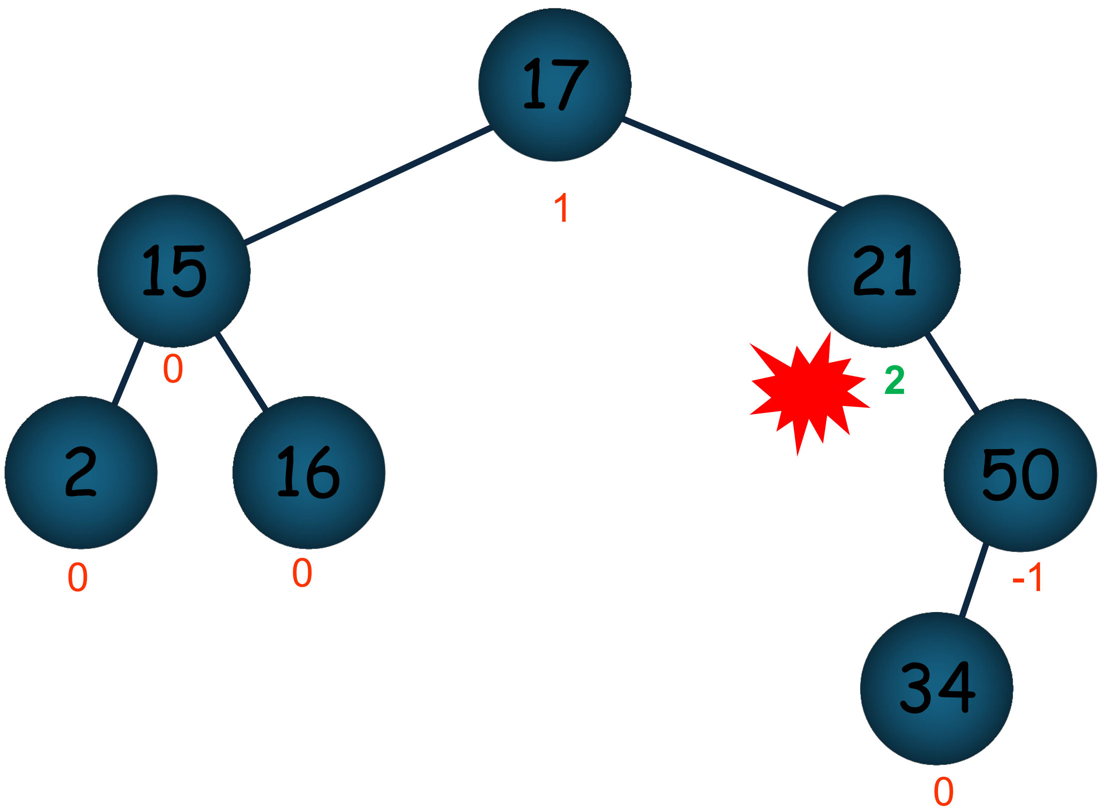


borrado en los AVL

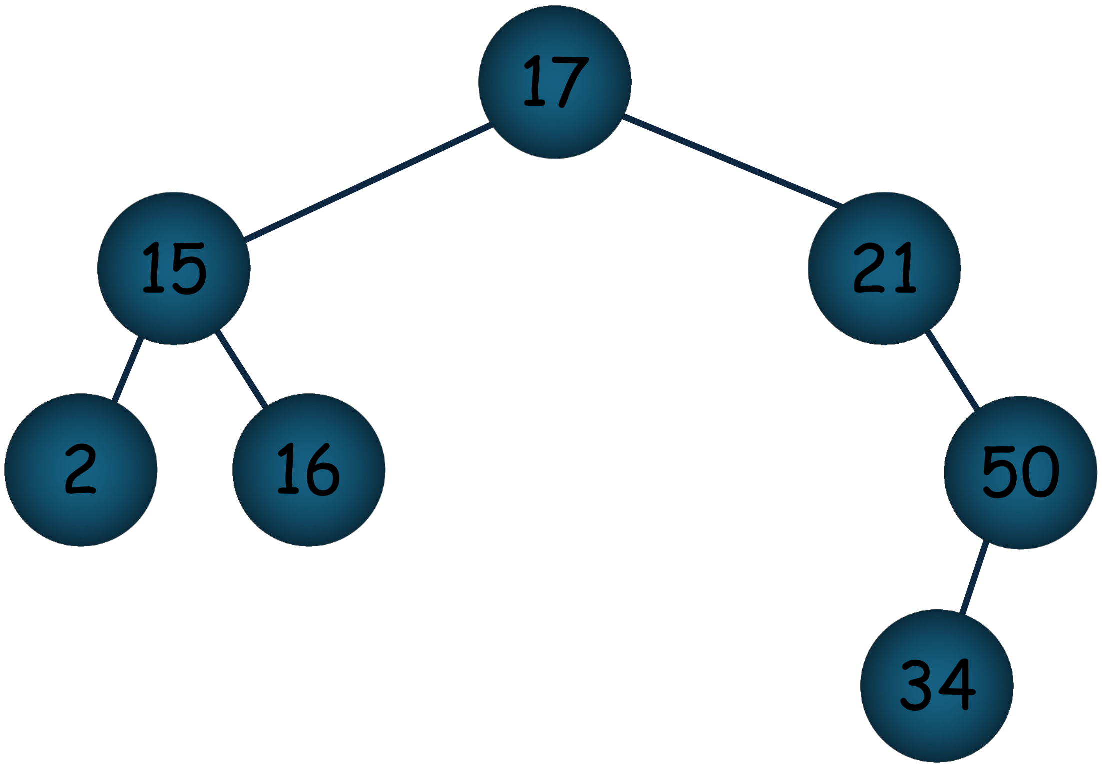
Borrar 19



borrado en los AVL

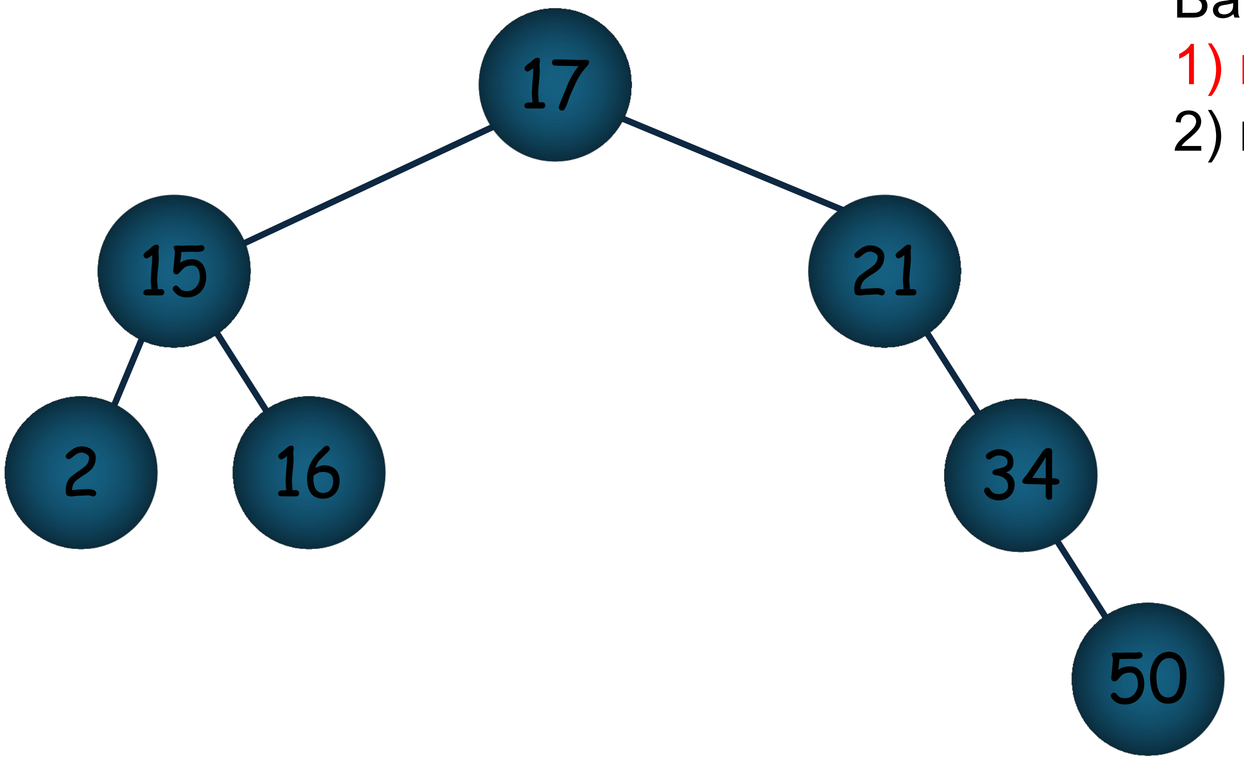


borrado en los AVL



Balanceo ID: derecha-izq
1) rotación a derecha(50)
2) rotación a izquierda(21)

borrado en los AVL

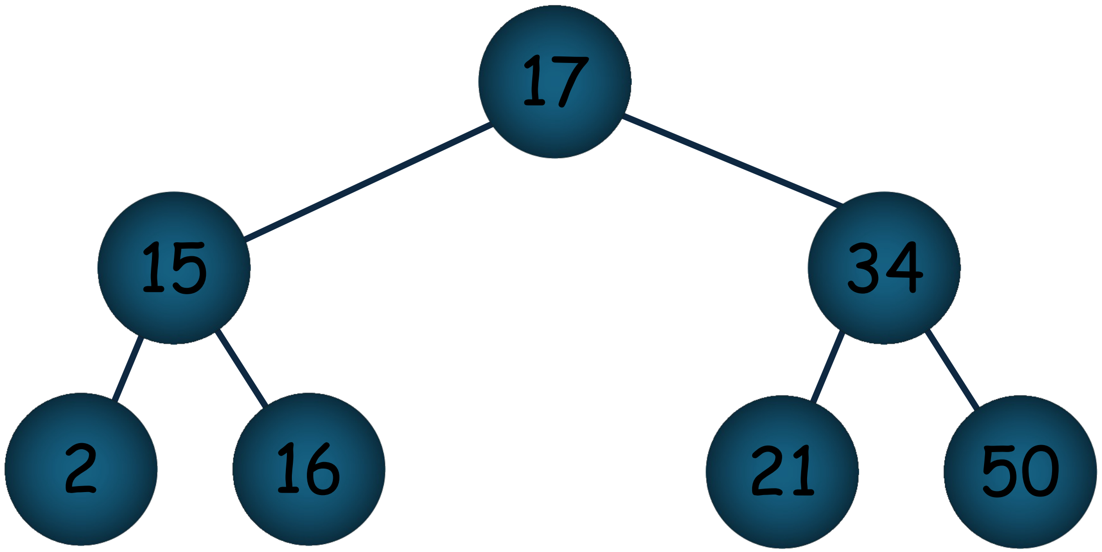


Balanceo ID: derecha-izq

1) rotación a derecha(50)

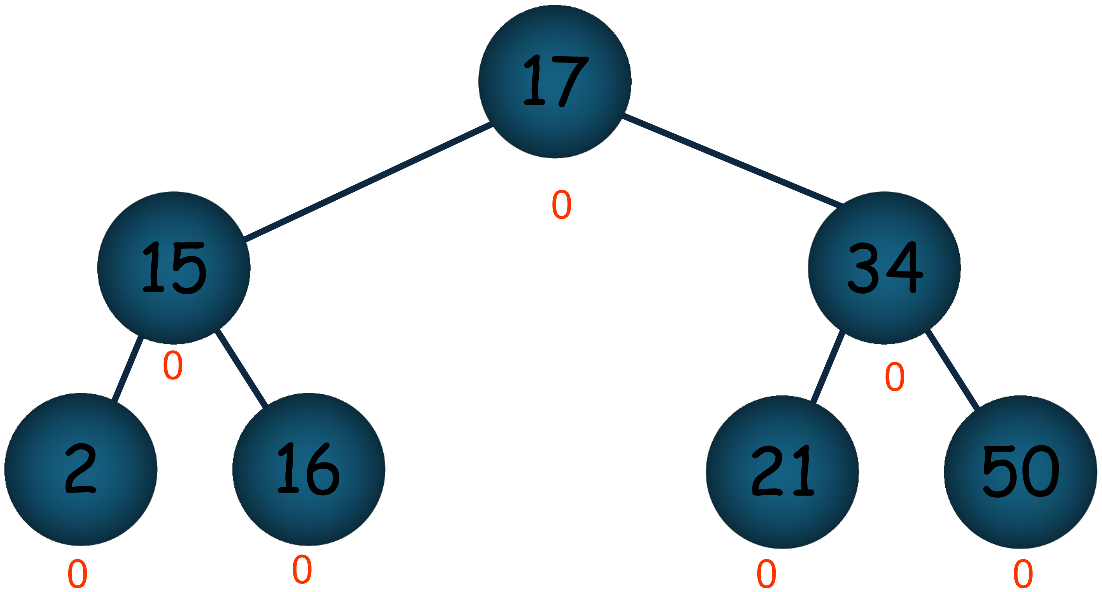
2) rotación a izquierda(21)

borrado en los AVL



Balanceo ID: derecha-izq
1) rotación a derecha(50)
2) rotación a izquierda(21)

borrado en los AVL



borrado en los AVL/costo

- en el caso peor hay que hacer rotaciones (simples o dobles) a lo largo de toda la rama
- paso 1: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 2: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 3: $\Theta(\lg n) \cdot \Theta(1)$

En total: $\Theta(\lg n)$