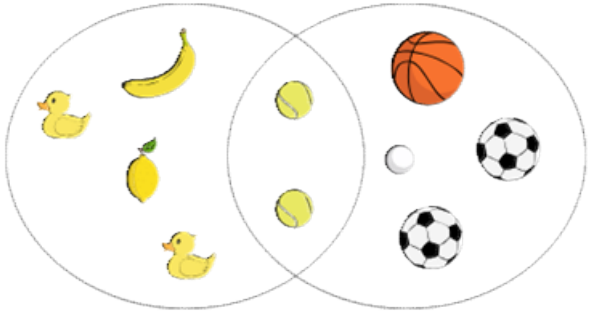
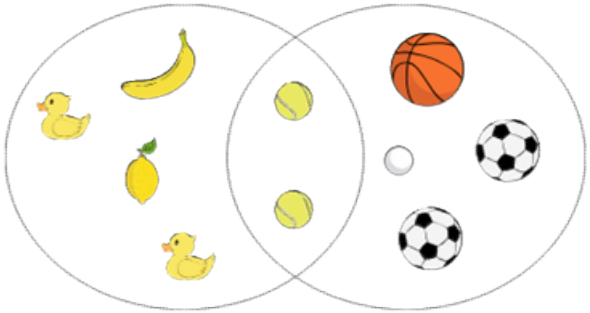
Diseño de Conjuntos y Diccionarios



Representación de Conjuntos y Diccionarios

- **TAD Diccionario(clave, significado)**
- **Observadores básicos**
- def?: clave c x dicc(clave, significado) d $\rightarrow$ bool
- obtener: clave c $\times$ dicc(clave, significado) d $\rightarrow$ significado (def?(c, d))
- **Generadores**
- vacío: $\rightarrow$ dicc(clave, significado)
- definir: clave $\times$ sign $\times$ dicc(clave, significado) $\rightarrow$ dicc(clave, significado)
- **Otras Operaciones**
- borrar: clave c $\times$ dicc(clave, significado) d $\rightarrow$ dicc(clave, significado) (def?(c, d))
- claves: dicc(clave, significado) $\rightarrow$ conj(clave)
- $\cdot =_{dicc} \cdot$: dicc(α) $\times$ dicc(α) $\rightarrow$ bool

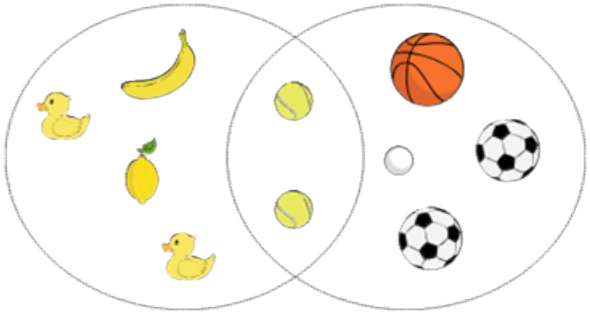
Definición del TAD Conjunto



```
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)
}
```

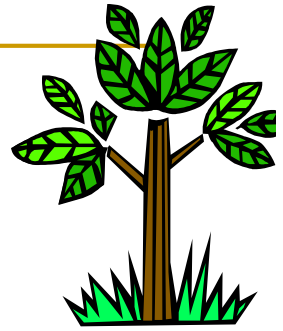
¿Conjuntos y Diccionarios?

- Vamos a pensar implementaciones de esos diccionarios, pero de paso, otras variantes:
  - Más de un significado es posible
    - Listas de significados
    - Conjuntos de significados
    - ¿qué obtenemos al `obtener`?
    - ¿y qué borramos al `borrar`?
  - Diccionarios con un solo significado posible (o sea |K|=1)
  - Los conjuntos son un caso particular de los diccionarios
  - Además, cualquier diccionario pueden ser pensados como si K fuera “punteros al significado”
  - En conclusión, lo más interesante es pensar en cómo representar conjuntos.

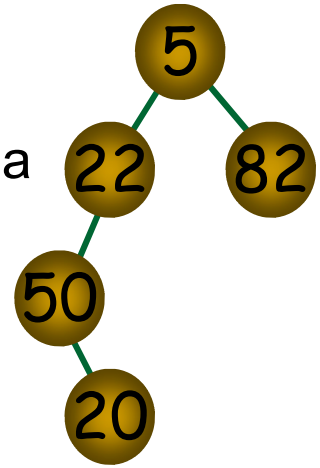
Representación de conjuntos y diccionarios a través de arrays

- Conjuntos y diccionarios pueden representarse a través de arrays (con o sin repetidos, ordenados o desordenados).
- Ya vimos varias de esas soluciones.
- Intenten hacer Uds. mismos el ejercicio de escribir INV, ABS, y los algoritmos
- Complejidad de las operaciones: depende de la implementación, pero
  - Tiempo: alguna de las operaciones requieren O(n) en el peor caso
  - Espacio: O(n).
  - ¿se podrá hacer mejor?

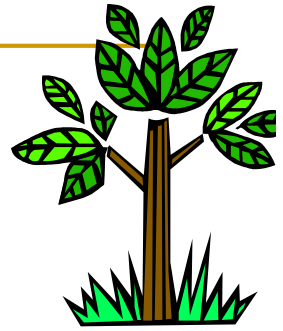
Árboles/Árboles Binarios



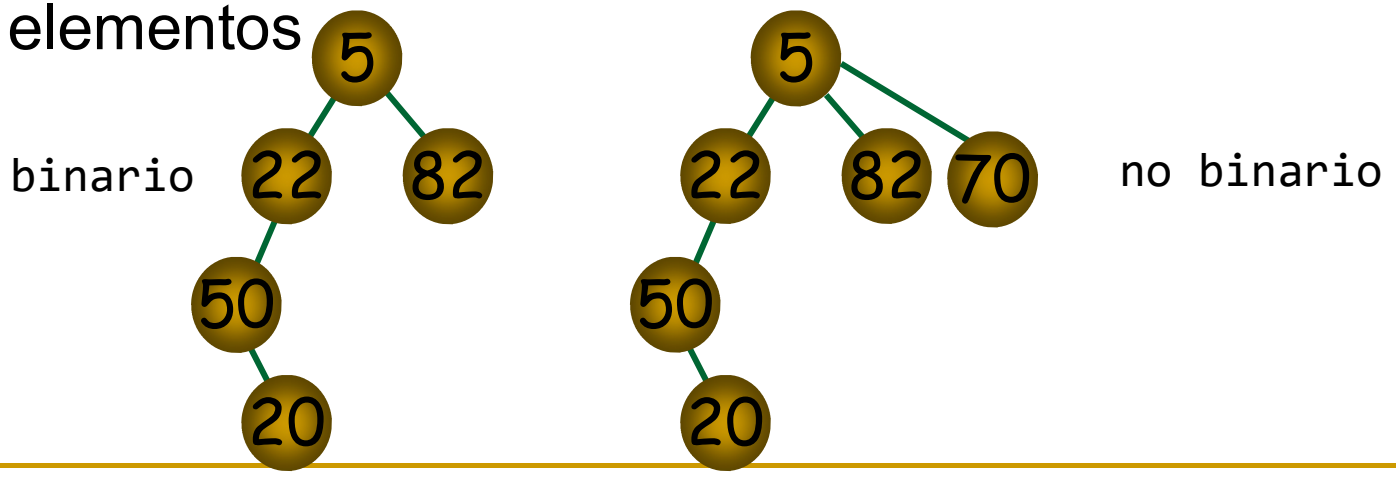
- Podemos definir el tipo conceptual (matemático) árbol<T>.
- Así como con las secuencias, podemos definir árboles de cualquier tipo T
- Se puede definir recursivamente como
  - Nil es un árbol<T>
  - una tupla que contiene un elemento de T y una secuencia de árboles<T>, es un árbol<T>.
- ¡Y se pueden dibujar!
- Ejemplos
  - Nil
  - <5,<nil,nil>>
  - <5,<<22,<50,<nil,<20,<nil,nil>>>,nil>,<82, <nil,nil>>



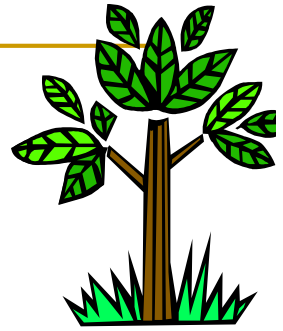
Árboles/Árboles Binarios



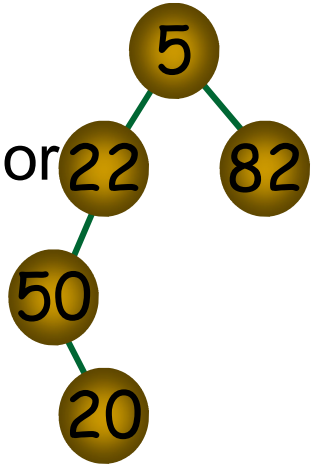
- Sobre árboles, usamos terminología variada:
  - «botánica» (raíz, hoja)
  - «genealógica» (padre, hijo, nieto, abuelo, hermano),
  - «física» (arriba, abajo)
  - «topológica»(?) (nodo interno, externo)
- Hay un tipo particular de árboles, que son los Árboles Binarios: la secuencia de árboles tiene como máximo dos elementos



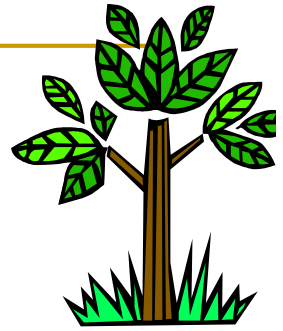
Árboles/Árboles Binarios



- El concepto matemático árbol tiene muchos usos, propiedades y funciones muy conocidas.
- Por ejemplo, dado un árbol a, podemos hablar de *nil?(a), raiz(a), altura(a), elementos(a), está(e,a)* y muchas más.
- Y para árboles binarios, también *izq(a) y der(a)*
- Esas funciones se puede definir recursivamente por ejemplo
  - Altura(nil)=0
  - Altura(<a,s>)=1+max$_i${altura(s_i)}



Árboles/Árboles Binarios



- Podemos definir el TAD Arbol y el TAD Arbol Binario
- El TAD Arbol Binario podría tener versiones de todas las operaciones conceptuales y los constructores *nil* y *bin(c,I,D)*.
- Podemos implementar Árboles binarios con punteros:

```
Nodo = Struct <dato: N, izq: Nodo, der: Nodo >

Módulo AB implementa Árbol Binario {
raíz: Nodo
}
```

- Y podemos escribir un invariante de representación, y una función de abstracción (pero no lo vamos a hacer acá).

Representación de conjuntos y diccionarios a través de Arboles Binarios

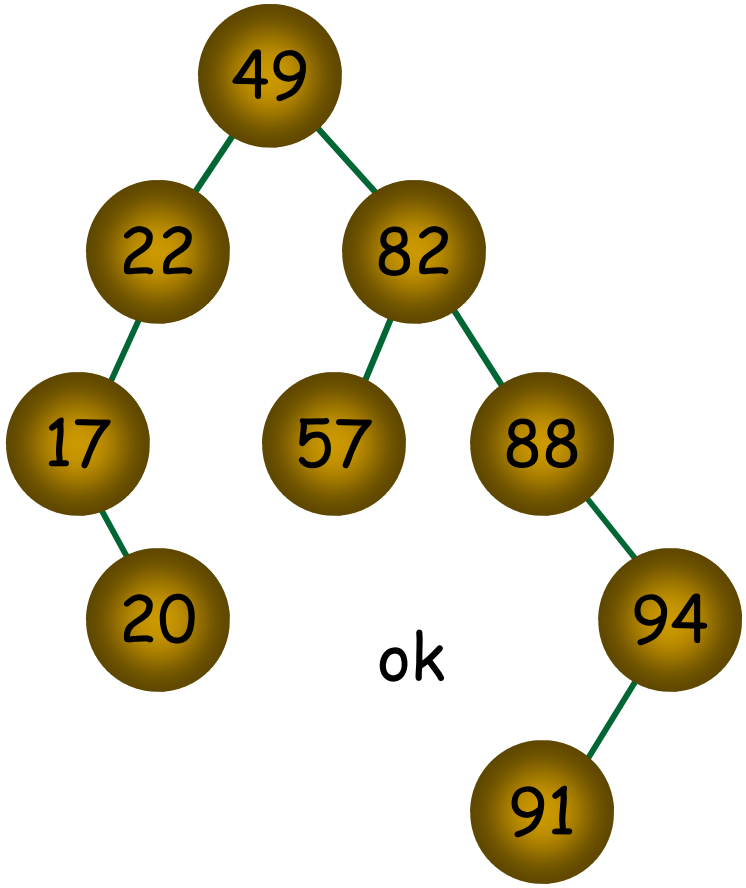
- ¿Podríamos representar conjuntos o diccionarios a través de árboles binarios?
- Claro que podríamos
- ¿Ganaríamos algo? No demasiado en principio ¿no?
- Pero.....

Arboles Binarios de Búsqueda (ABB)

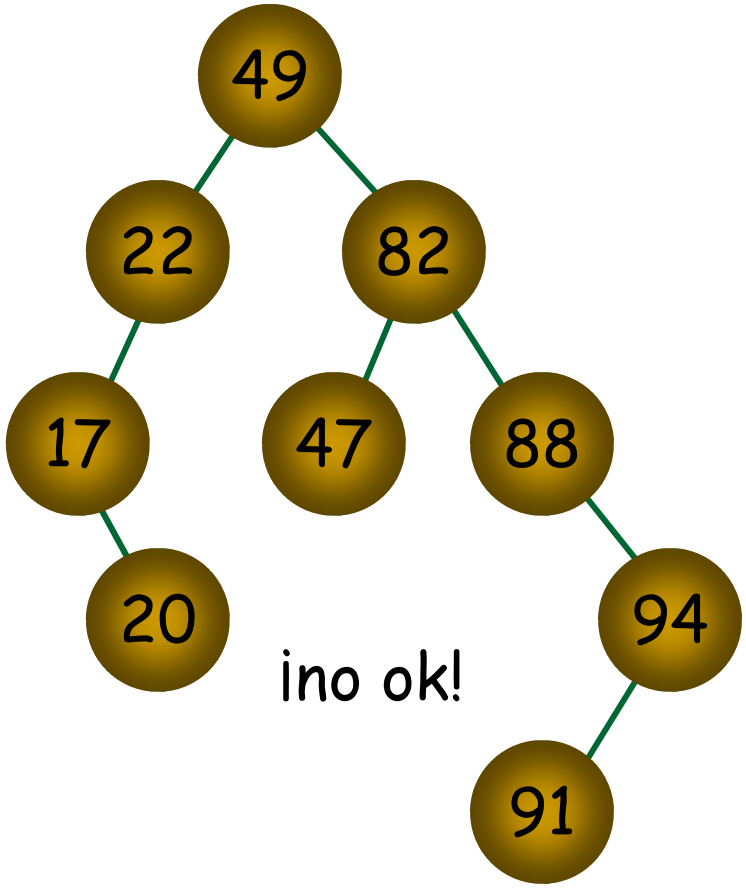
- ¿Qué es un Árbol Binario de Búsqueda?
- Es un AB que satisface la siguiente propiedad:
  - Para todo nodo, los valores de los elementos en su subárbol izquierdo son menores que el valor del nodo, y los valores de los elementos de su subárbol derecho son mayores que el valor del nodo
  - Dicho de otra forma, el valor de todos los elementos del subárbol izquierdo es menor que el valor de la raíz, el valor de todos los elementos del subárbol derecho es mayor que el valor de la raíz, y tanto el subárbol izquierdo como el subárbol derecho....son ABB.
- Formalmente:

```
esABB(nil): TRUE
esABB(<C,<I,D>>: ∀e in elems(I) e<=C ^ ∀e in elems(D) e>C ^ esABB(I)
^ esABB(D)
```

Ejemplos



ok



¡no ok!

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.arbol)=TRUE}
```

- La función de abstracción es obvia.
- Como adelantamos, vamos a dejar de usar la notación que usamos hasta ahora, para pasar a usar….alguna que se entienda.

Los algoritmos para ABB

- vacío()
  - nil
- agregar(A,c)
  - If nil?(A) then bin(c,nil,nil) else

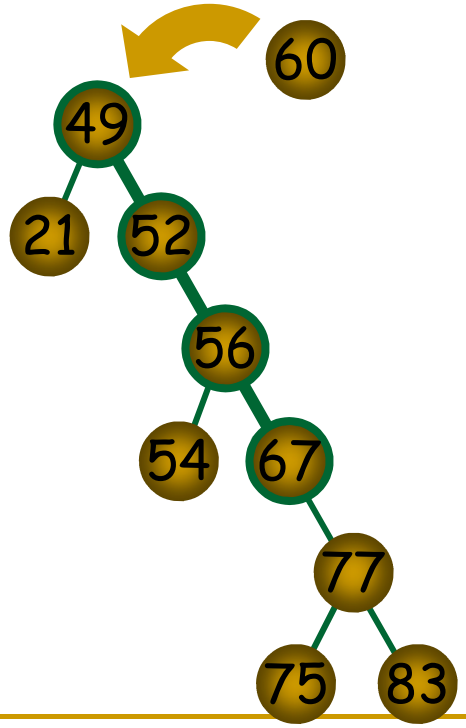
    if c<raíz(A) then

    bin(raíz(A),agregar(izq(A),c),der(A))

    else bin(raíz(A),izq(A), agregar(der(A),c))

Los algoritmos para ABB

- O sea:
  - Buscar al padre del nodo a insertar
  - Insertarlo como hijo de ese padre



Los algoritmos para ABB

- Costo de la inserción:
  - Depende de la distancia del nodo a la raiz
- En el peor caso
  - O(*n*)
- En el caso promedio (suponiendo una distribución uniforme de las claves):
  - O(lg *n*)

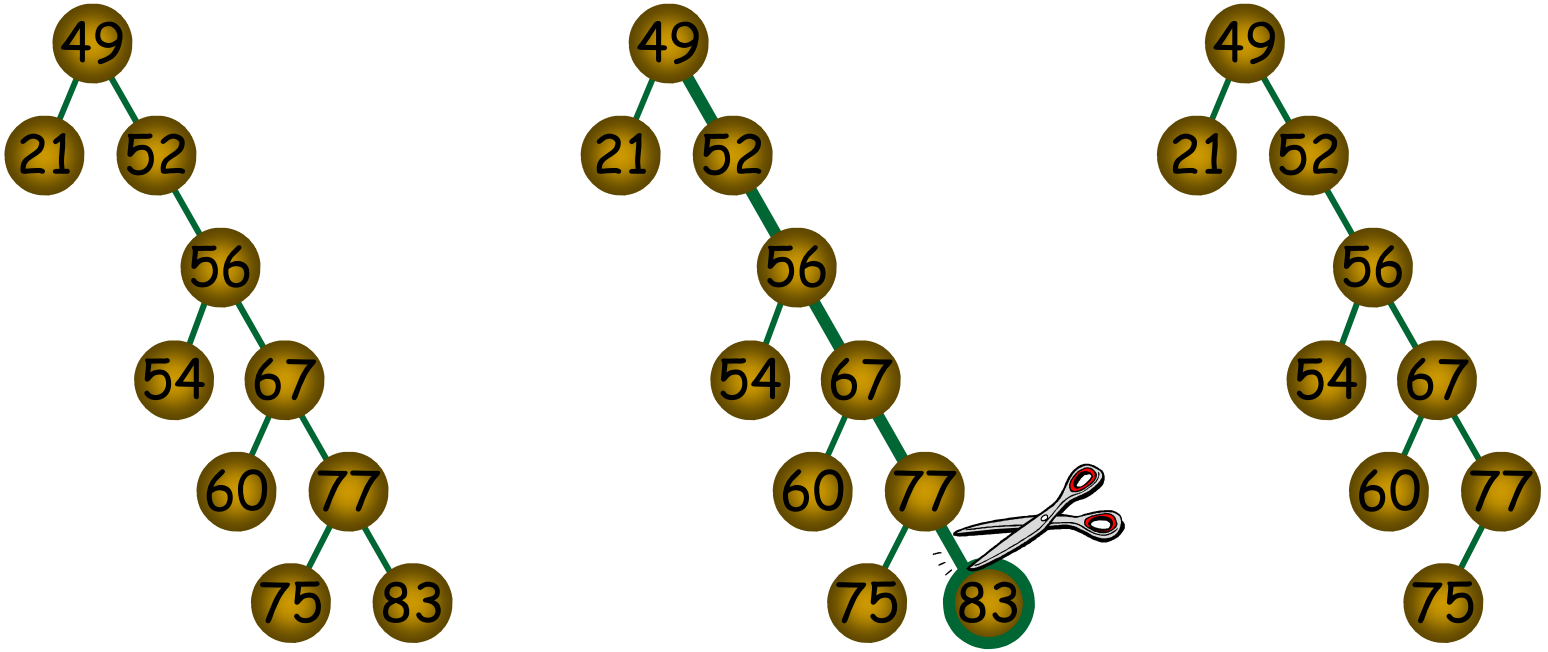
Los algoritmos para ABB: borrado

- Borrar(u,A)
- Tres casos
  1. u es una hoja
  2. u tiene un solo hijo
  3. u tiene dos hijos

- Vamos a ver la idea, Uds. la pueden formalizar luego

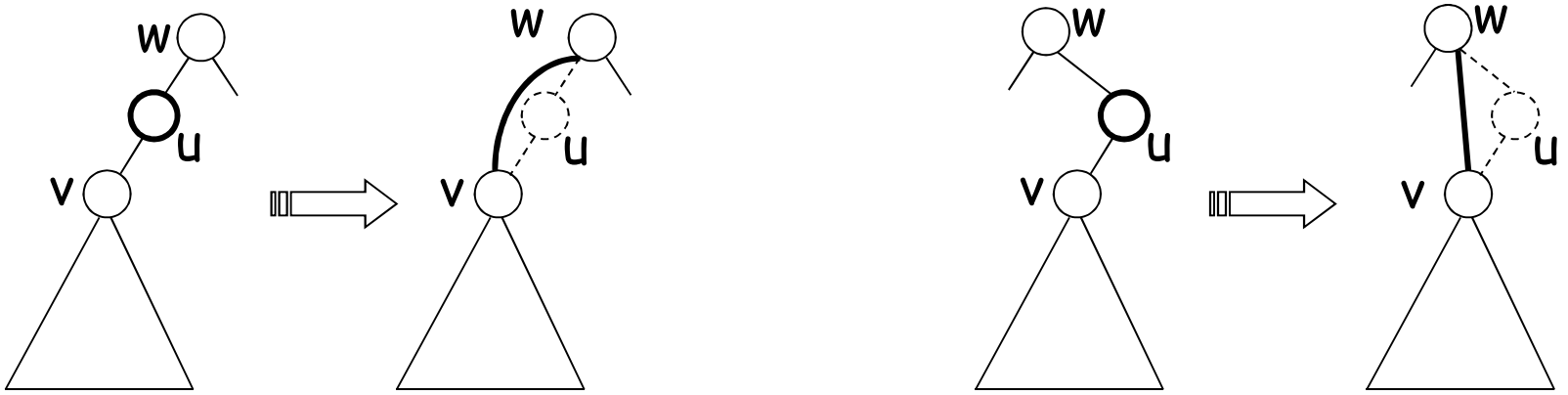
Borrado en ABB

1. **Borrar una hoja**
   - Buscar al padre
   - Eliminar la hoja



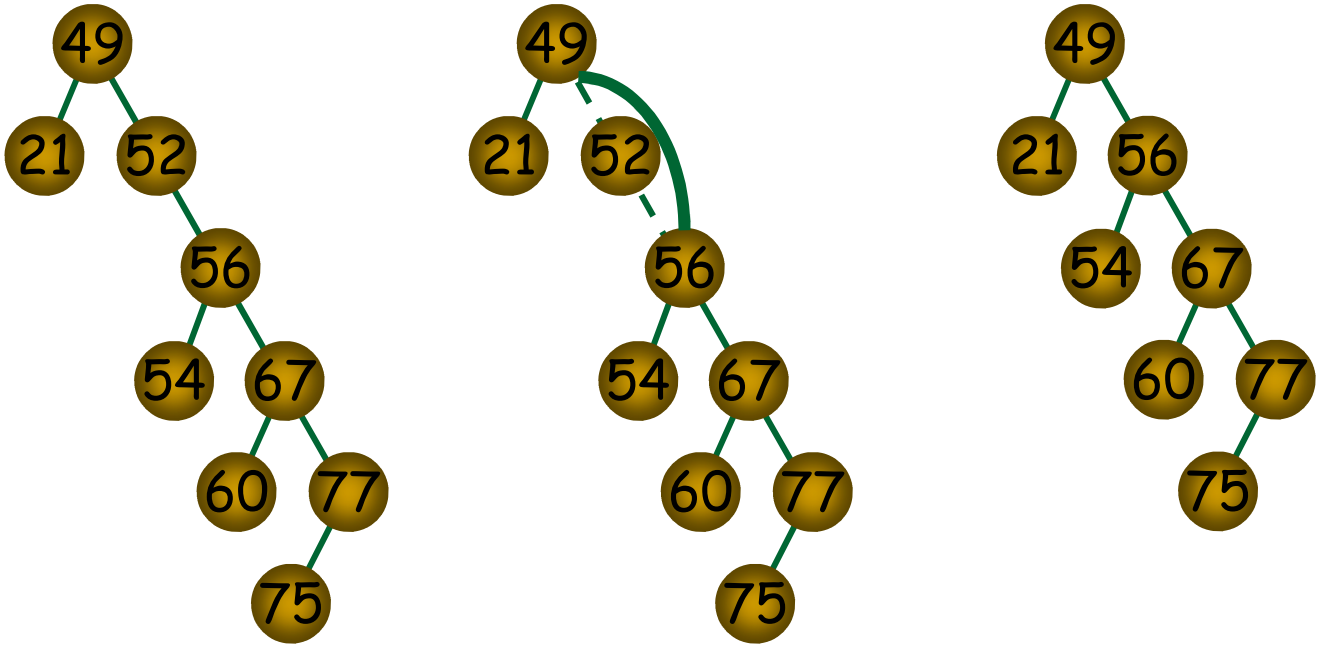
Borrado en ABB

2. **Borrar un nodo u con un solo hijo v**
   - Buscar al padre w de u
   - Si existe w, reemplazar la conexión (w,u) con la conexión (w,v)



Borrado en ABB

Ejemplo del caso 2

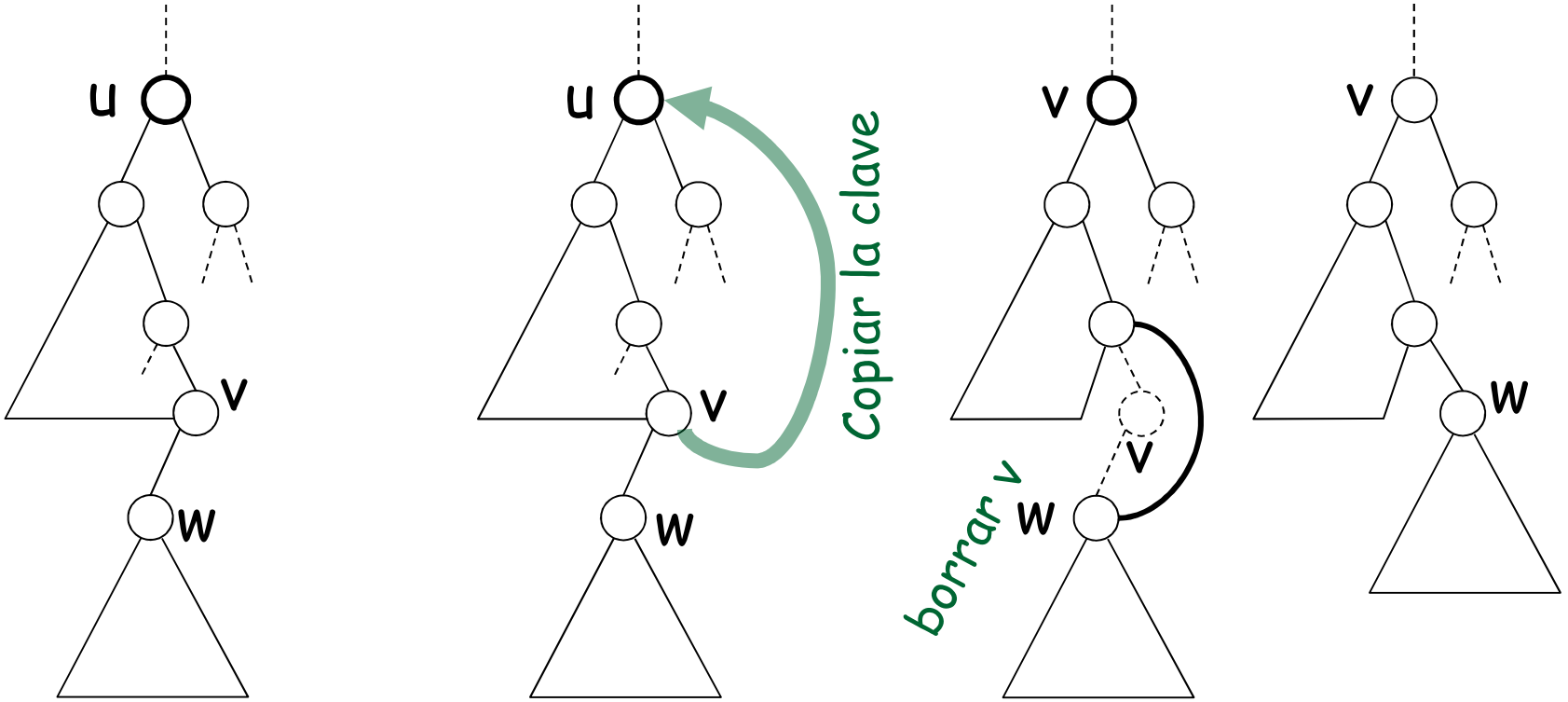


Borrado en ABB

3. **Borrado de un nodo u con dos hijos**

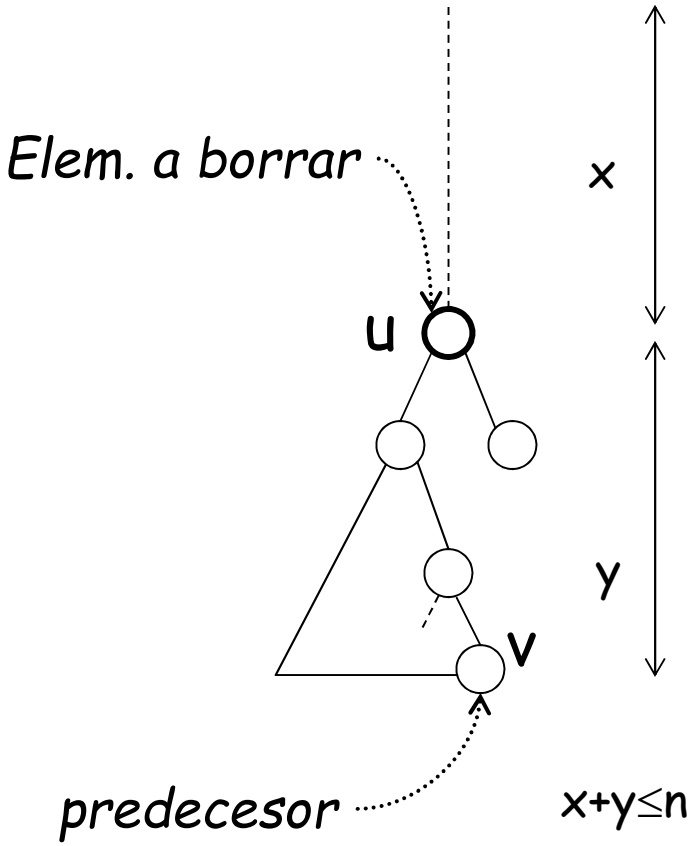
- Encontrar el “predecesor inmediato” v (o sucesor inmediato) de u
  - v no puede tener dos hijos, en caso contrario no sería el predecesor inmediato (sucesor)
- copiar la clave de v en lugar de la de u
- Borrar el nodo v
  - v es hoja, o bien tiene un solo hijo, lo que nos lleva los casos anteriores

Borrado en ABB



Costo del borrado en un ABB

- El borrado de un nodo interno requiere encontrar al nodo que hay que borrar y a su predecesor inmediato
- En el caso peor ambos costos son lineales:
  O(n) + O(n) = O(n)



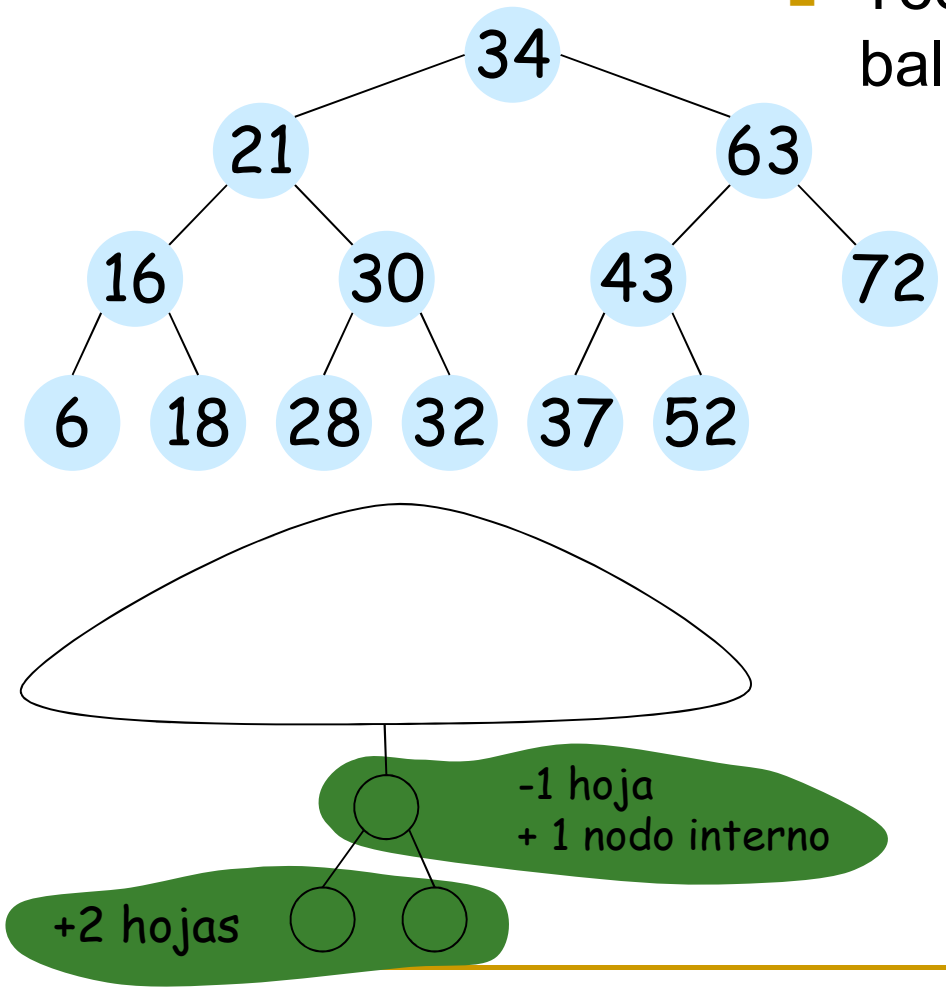
Representación de conjuntos y diccionarios a través de AVL

- Todas las representaciones vistas hasta ahora tienen al menos una operación de costo linear en función de la 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 (¡mantenerlos completos sería demasiado caro!)
- 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



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

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

Dem: Si cada nodo tiene 0 o 2 hijos

$n_h = n_i + 1$

n_h = # hojas

n_i = # nodos internos

$n = n_h + n_i$

$n_h' = n_h + 2 - 1 = n_h + 1 = n_i + 1 + 1 = n_i' + 1$

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

balanceo perfecto/2

- “Podamos” el árbol eliminando primero las hojas de las ramas más largas
- Luego podamos todas las hojas, nos queda otro árbol con las mismas características (cantidad de hijos por nodo y balanceo)
- ¿Cuántas veces podemos “podar” el árbol?
- Fácilmente generalizable a árboles k-arios

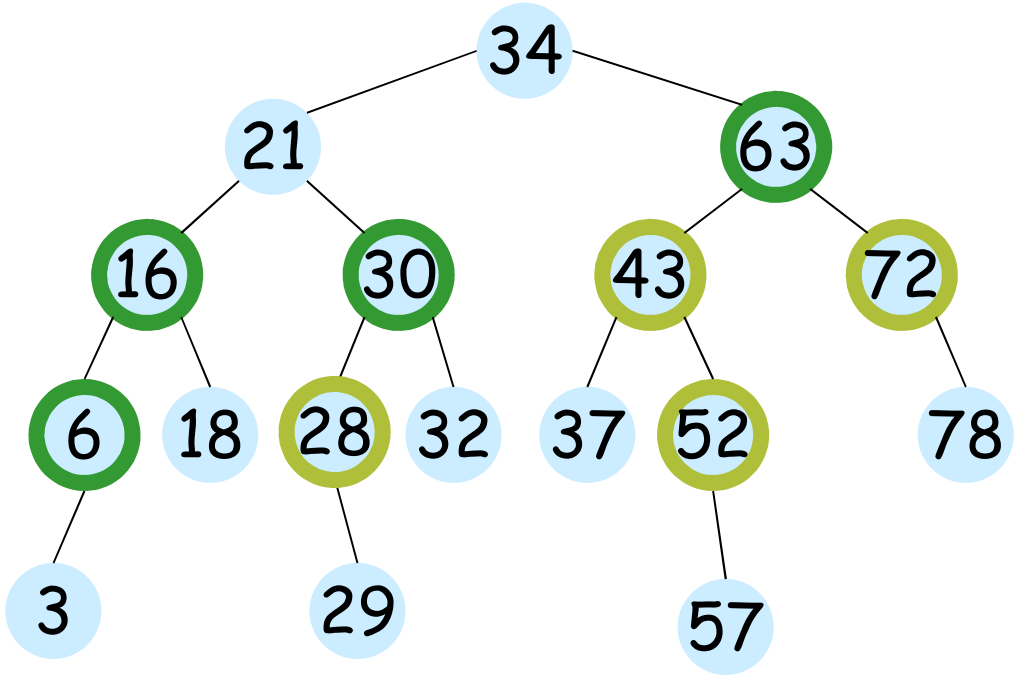
$$n_{\mathrm{h}} = (k-1)n_{\mathrm{i}} + 1 \quad \Rightarrow \quad n_{\mathrm{h}} = \frac{(k-1)n+1}{k}$$

- costo de búsqueda/inserción/borrado O(log *n*)
- Pero....sucesiones de inserciones y borrados pueden destruir el balanceo!
  - Degradación de la performance

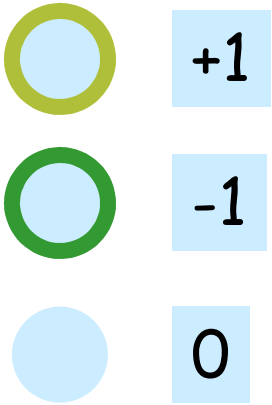
Balanceo en altura

- 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 Геóргий Максúмович Адельсóн у Вéльский у Евгений Михайлович Ландис
- También conocidos como Gueorgui Maksimovich Adelsón-Velski (1922-2014) y Yevgueni Mijáilovich Landis (1921-1997)

Factor de balanceo

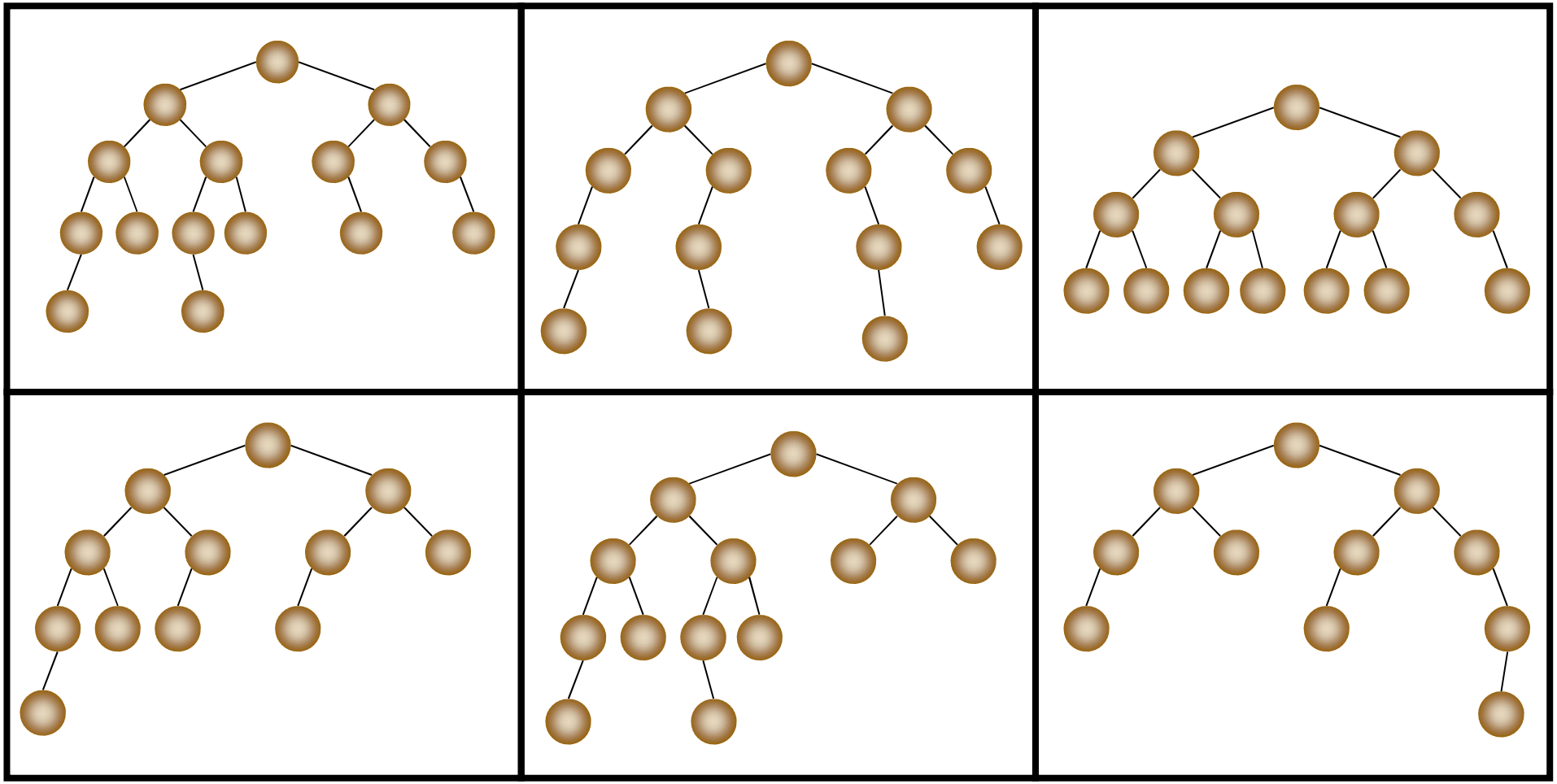


factor de balanceo (FDB):
altura subárbol Der -
altura subárbol Izq



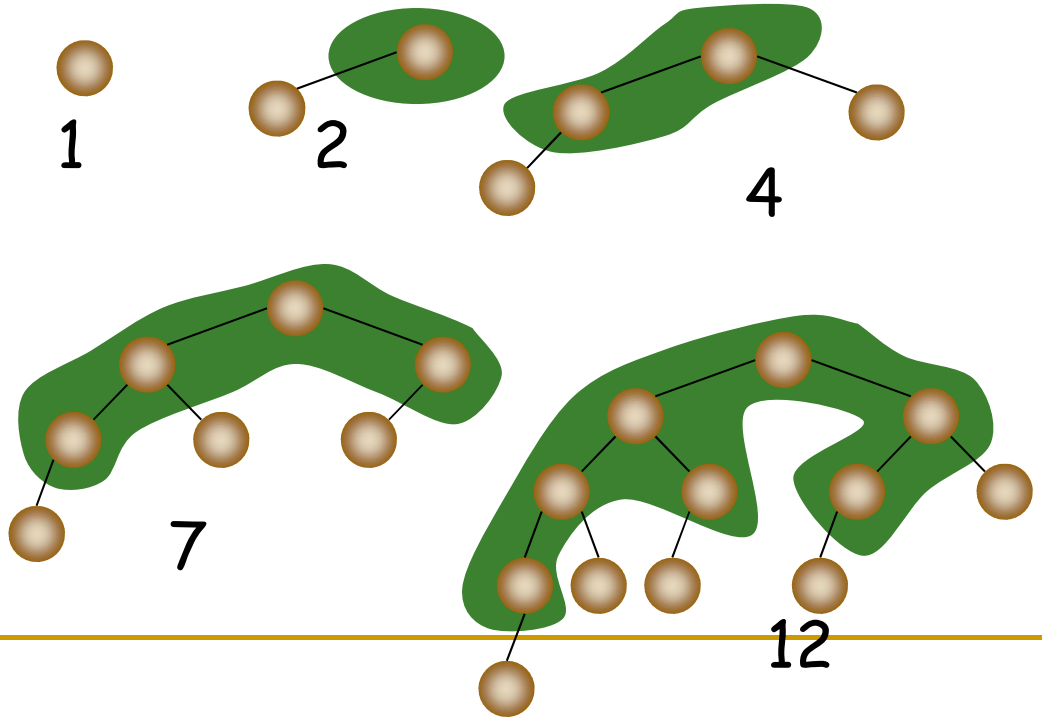
en un árbol balanceado en altura
$|FDB| \leq 1$, para cada nodo

Árboles AVL?



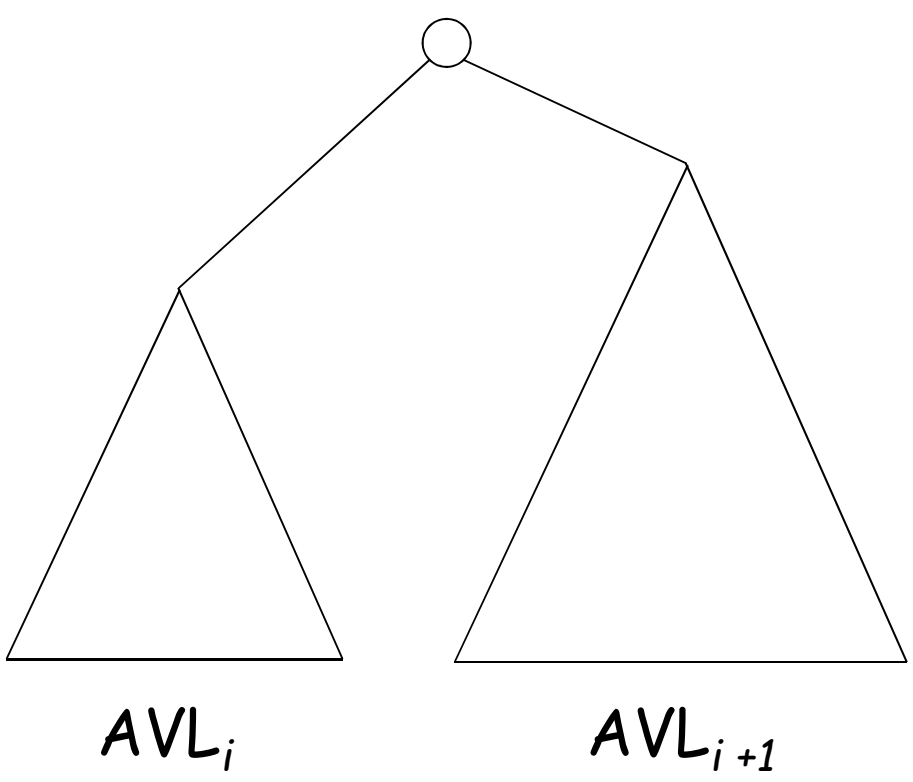
árboles de Fibonacci

- 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



árboles de Fibonacci

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

AVL_{i+2}

Relaciones

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

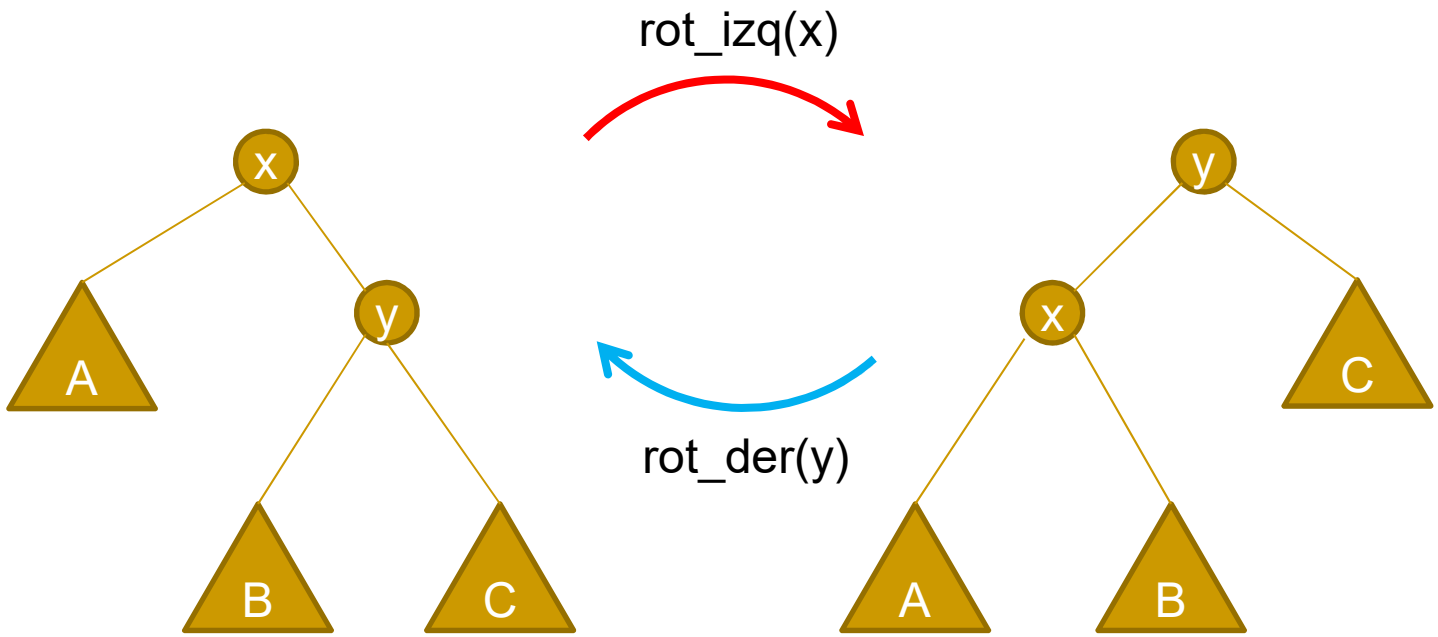
$F_{i+2} = F_i + F_{i+1}$

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

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 Θ(lg *n)*

rotaciones



AxByC

AxByC

inserción en AVL

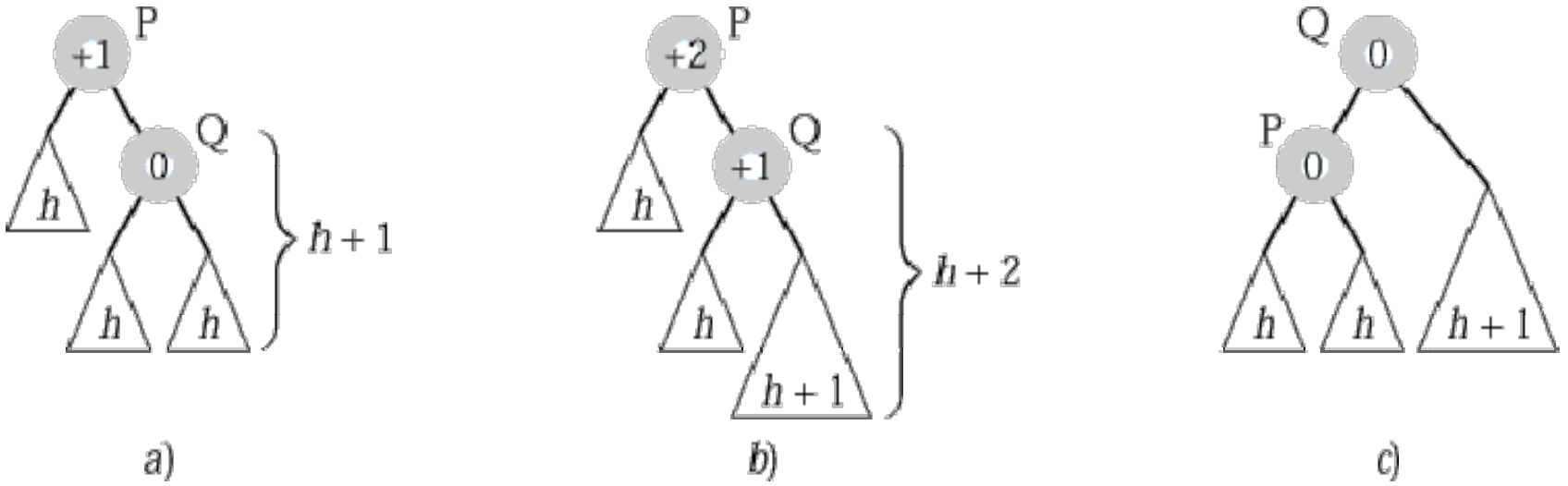
1. Insertar el nuevo nodo como en un ABB “clásico”
   - el nuevo nodo es una hoja
2. Recalcular los factores de balanceo que cambiaron por la inserción
   - sólo en la rama en la que ocurrió la inserción (los otros factores no pueden cambiar!), de abajo hacia arriba
3. Si en la rama aparece un factor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

rotaciones en los AVL

casos posibles

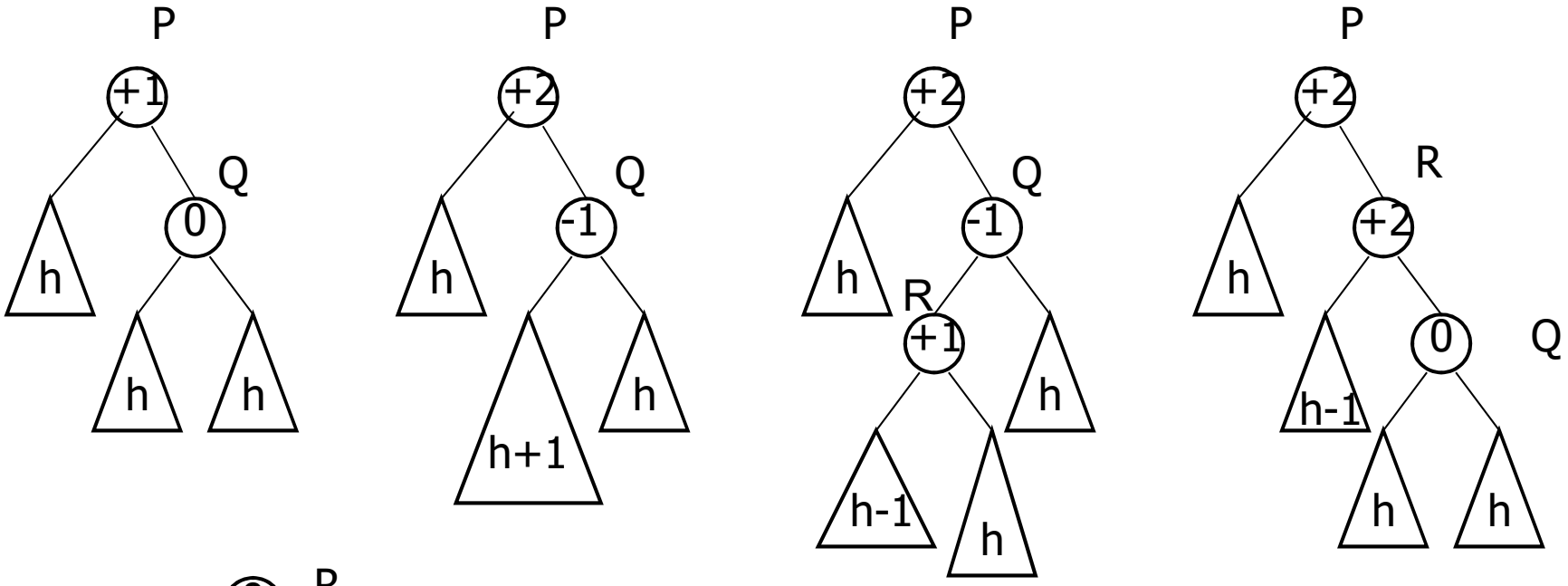
- RR: inserción en el subárbol derecho de un hijo derecho (del nodo que se desbalancea)
- LR: inserción en el subárbol izquierdo de un hijo derecho (del nodo que se desbalancea)
- RL: inserción en el subárbol derecho de un hijo izquierdo (del nodo que se desbalancea)
- LL: inserción en el subárbol izquierdo de un hijo izquierdo (del nodo que se desbalancea)

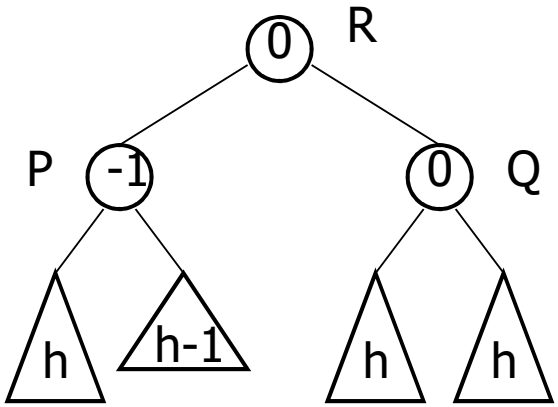
rotación simple (caso RR)



- La inserción no influye en los antepasados de P porque luego de la rotación recuperan su factor de balanceo anterior

rotación doble (caso LR)





- la inserción no influye en los antepasados de P
- ¿Cómo sería el subcaso en que el FDB de R es -1?

inserción en los AVL/costo

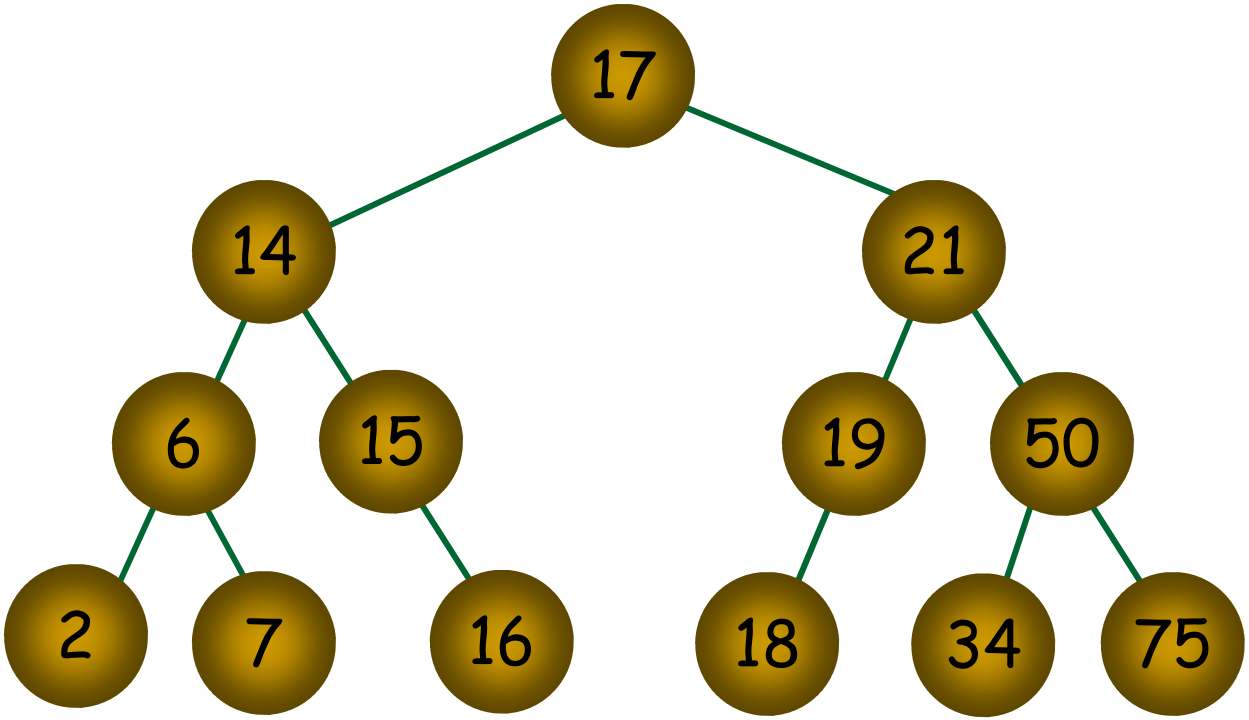
- paso 1: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 2: proporcional a la altura del árbol $\Theta(\lg n)$
- paso 3: $O(1)$ (se hace una o dos rotaciones por inserción)

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

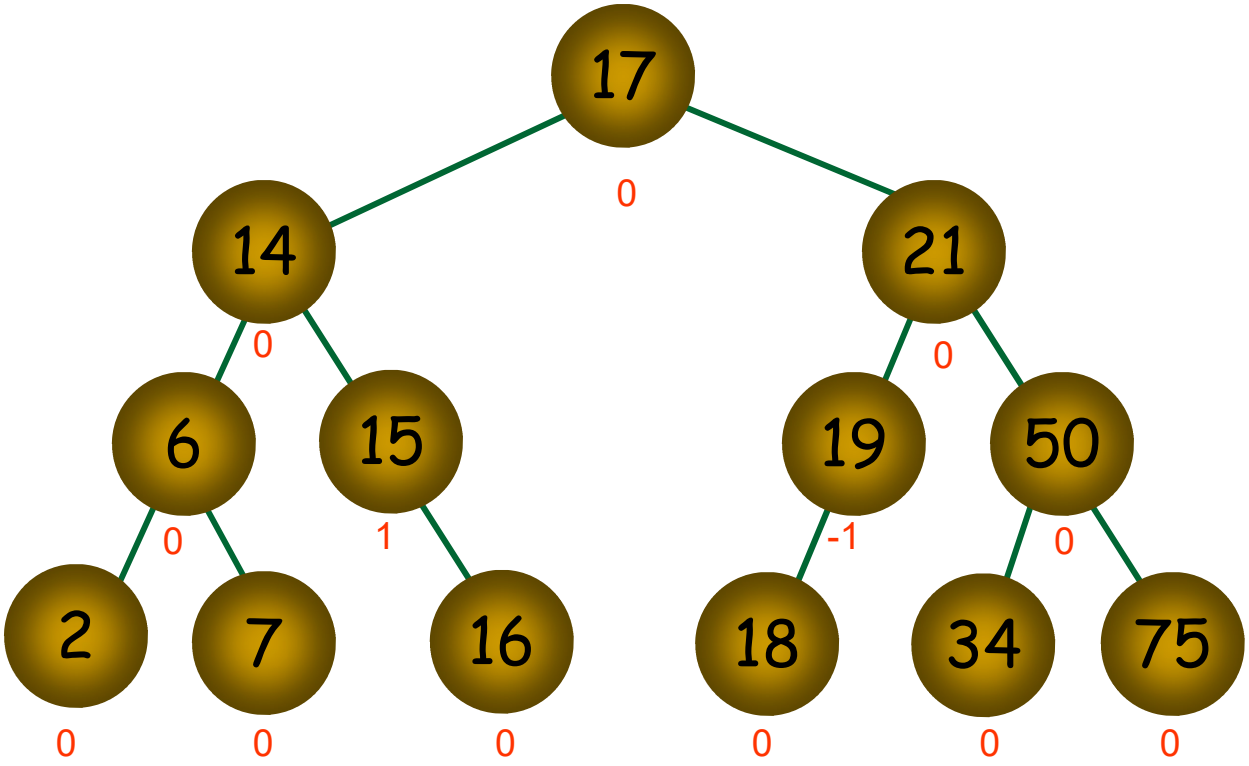
borrado en los AVL

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. para cada nodo 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

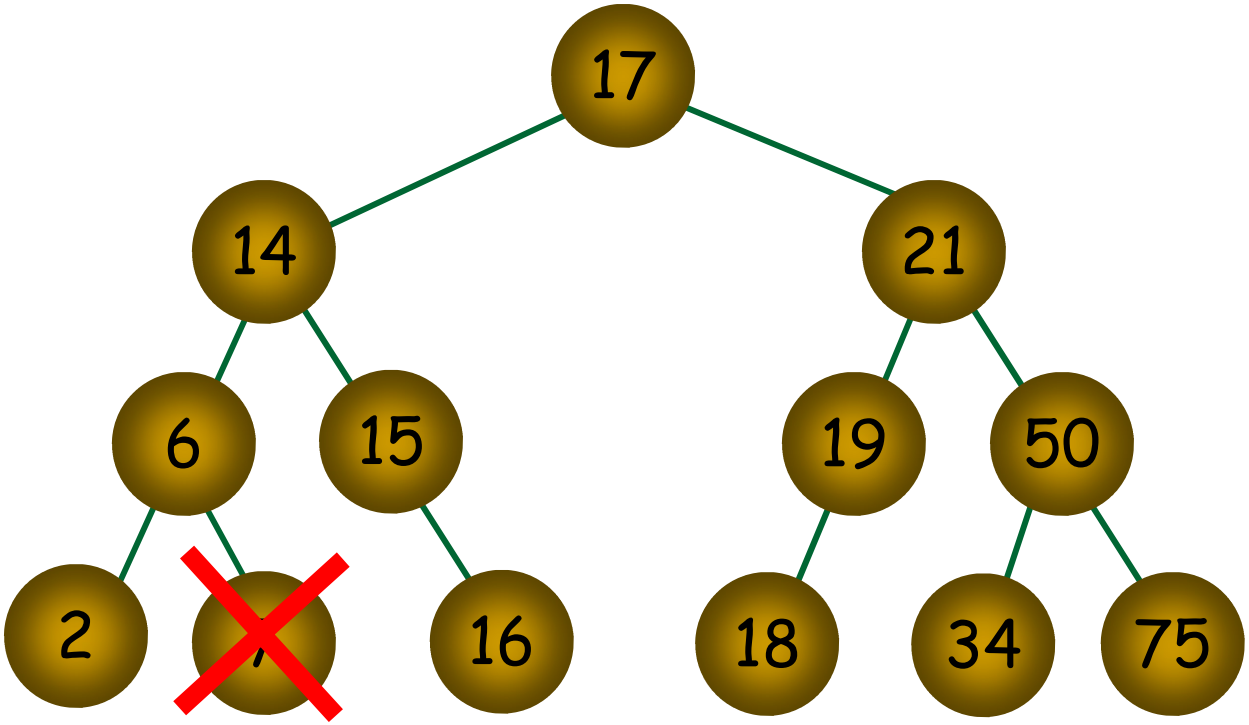


borrado en los AVL

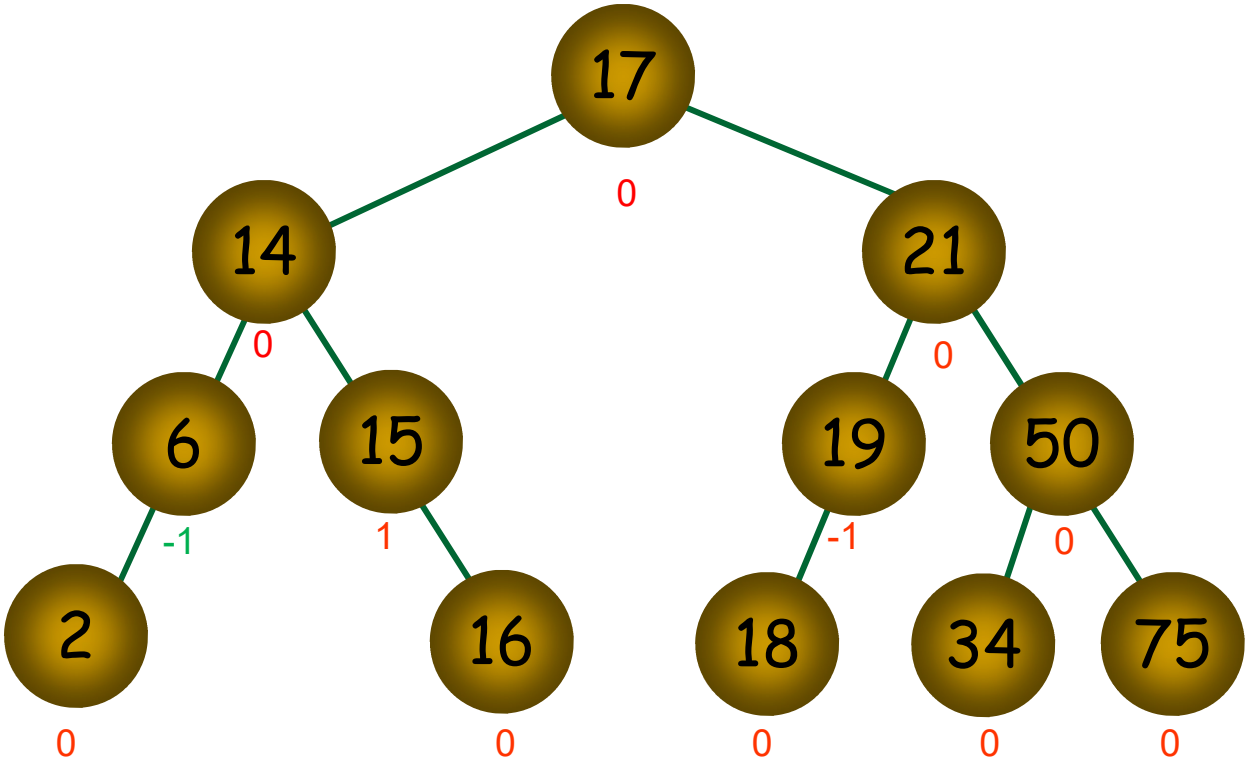


borrado en los AVL

Borrar 7

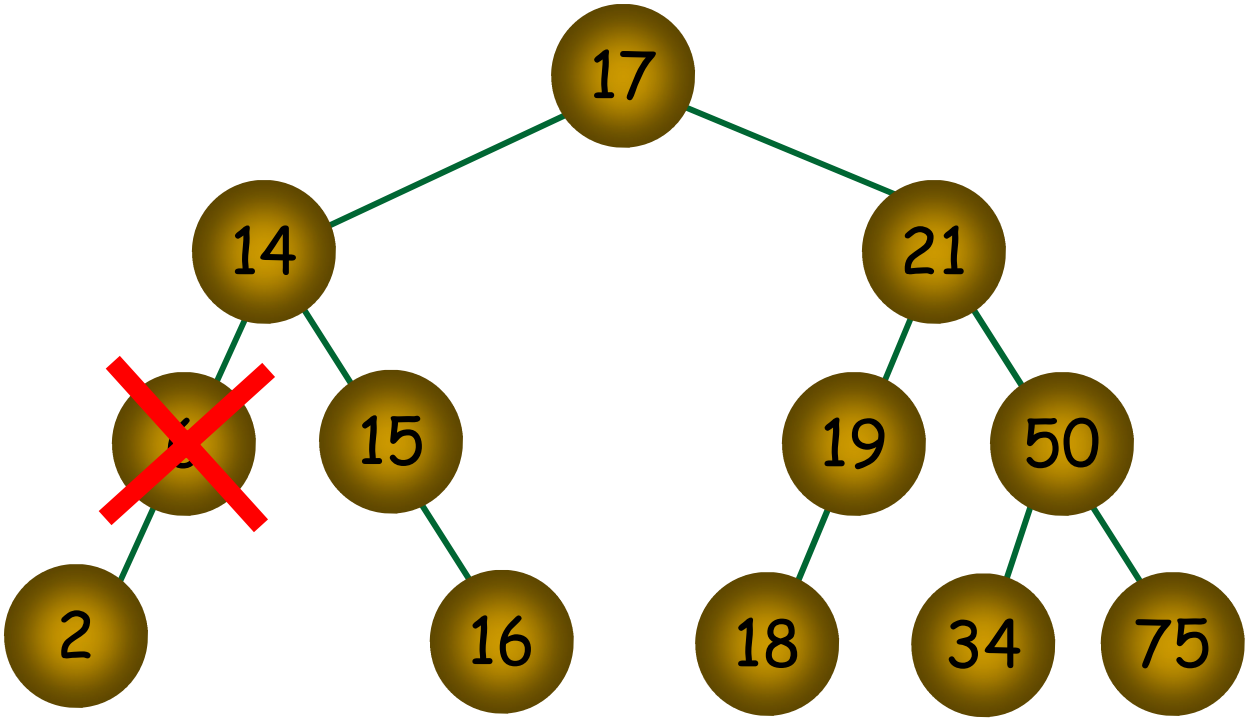


borrado en los AVL

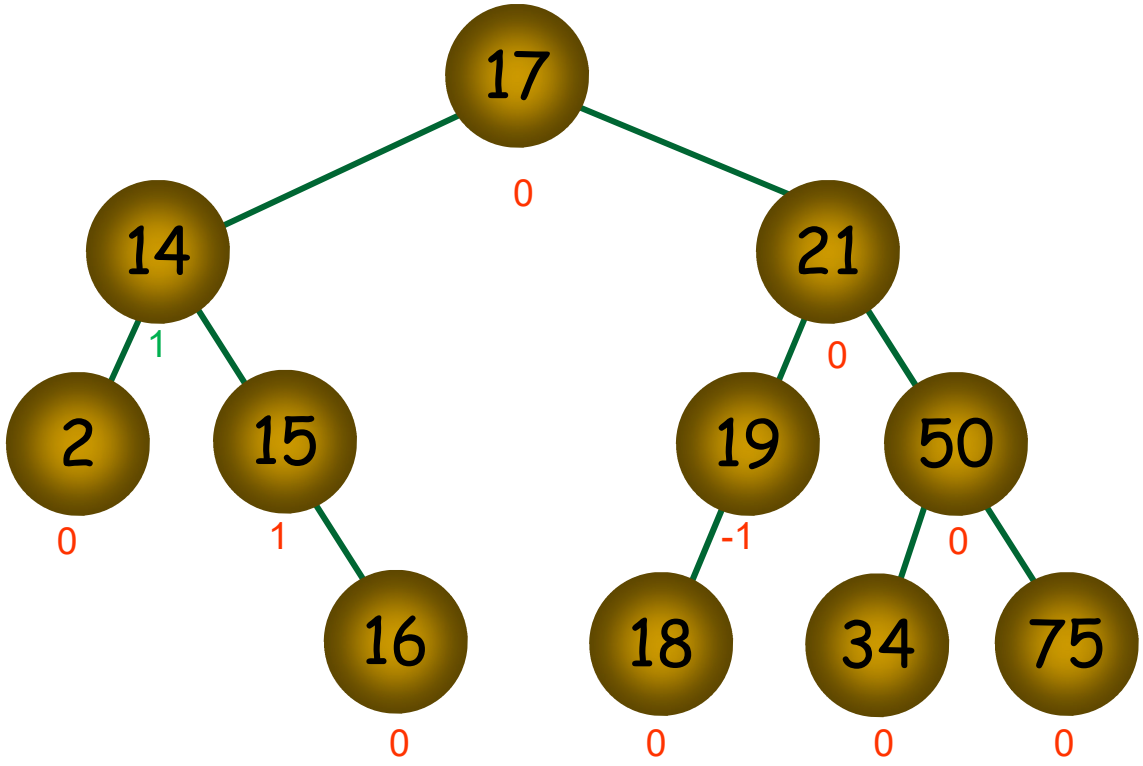


borrado en los AVL

Borrar 6

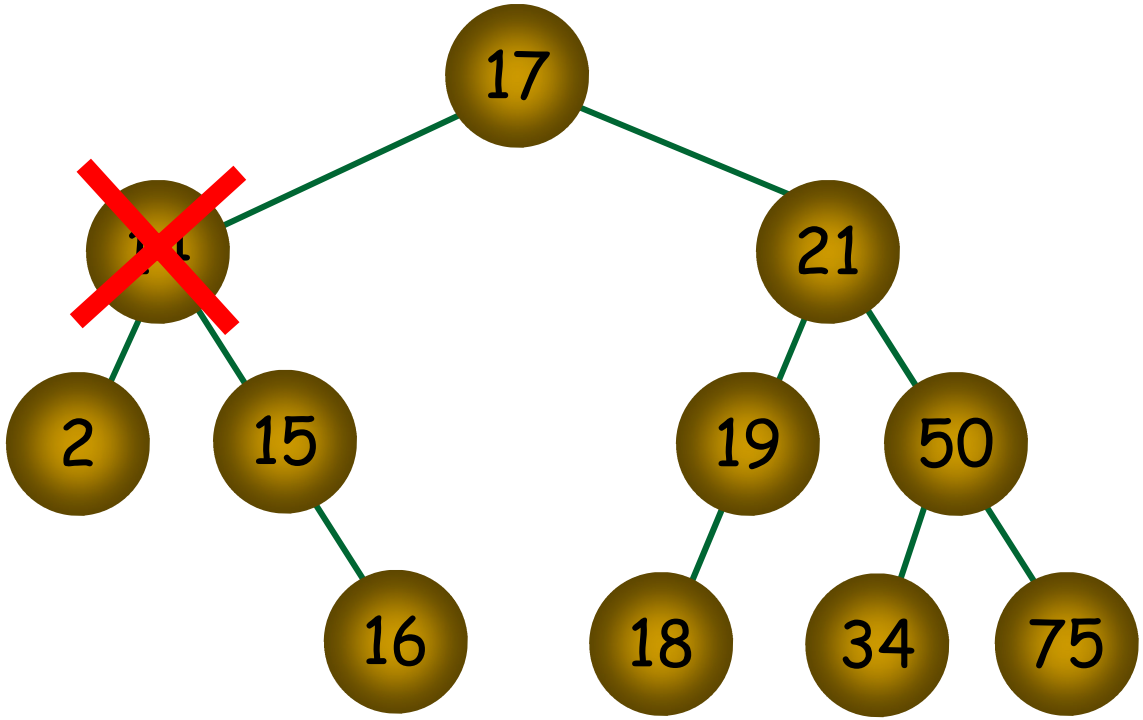


borrado en los AVL

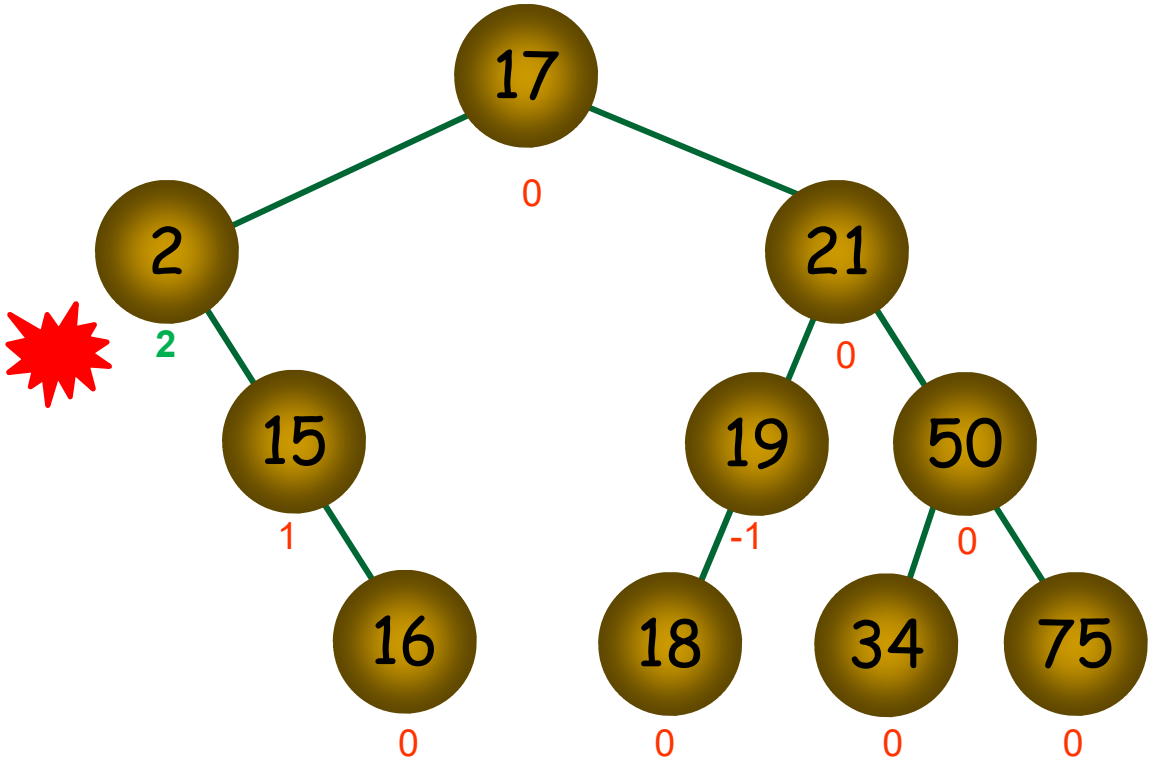


borrado en los AVL

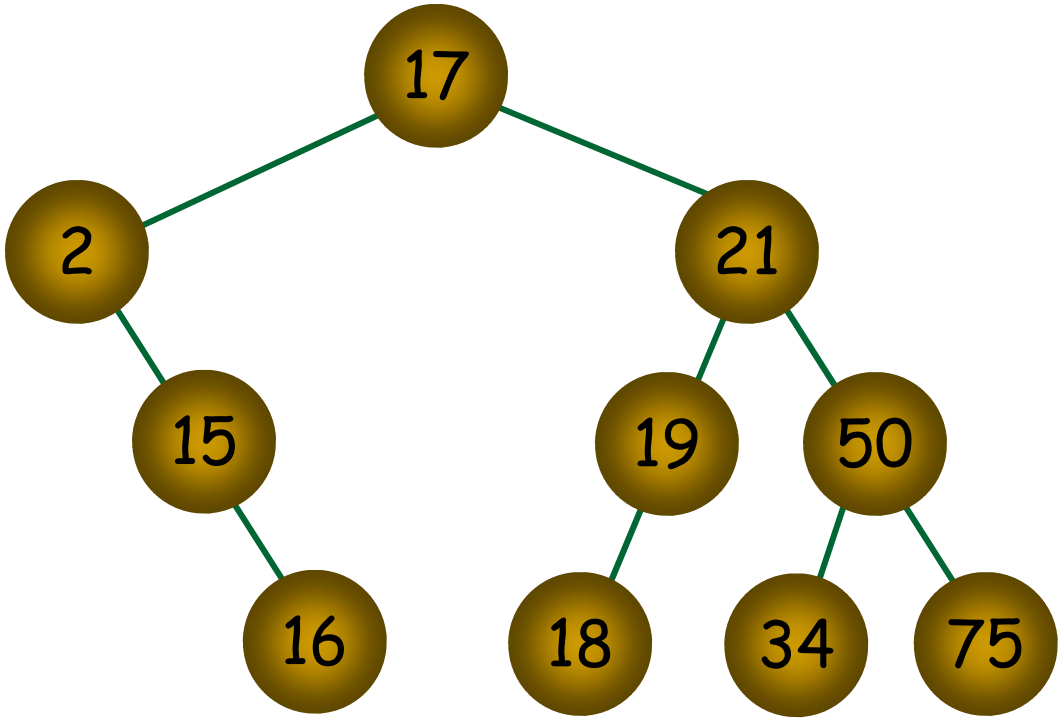
Borrar 14



borrado en los AVL

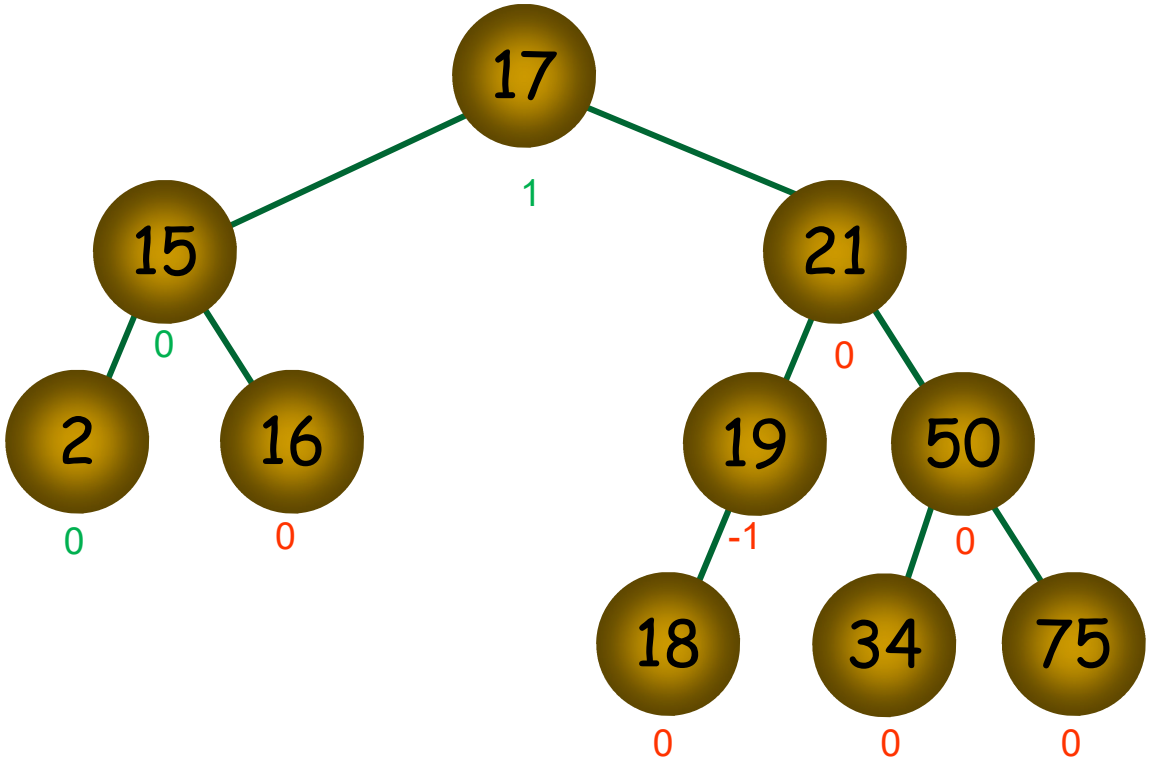


borrado en los AVL



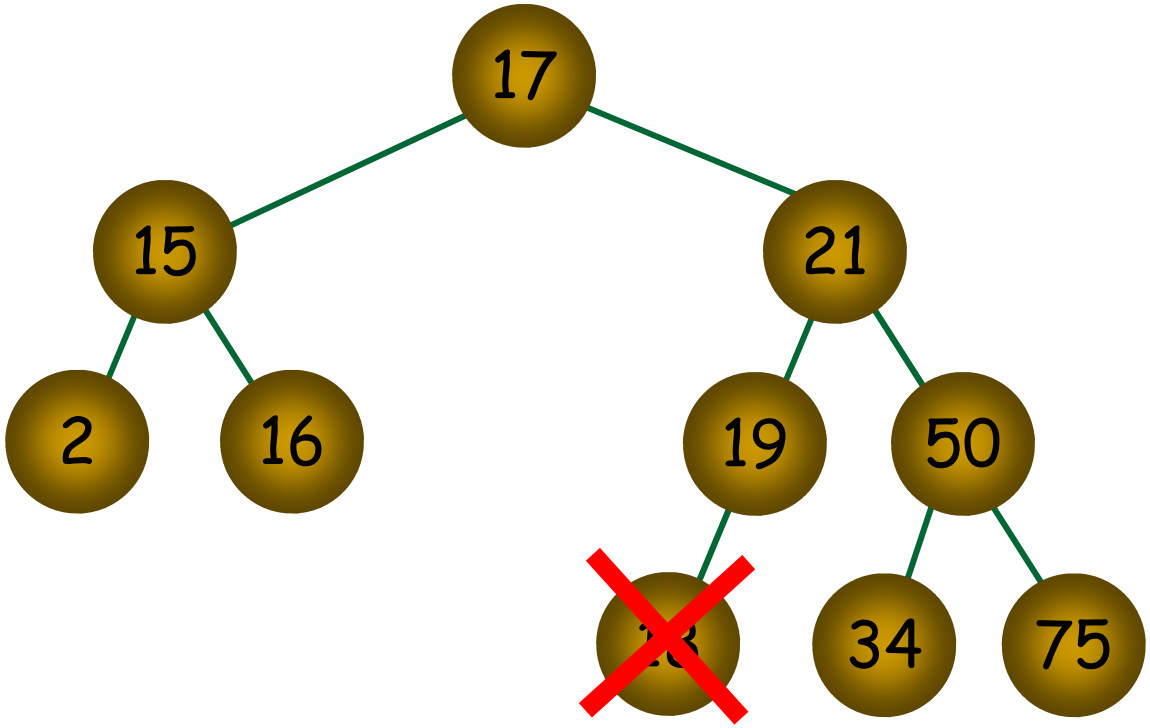
Balanceo RR:
rotación a izquierda(2)

borrado en los AVL

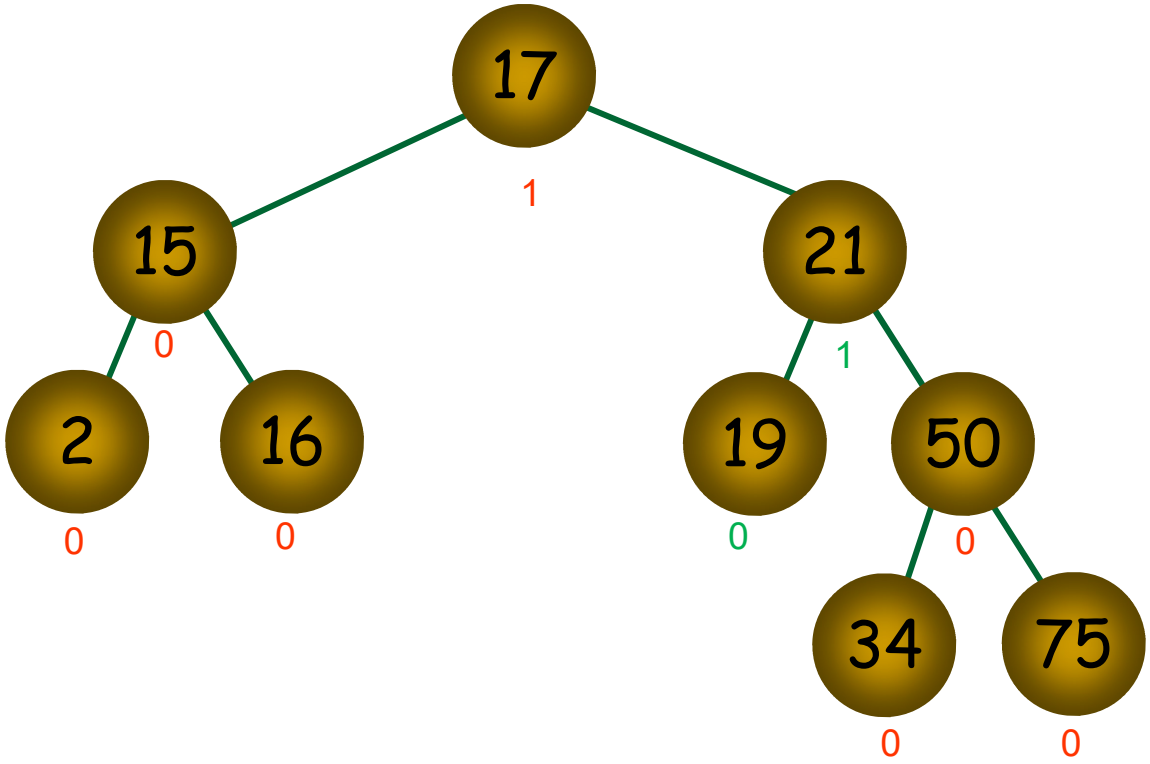


borrado en los AVL

Borrar 18

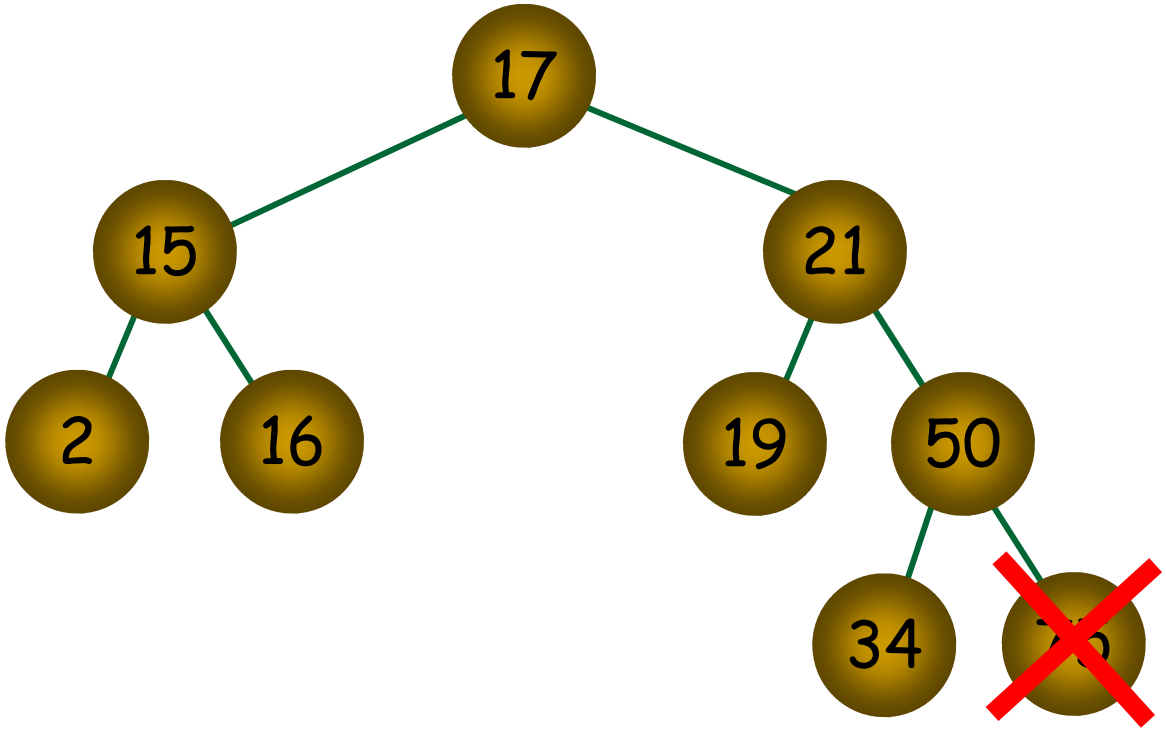


borrado en los AVL

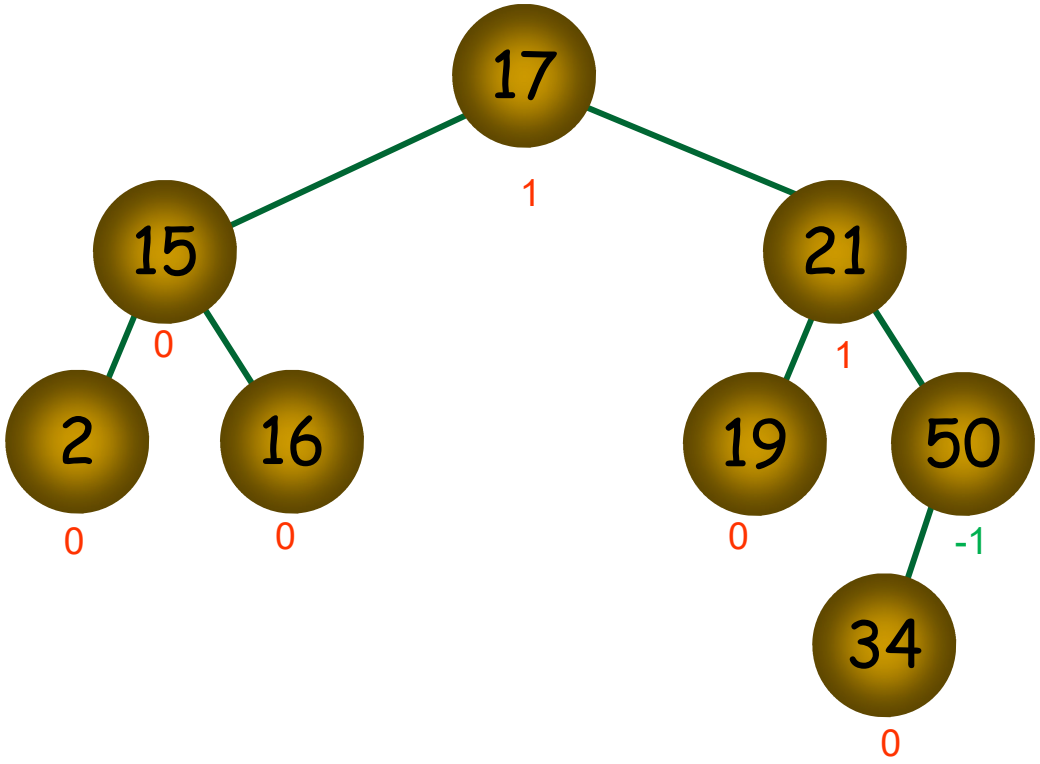


borrado en los AVL

Borrar 75

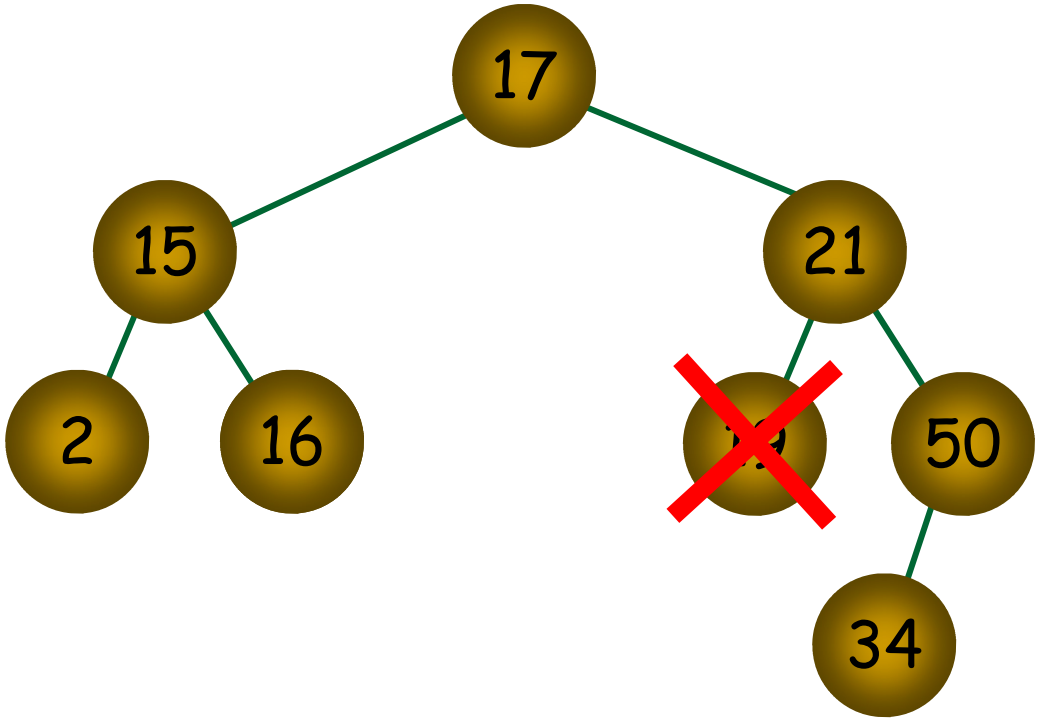


borrado en los AVL

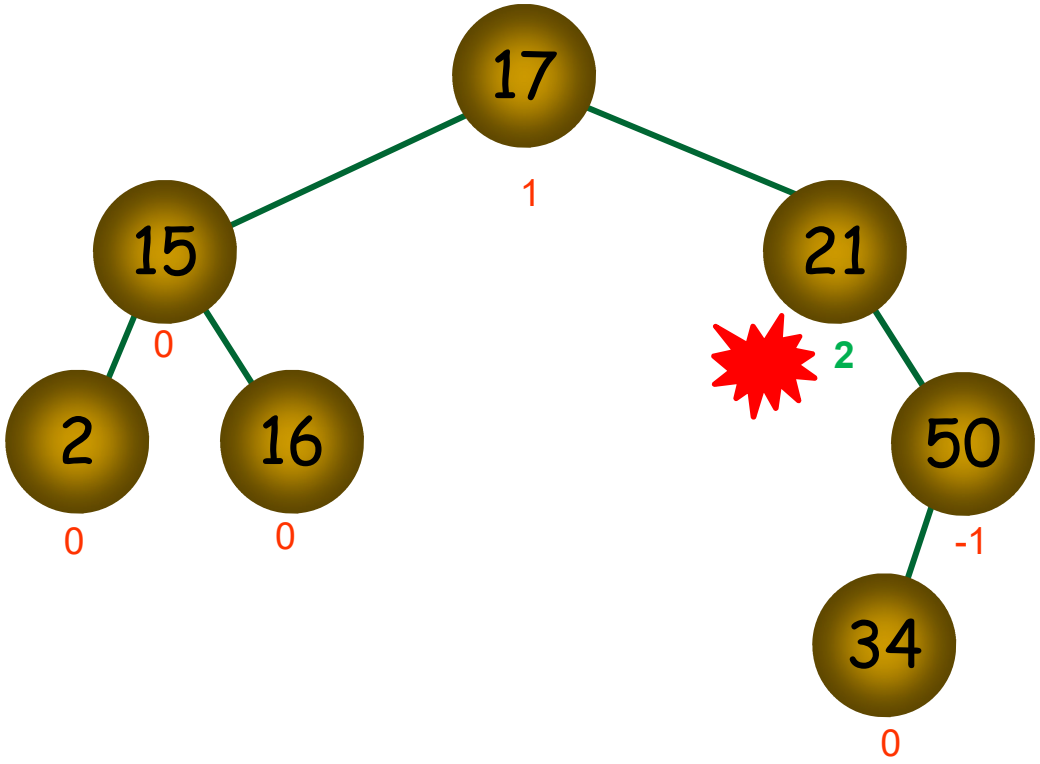


borrado en los AVL

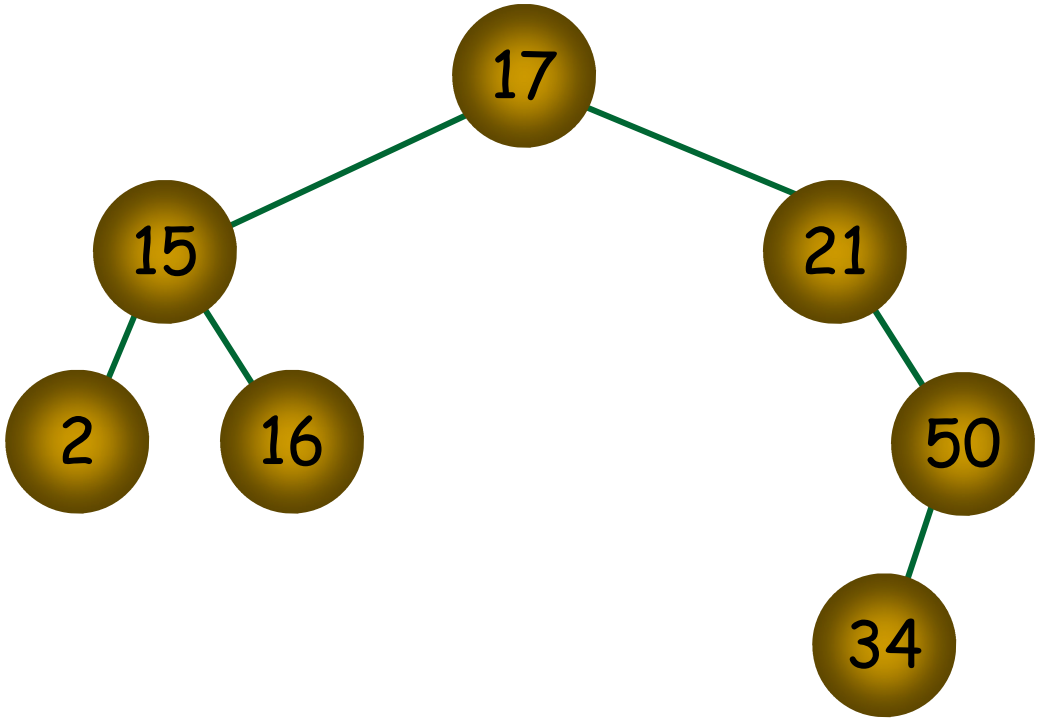
Borrar 19



borrado en los AVL

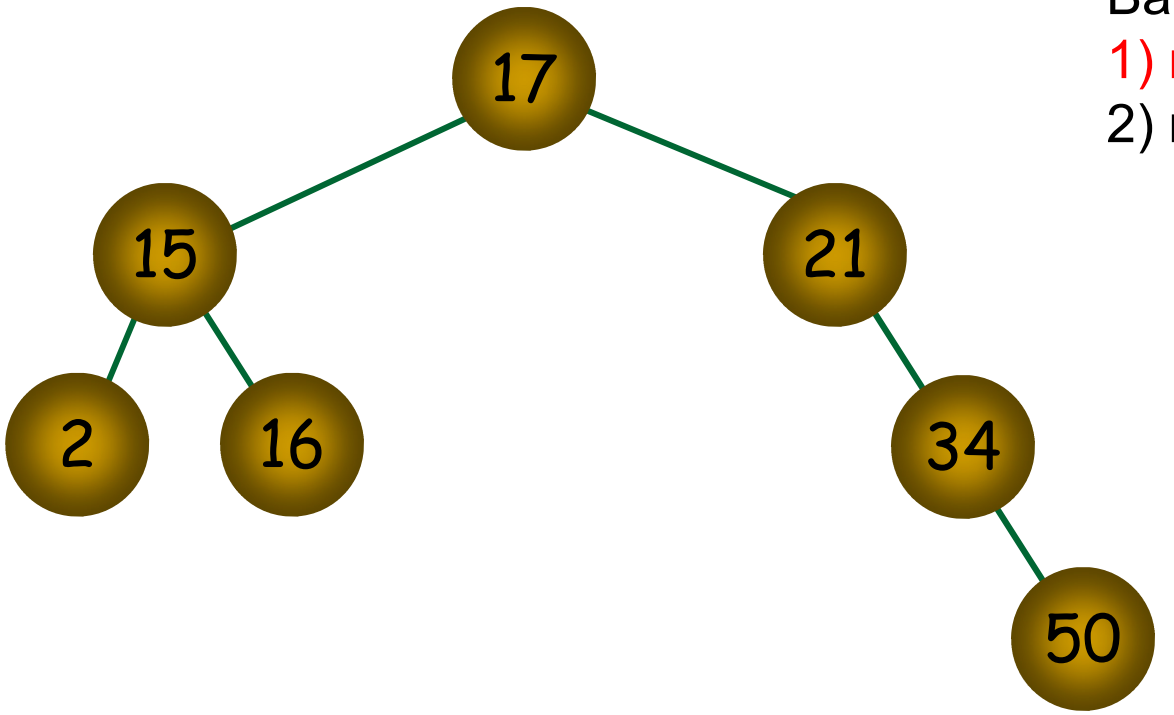


borrado en los AVL



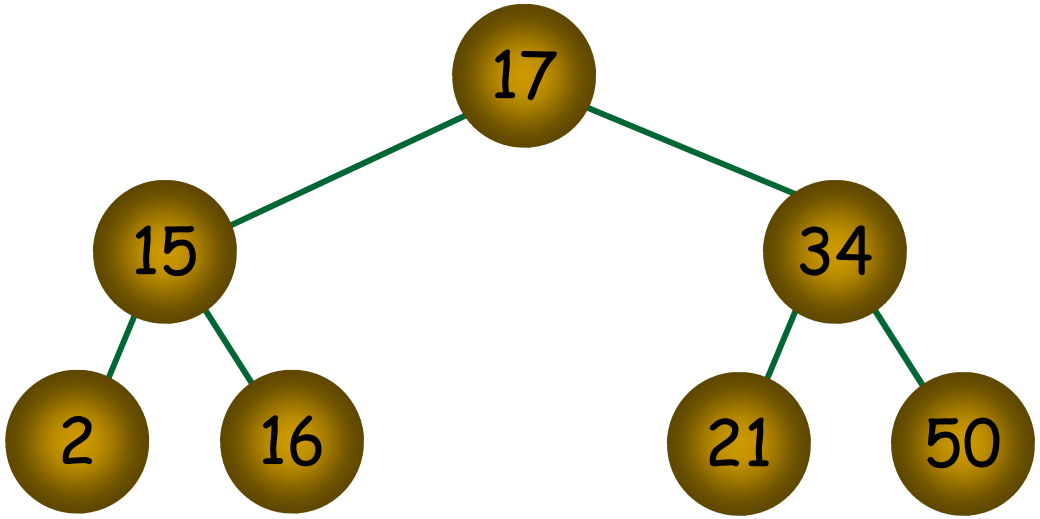
Balanceo LR:
1) rotación a derecha(50)
2) rotación a izquierda(21)

borrado en los AVL



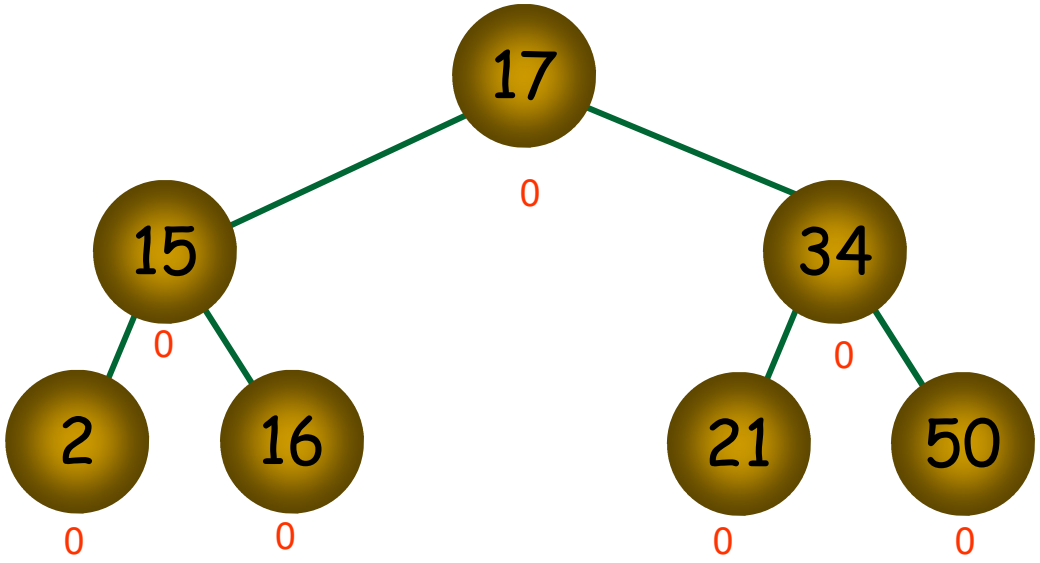
Balanceo LR:
1) rotación a derecha(50)
2) rotación a izquierda(21)

borrado en los AVL

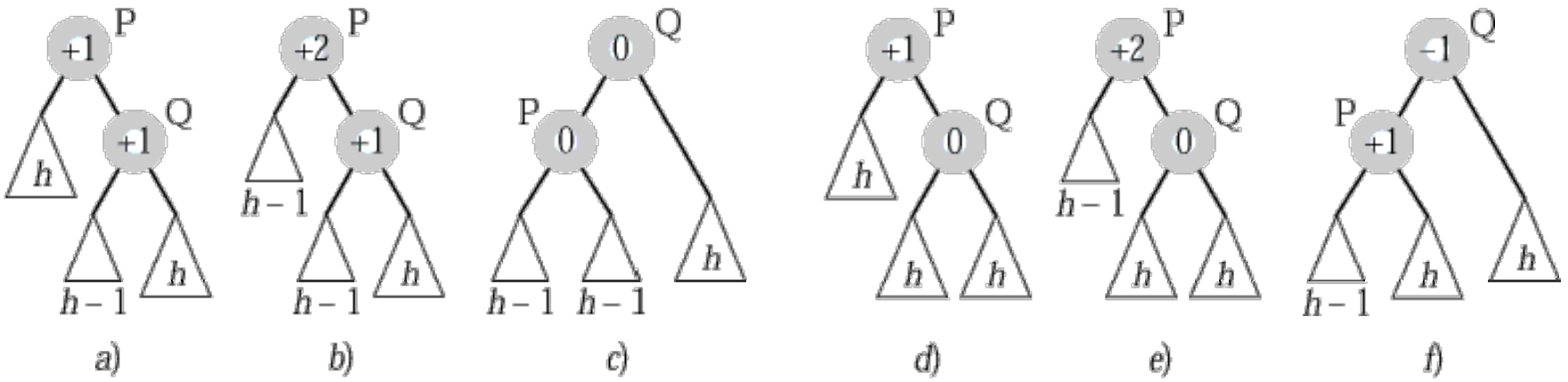


Balanceo LR:
1) rotación a derecha(50)
2) rotación a izquierda(21)

borrado en los AVL

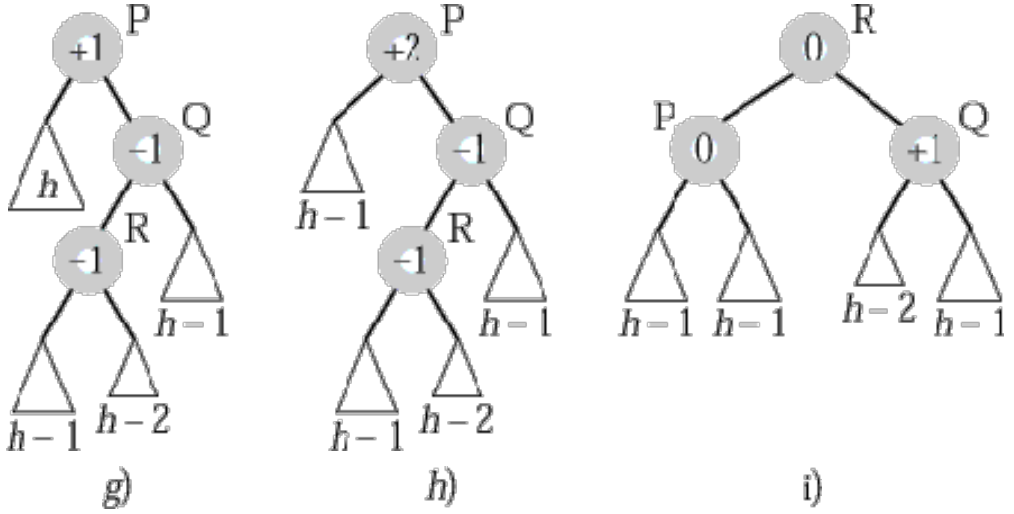


rotación simple



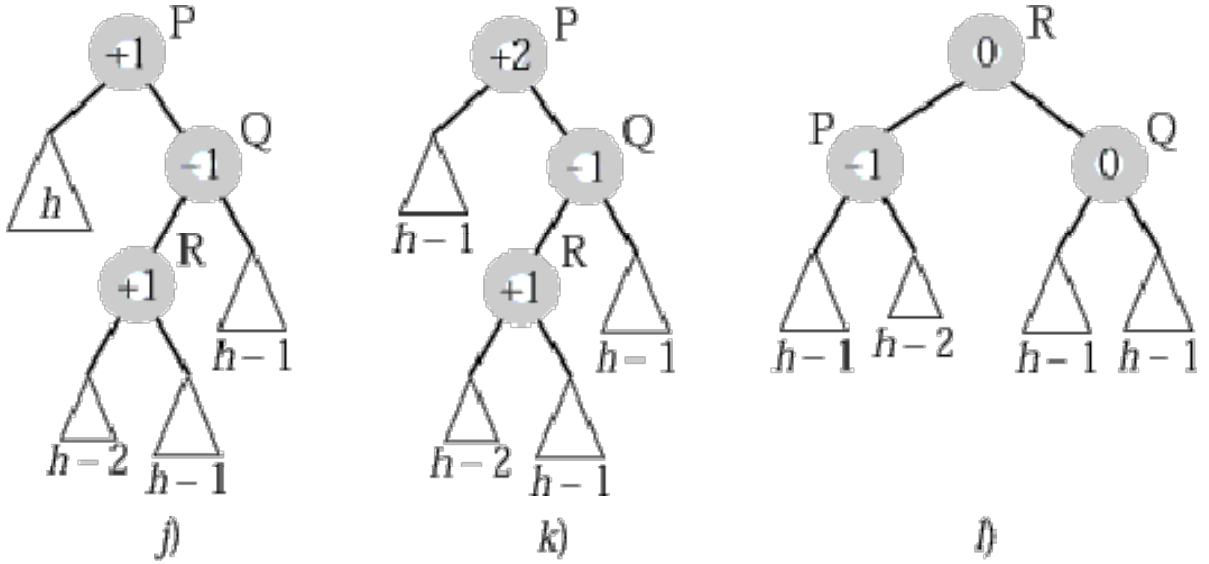
- Eliminación de una hoja de un subárbol izquierdo de P
  - el hijo derecho tiene FDB +1; a), b) y c)
  - el hijo derecho tiene FDB 0; d), e) y f)

rotación doble



- Eliminación de una hoja del subárbol izquierdo de P
  - FDB(Q) = –1 y FDB(R)= –1; g), h) e i)
  - rotación R-Q (P queda en +2, R y Q pasan a +1) y rotación P-R

rotación doble/2



- Eliminación de una hoja del subárbol izquierdo de P
  - FDB(Q) = -1, FDB(R)=+1, j), k) y l)
- rotación R-Q (P queda en +2, R pasa a +2 y Q pasa a 0) y rotación P-R

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: $\Theta(\lg n)$