Árboles AVL

Lección 14

Bibliografía

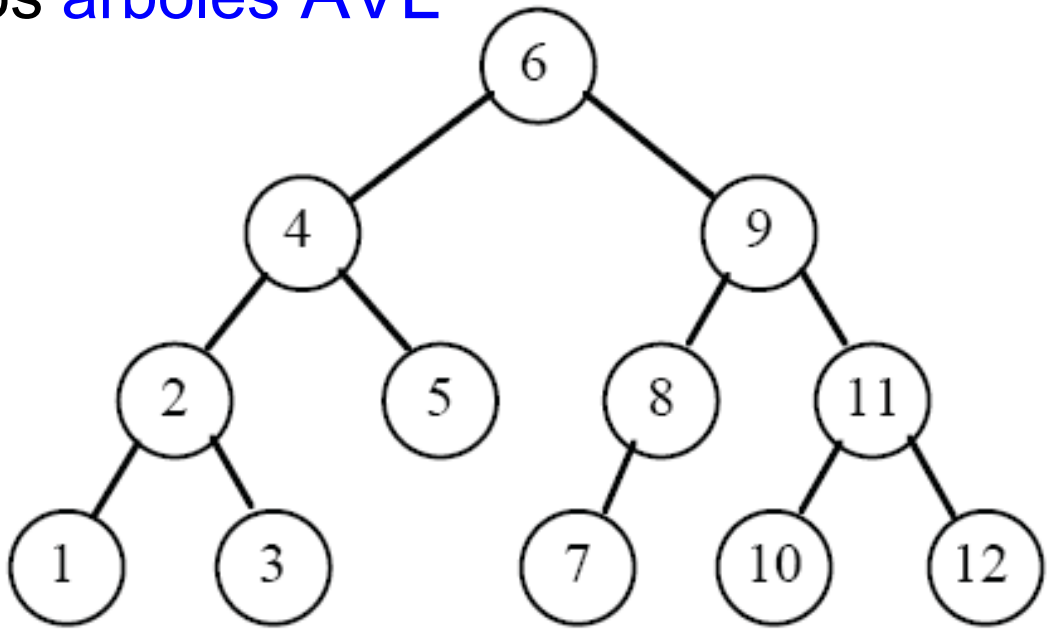
- *Sección 4.5, páginas 163-171, del libro "N. Wirth: Algorithms and Data Structures. 1985" (Oberon versión 2004, disponible como parte de la bibliografía del proyecto Oberon, en la página Wirth: https://people.inf.ethz.ch/wirth/AD.pdf (https://informatika-21.ru/ADen/AD2012.pdf)*
- *Capítulo 7 del libro "Z. J. Hernández, J. C. Rodríguez, etc .: Fundamentos de Estructuras de Datos. Soluciones en Ada, Java y C++. Thomson Paraninfo, 2005"*
- *Capítulo 9 del libro: "Joyanes, L., Zahonero, I., Fernández, M. y Sánchez, L.: Estructuras de datos. Libro de problemas, McGraw Hill, 1999.",*
- *Capítulo 5 del libro "X. Franch: Estructuras de datos. Especificación, diseño e implementación, 3ª edición, Ediciones UPC, 2001"*
- *Tema III (lección 14) del libro de apuntes "Campos Laclaustra, J.: Apuntes de Estructuras de Datos y Algoritmos, segunda edición, 2018* (versión 4, 2022).
- *Y muchos otros....*

Esquema

- Árboles equilibrados y árboles AVL
- Teorema de AVL
- Equilibrado de árboles AVL
- Implementación de los árboles AVL

Árboles Binarios Equilibrados

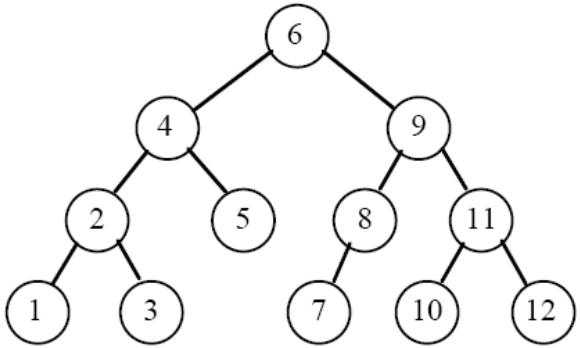
- Un **árbol binario de búsqueda**[1] se dice **equilibrado** (o balanceado) si y solo si, para cada uno de sus nodos ocurre que las alturas de sus dos subárboles difieren como mucho en 1
  (definición dada por Adelson-Velskii y Landis, en 1962)
  → Denominados árboles AVL



1) Árboles binarios de búsqueda **sin repetidos**

Árboles Binarios Equilibrados

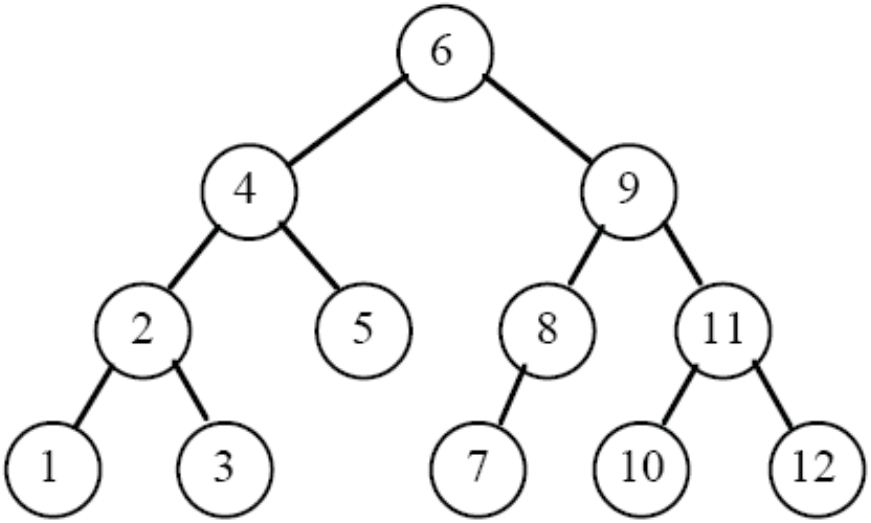
- Árboles balanceados o equilibrados:
  - Un árbol binario de búsqueda[1] es ***k-equilibrado*** si cada nodo lo es.
  - Un nodo es **k-equilibrado** si las alturas de sus subárboles izquierdo y derecho difieren en no más de **k**. (*Si k=0 sería equilibrio perfecto)*
- Árboles AVL (Adelson-Velskii, Landis):
  - Un árbol binario de búsqueda 1-equilibrado se llama árbol AVL.



1) Árboles binarios de búsqueda sin repetidos

Árboles AVL

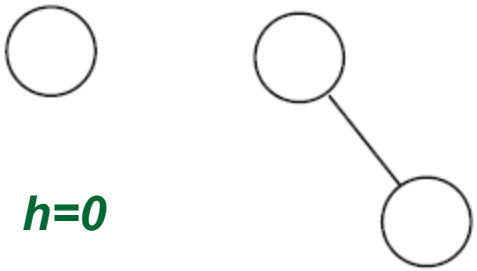
- En un árbol AVL, siendo N el número de nodos del árbol :
  - Se garantiza que la **altura del árbol** es mínima, o casi: **Θ(log N)** (por el Teorema de A-V y L)
  - el coste de las operaciones de **inserción, búsqueda y borrado**, en el peor caso, es **Θ(log N)**



Teorema de A-V y L

- ¿Cuál es el número mínimo de nodos en un árbol AVL de altura h?

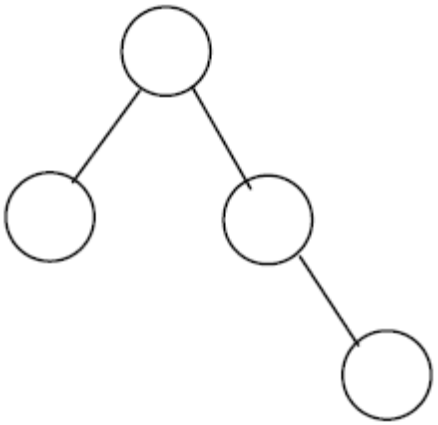
→ ¿ Cuál es la estructura o forma del *peor* árbol AVL para una determinada altura?



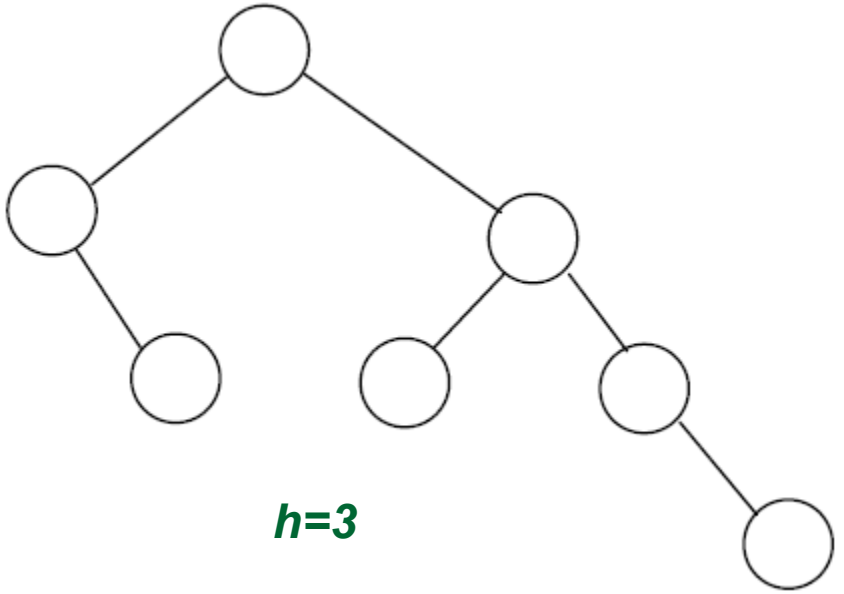
h=0

h=1

¿y para h=4?



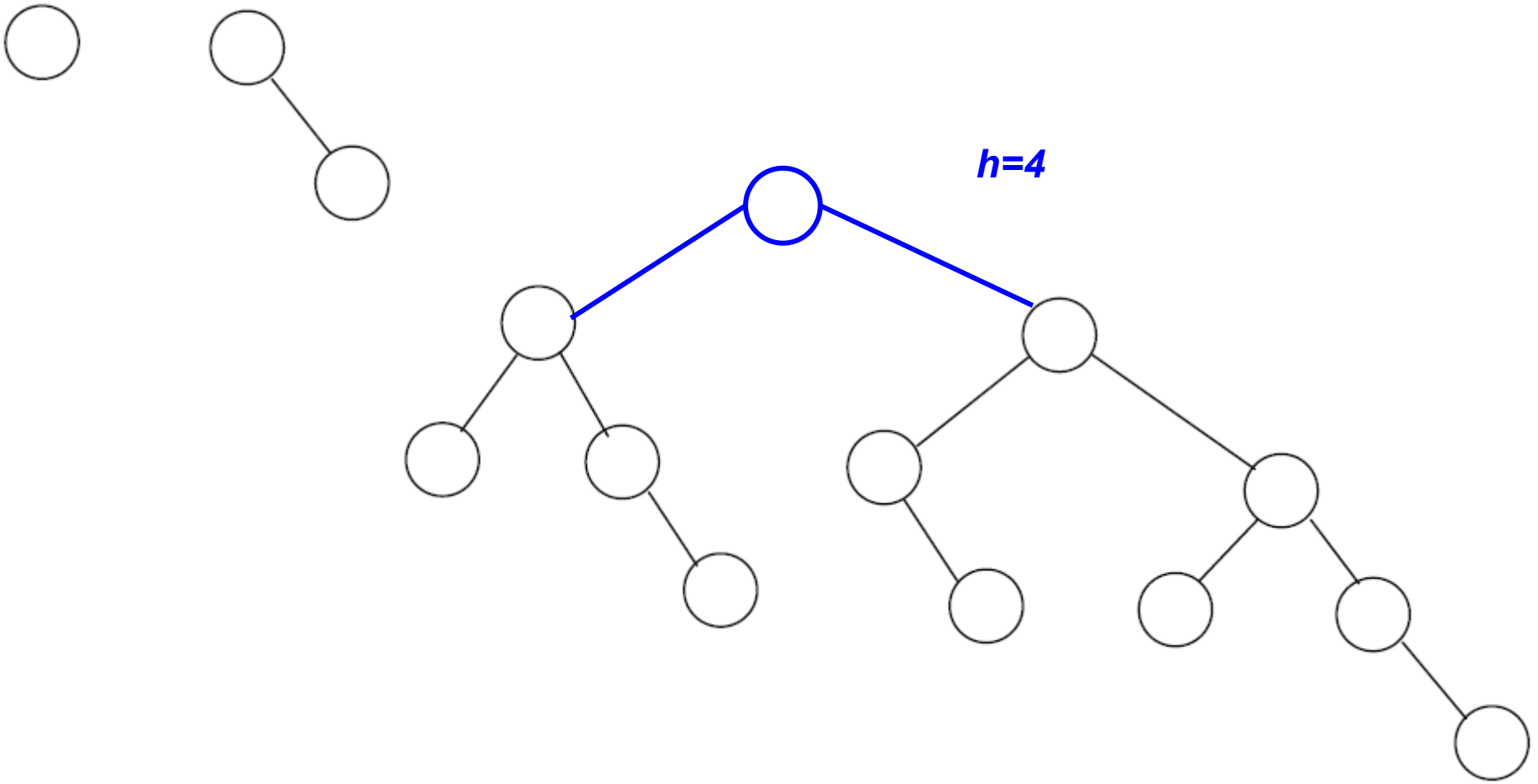
h=2



h=3

Teorema de A-V y L

- Número mínimo de nodos en un árbol AVL de altura h:

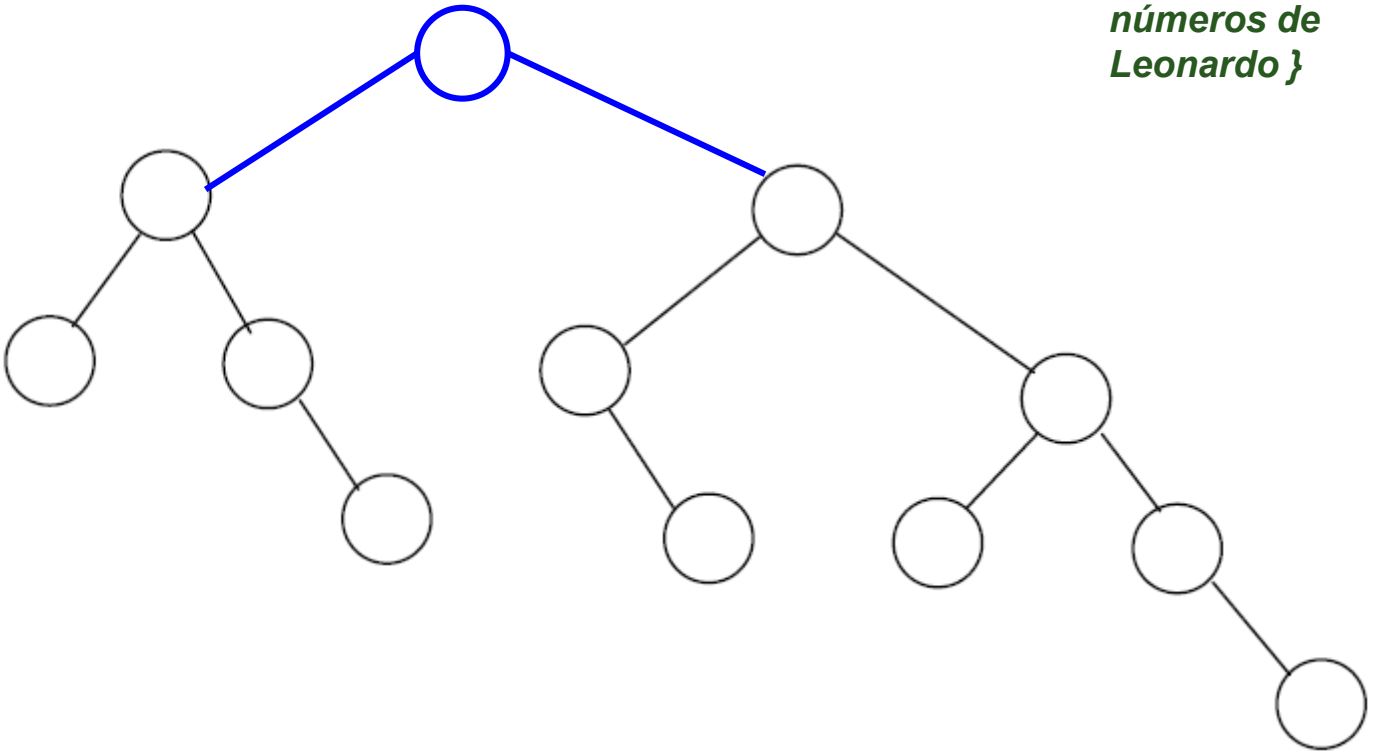


Teorema de A-V y L

- *Gn* = nº nodos en el AVL de altura *n* que tiene el mínimo número posible de nodos

Entonces: $G_n = G_{n-1} + G_{n-2} + 1$

{ Serie conocida como los números de Leonardo }



Teorema de A-V y L

{ Más detalles en el libro Algorithms and Data Structures (2004) de N. Wirth, o en The Art of Computer Programming, Vol. 1 de Knuth }

- Por su semejanza y relación con la serie de Fibonacci:

$$F_n = F_{n-1} + F_{n-2}; \quad F_0 = 1 \; ; \; F_{-1} = 0$$

- A los árboles AVL construidos con el menor número de nodos posible para una altura dada, se les denomina **árboles de Fibonacci**

n	G_n	F_n
0	1	1
1	2	1
2	4	2
3	7	3
4	12	5
5	20	8
6	33	13
…	…	…

Se cumple que: $G_n = F_{n+2} - 1$

Y se sabe que: $F_n > (\phi^n / \sqrt{5}) - 1$, con $\phi = (1 + \sqrt{5})/2$

Por tanto: $G_n > (\phi^{n+2} / \sqrt{5}) - 2$

ϕ: número áureo (fi)

es decir, el nº de nodos N de un árbol AVL de altura h verifica: $N \geq G_h > (\phi^{h+2} / \sqrt{5}) - 2$

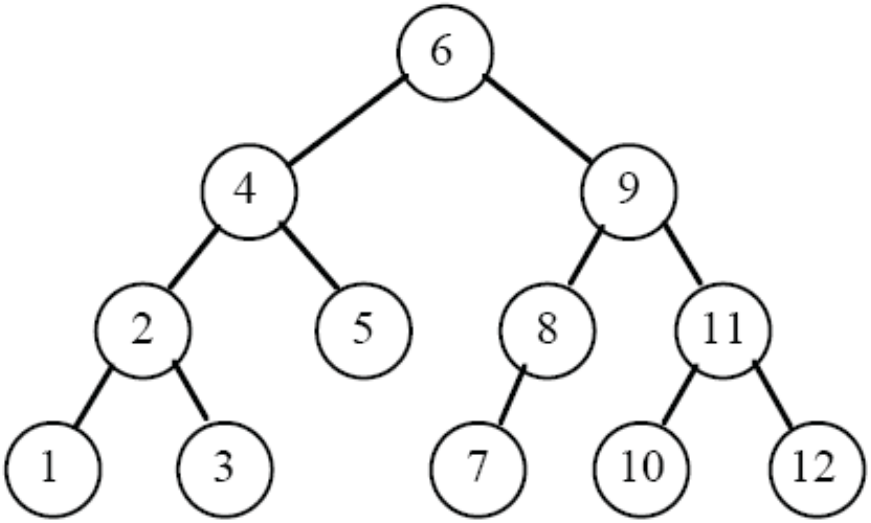
Teorema de A-V y L(1962): $h < 1'4404 \log_2(N+2) - 0'328$

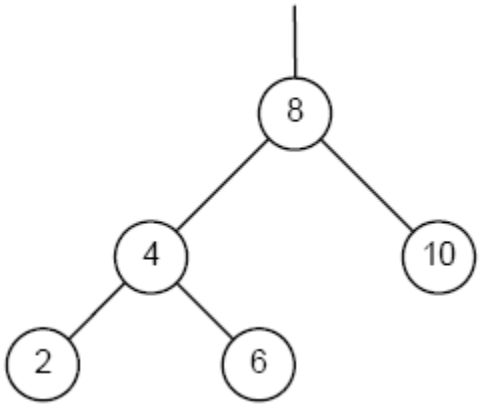
"La altura máxima de un árbol AVL está acotada a aproximadamente 1,5logN"

Árboles AVL

- En un árbol AVL, siendo N el número de nodos del árbol:
  - √ Se garantiza que la **altura del árbol** es mínima, o casi: **Θ(log N)**
  - √ El **coste de las operaciones** de **búsqueda, inserción y borrado,** en el peor caso, es Θ(log N) → *visto en los ABBs*

  →¿Cómo mantener el árbol como un AVL?



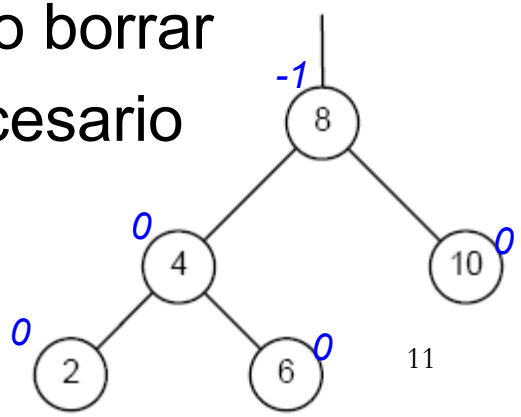


Equilibrado de árboles AVL

Factor de equilibrio (F_e)=

= (Altura subárbol Derecho – Altura subárbol Izq.)

- Algoritmos más simples que considerando las alturas de los árboles
- Si el factor de equilibrio de un nodo es:
  - *0* → el nodo está equilibrado, y sus subárboles tienen exactamente la misma altura (*perfectamente equilibrado*)
  - *1* → el nodo está equilibrado, pero su subárbol derecho es un nivel más alto (*pesado a la derecha*)
  - *-1* → el nodo está equilibrado, pero su subárbol izquierdo es un nivel más alto (*pesado a la izquierda*)
- Recalcular el factor de equilibrio al insertar o borrar
- Si el factor de equilibrio $F_e \geq 2$ o $F_e \leq -2$ es necesario ***reequilibrar***



Equilibrado de árboles AVL

- **Es necesario guardar información sobre el equilibrio en cada nodo**

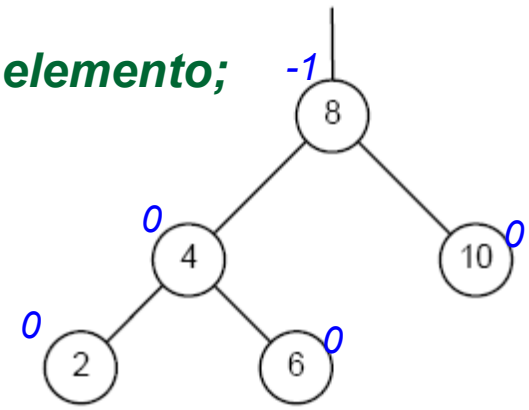
- **Estructura de datos:**

tipo avl = ↑nodo;

factor_equilibrio =(pesado_izq, equilibrado, pesado_dch); -- {-1, 0, 1}

```
nodo = registro
    laClave: clave;
    elValor: valor;
    equilibrio: factor_equilibrio;
    izq, dch: avl;
freg
```

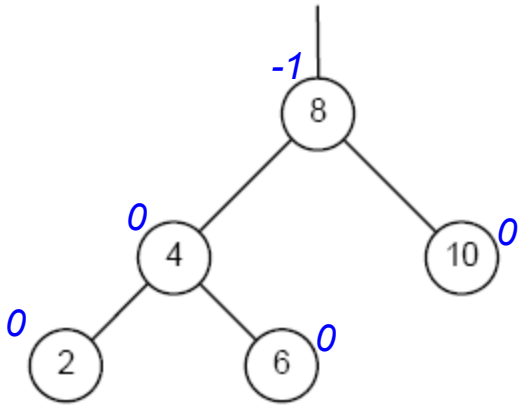
podrían ser un campo dato: elemento;



{Implementación de Diccionario: No se insertarán claves repetidas, se actualizarán sus valores}

Equilibrado de árboles AVL

Proceso de inserción:



Ejemplo:

1) Insertar 15

2) Insertar 1

3) Insertar 9

4) Insertar 0

Equilibrado de árboles AVL

Proceso de inserción:

1. buscar (como en los ABBs) hasta encontrar la posición de inserción o modificación (proceso idéntico a inserción en árbol binario de búsqueda → *se insertará siempre como hoja*)
2. insertar el nuevo nodo (será hoja) con factor de equilibrio: *0 "equilibrado"*
3. desandar el camino de búsqueda, verificando el equilibrio de los nodos del camino, y re-equilibrando si es necesario

- Se implementa mediante recursión:
  - Parámetro booleano, que al retornar **indica si el subárbol ha aumentado su altura** (ha crecido)
    - si no ha aumentado, no cambia el factor de equilibrio
    - **si ha aumentado**, hay varias *posibilidades….*

Equilibrado de árboles AVL

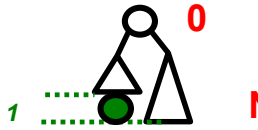
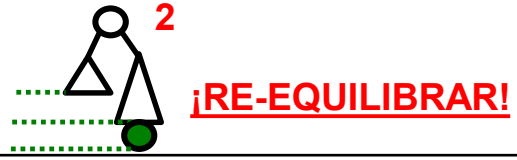
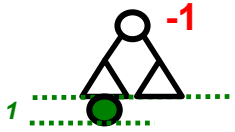
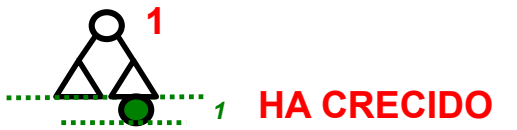
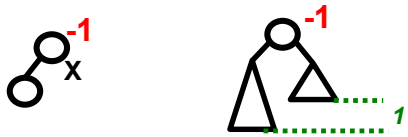
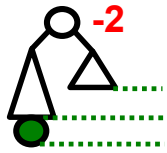
Proceso de inserción:

- (como en los ABBs…) → *se insertará siempre como hoja*

< ANTES: > **DESPUES:**

- desandar el camino de búsqueda (se implementa mediante recursión) al volver de la inserción, si el hijo en el que se ha insertado ha crecido, verificar el factor de equilibrio y si es necesario re-equilibrar.
  Posibles casos:

< ANTES: > **< ---------- DESPUES: ---------- >**

Caso	se insertó en hijo izq (e izq creció)	se insertó en hijo dch (y dch creció)
		
		
		

Equilibrado de árboles AVL

Proceso de inserción:

1. buscar (como en los ABBs) hasta encontrar la posición de inserción o modificación (proceso idéntico a inserción en árbol binario de búsqueda → *se insertará siempre como hoja*)
2. insertar el nuevo nodo (será hoja) con factor de equilibrio: *0 "equilibrado"*
3. desandar el camino de búsqueda, verificando el equilibrio de los nodos del camino, y re-equilibrando si es necesario

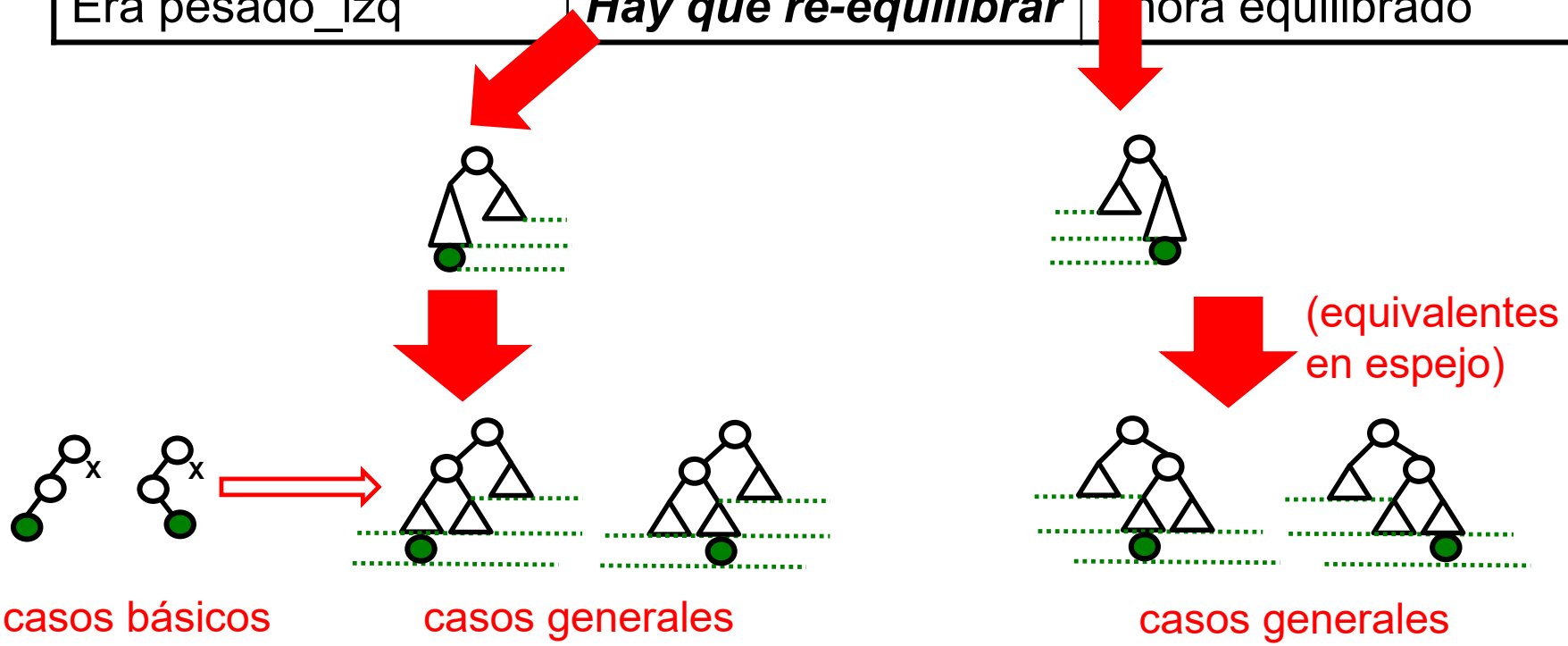
- Se implementa mediante recursión:
  - Parámetro booleano, que al retornar **indica si el subárbol ha aumentado su altura** (ha crecido)
    - si no ha aumentado, no cambia el factor de equilibrio
    - **si ha aumentado**, hay varias *posibilidades:*

Caso	**Se insertó en el izq.**	**Se insertó en el dch.**
Era pesado_dch	Ahora equilibrado	***Hay que re-equilibrar***
Era equilibrado	Ahora pesado_izq	Ahora pesado_dch
Era pesado_izq	***Hay que re-equilibrar***	Ahora equilibrado

Equilibrado de árboles AVL

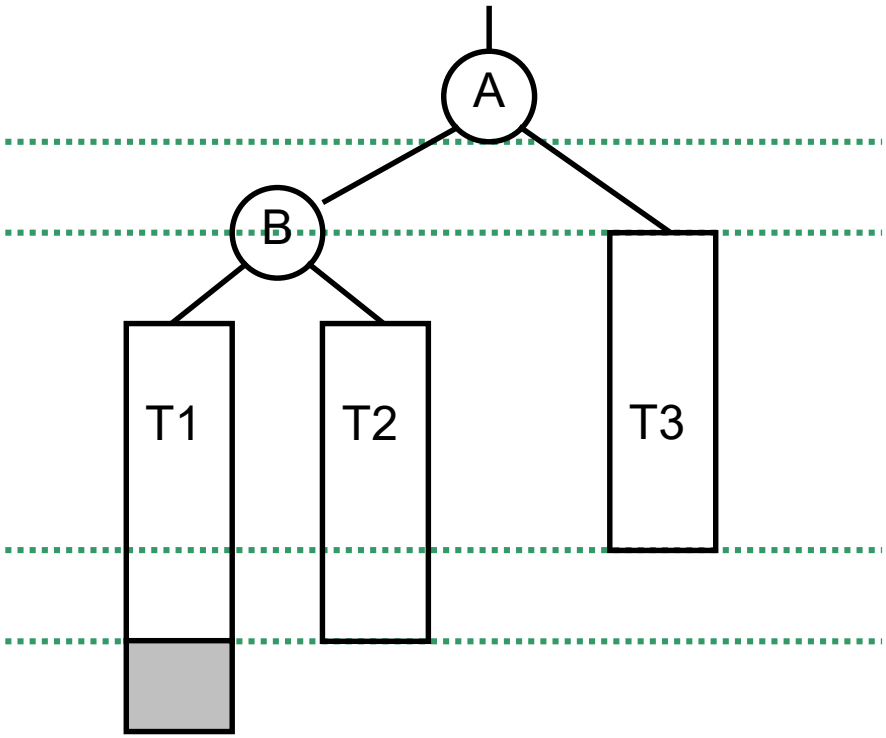
Proceso de inserción:

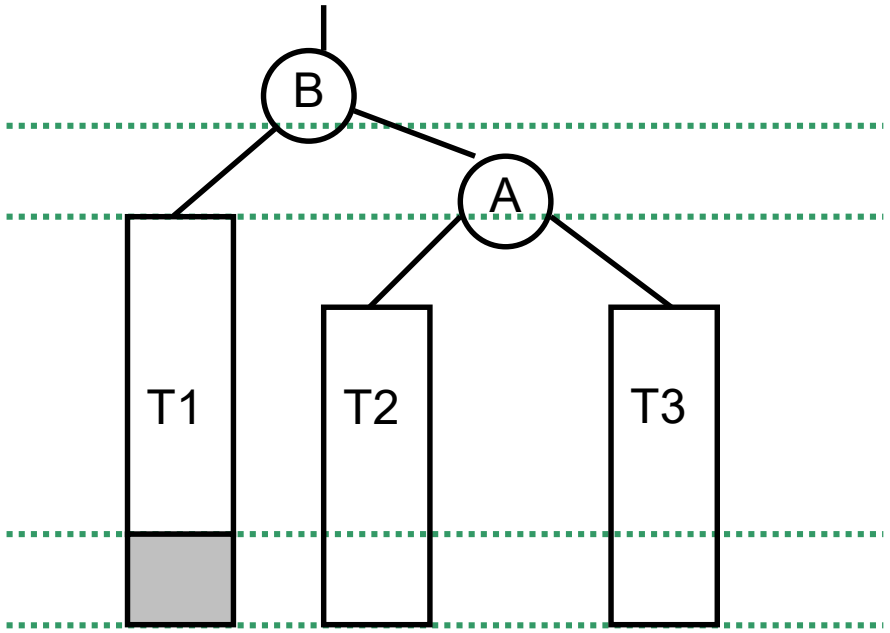
Caso	Se insertó en el izq.	Se insertó en el dch.
Era pesado_dch	Ahora equilibrado	***Hay que re-equilibrar***
Era equilibrado	Ahora pesado_izq	Ahora pesado_dch
Era pesado_izq	***Hay que re-equilibrar***	Ahora equilibrado



Equilibrado de árboles AVL

- Inserción:



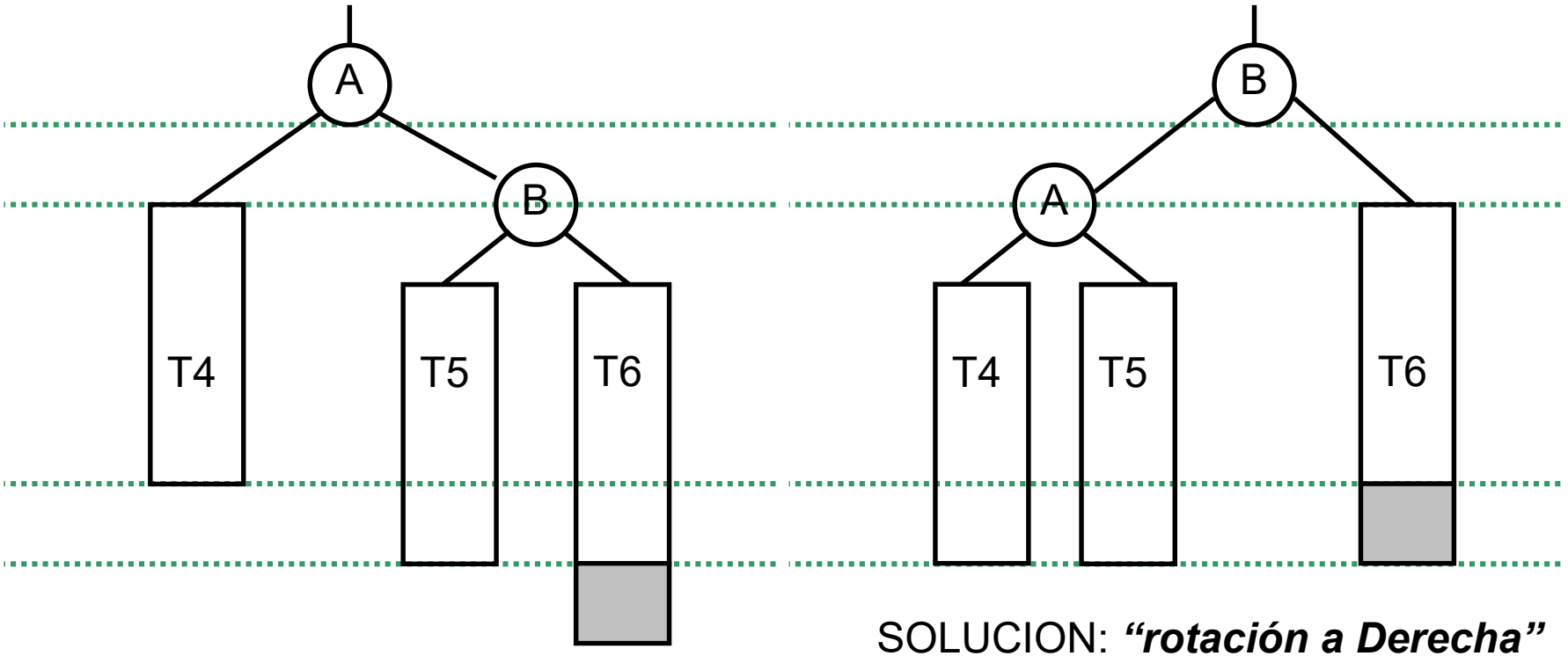


SOLUCION: ***"Rotación a izquierda"***

Caso 1: Izquierda-Izquierda

Equilibrado de árboles AVL

- Inserción:

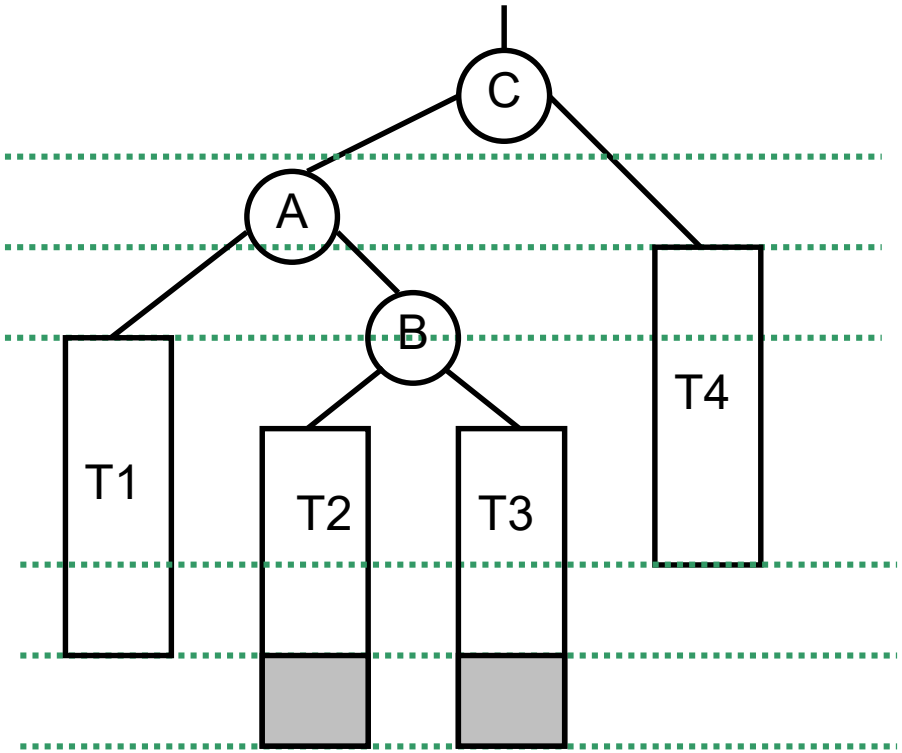


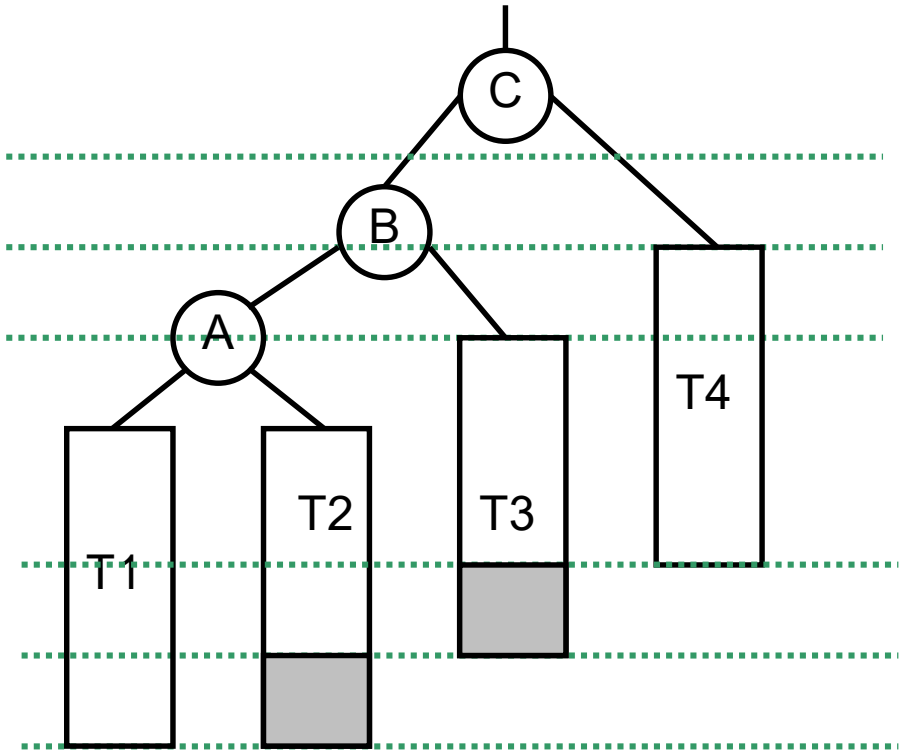
SOLUCION: ***"rotación a Derecha"***

Caso 2: Derecha- Derecha

Equilibrado de árboles AVL

- Inserción:





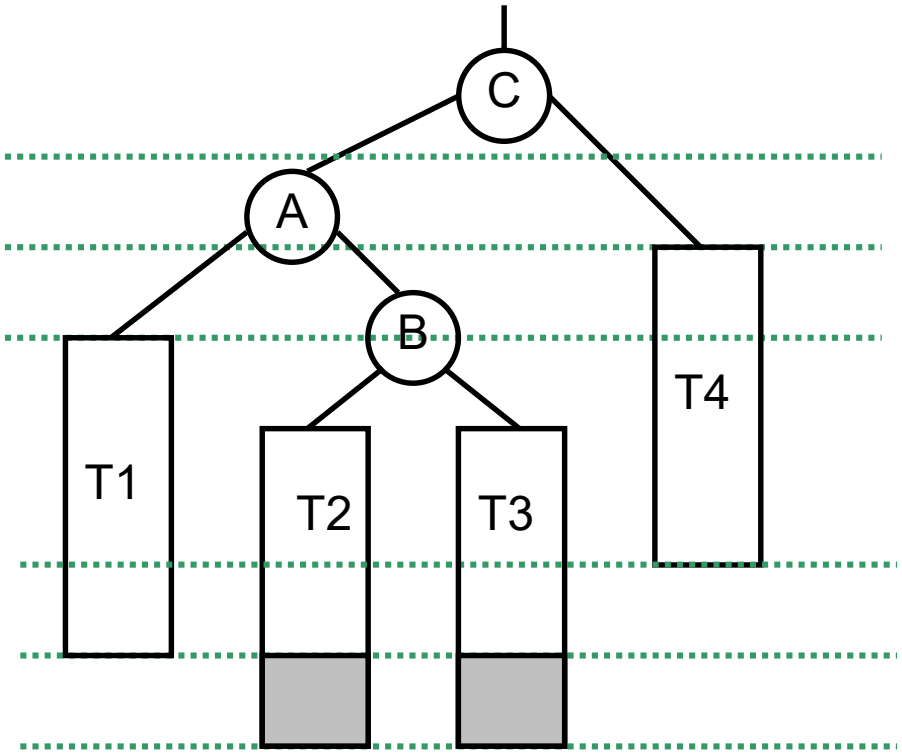
Caso 3: Izq-Dech

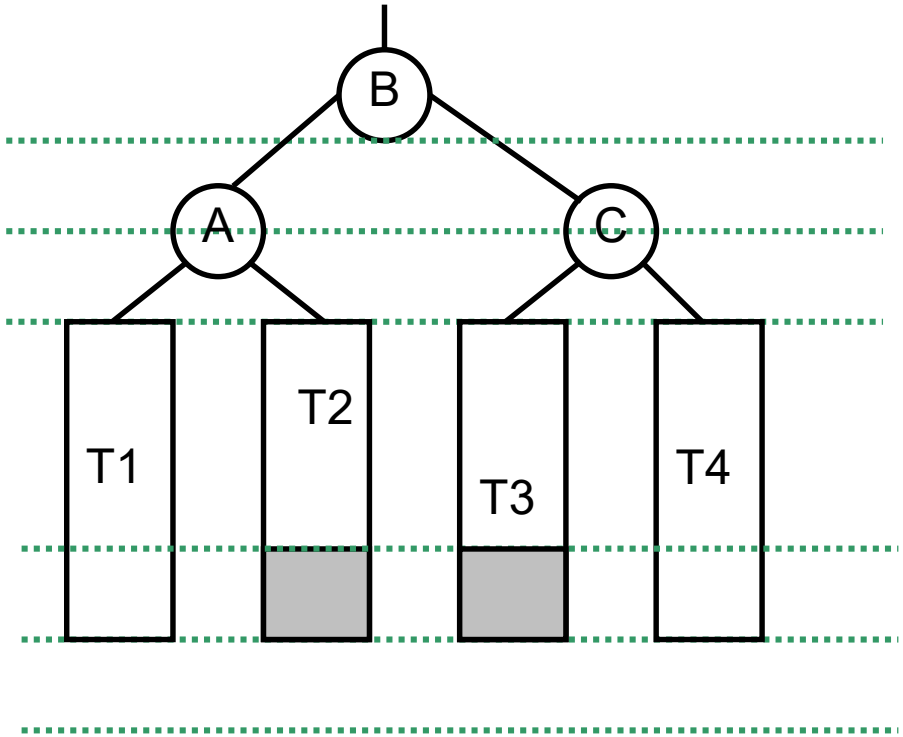
SOLUCION: ***"Rotación Doble"***

PASO 1: ***"Rotación a derecha"***

Equilibrado de árboles AVL

- Inserción:





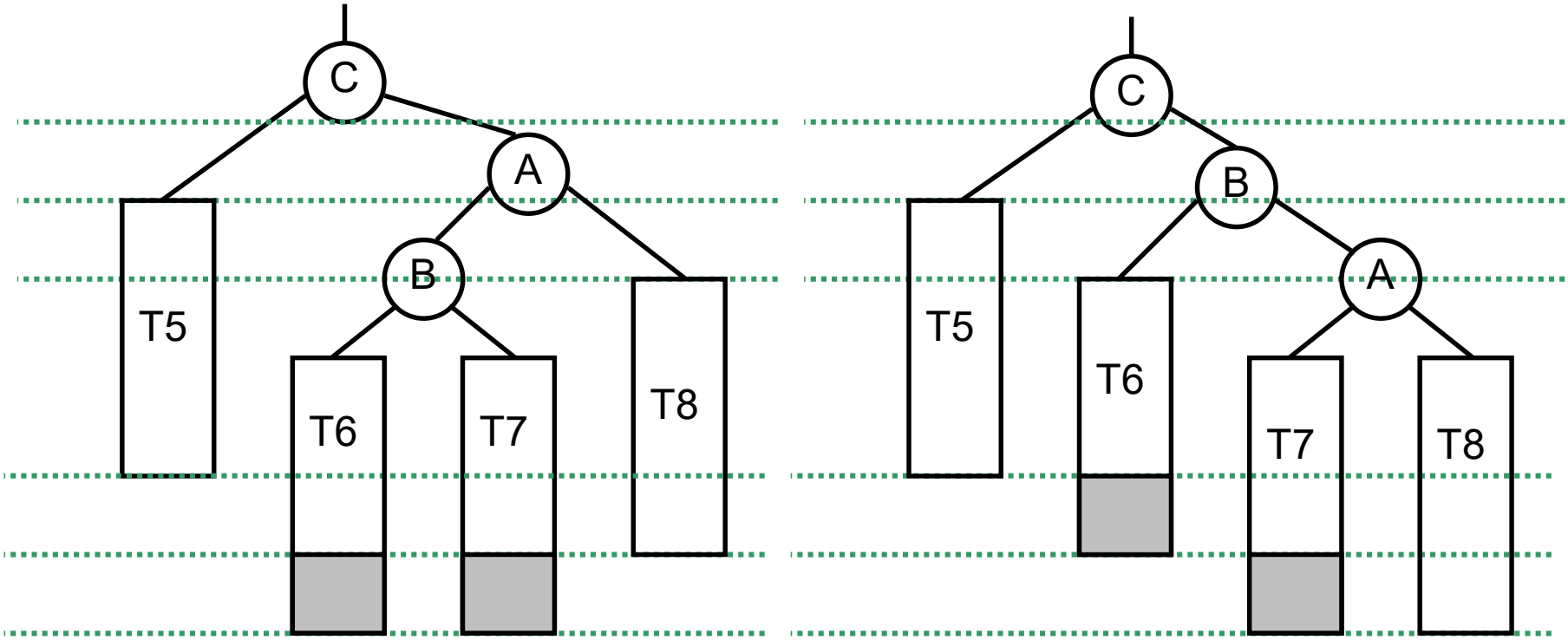
Caso 3: Izq-Dech

SOLUCION: ***"Rotación Doble"***

PASO 2: ***"Rotación a izquierda"***

Equilibrado de árboles AVL

- Inserción:



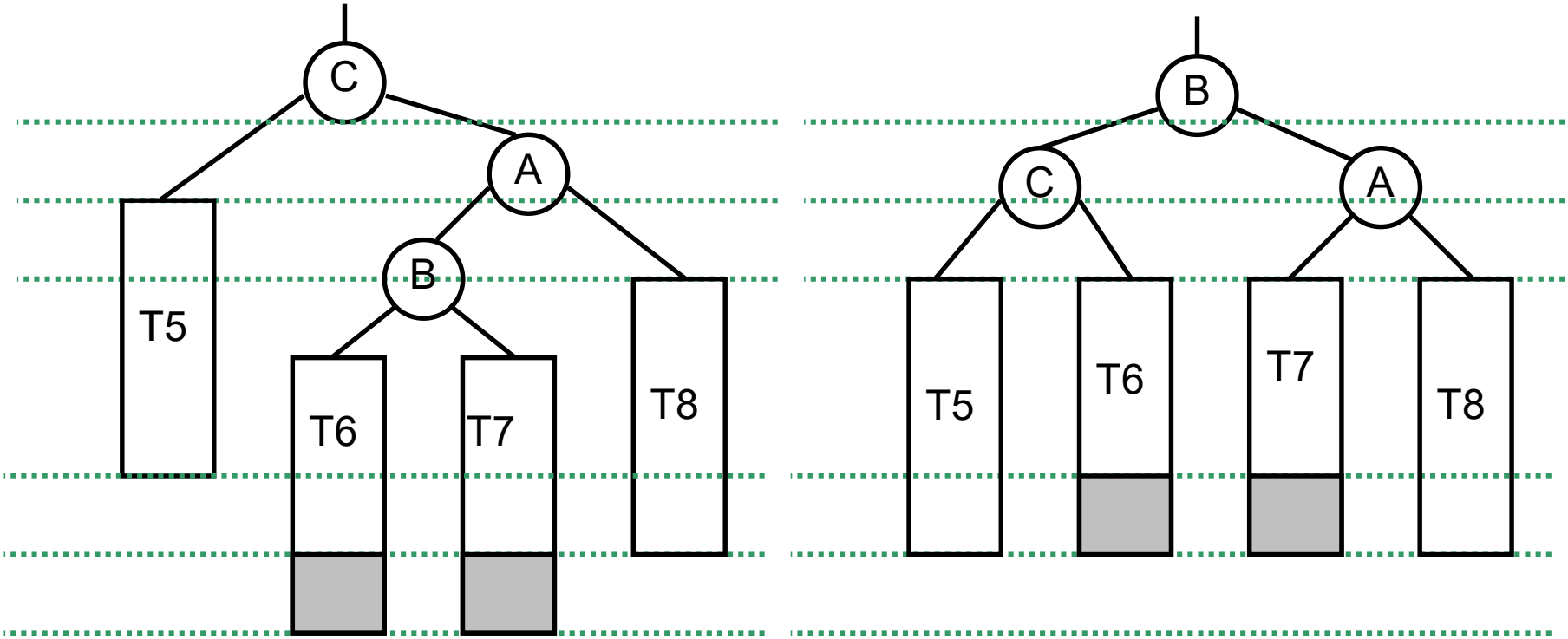
Caso 4: Dech-Izq

SOLUCION: ***"Rotación Doble"***

PASO 1: ***"Rotación a izquierda"***

Equilibrado de árboles AVL

- Inserción:



Caso 4: Dech-Izq

SOLUCION: ***"Rotación Doble"***

PASO 2: ***"Rotación a derecha"***

Equilibrado de árboles AVL

- **Importante:**
  - El nodo insertado (es hoja) se establece como “equilibrado”,
  - Se regresa por el camino de búsqueda:
    - recalculando el factor de equilibrio de los nodos, hasta alcanzar un nodo cuyo árbol no ha crecido, alcanzar la raíz o encontrar un nodo que no cumpla el factor de equilibrio, y requiera una reestructuración para reequilibrar
    - Reequilibrar se resuelve afectando a los dos o tres nodos implicados (según caso): con una reasignación de punteros y de sus factores de equilibrio (por tanto, con coste constante)

  - **Tras reequilibrar:**
    - **el árbol resultante queda de la misma altura que en su estado inicial.** Por lo tanto, si inicialmente estaba equilibrado, el árbol resultante tras la inserción, también estará equilibrado……

➢ **Basta una única rotación, u operación de reequilibrar**, para dejar todo el árbol resultante equilibrado, **no hace falta seguir evaluando hasta la raíz.**

Equilibrado de árboles AVL

Proceso de borrado: como en el Árbol Binario de Búsqueda

1) Localizar el nodo a borrar, y borrarlo:
   - Si el nodo es hoja, se borra → < ANTES: > → <DESPUES: > <Árbol vacío> x ha perdido altura (1 nivel)
   - Si no es hoja:
     - Si tiene un hijo vacío, se sustituye por el hijo no vacío → <DESPUES: > ha perdido altura (1 nivel)
     - Si ambos hijos son no vacíos:
       - se sustituye por el máximo del subárbol izquierdo, y
       - se borra dicho máximo del subárbol izquierdo → <DESPUES: > ¿ha perdido altura? (1 nivel)

2) desandar el camino de búsqueda, verificando el equilibrio de los nodos del camino, y re-equilibrando si es necesario

Se implementa mediante recursión: → Parámetro booleano, que al retornar **indica si el subárbol ha perdido altura**

- si no ha perdido altura, no cambia su factor de equilibrio
- **si ha perdido altura**, hay varias *posibilidades:*

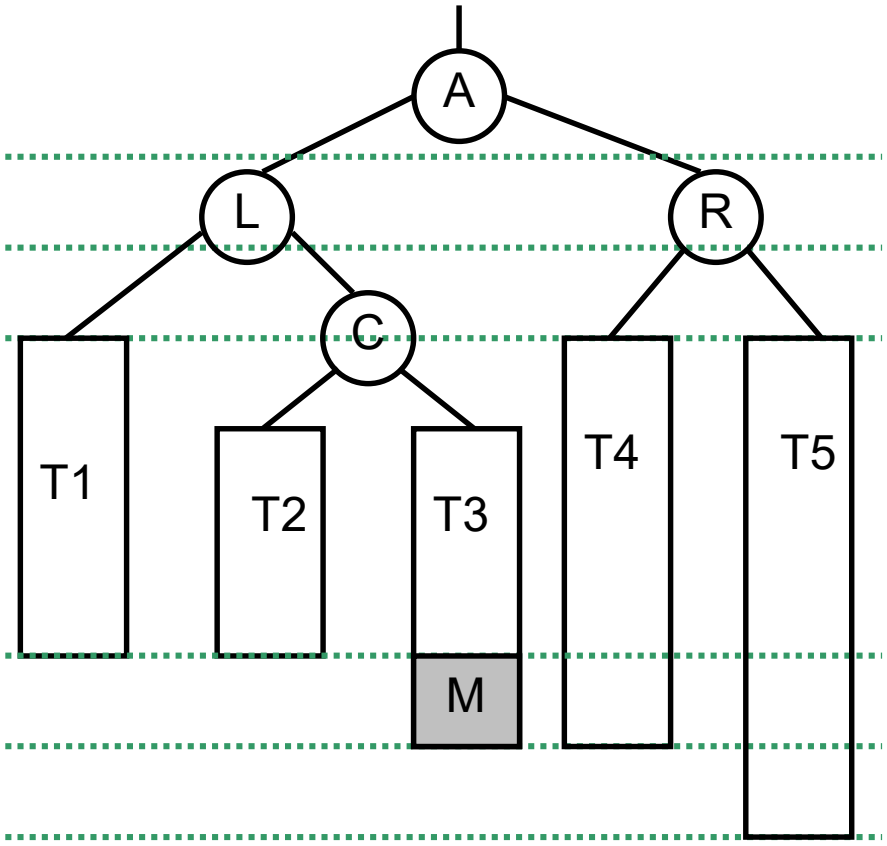
Caso		
Era pesado_dch		
Era equilibrado		
Era pesado_izq		

– Si en alguno de los nodos se pierde la condición de equilibrio (por perder altura), debe ser restaurada → ***reequilibrar***

➢ *debe continuarse hasta la raíz, porque en el borrado pueden ser necesarias varias restauraciones*

Equilibrado de árboles AVL

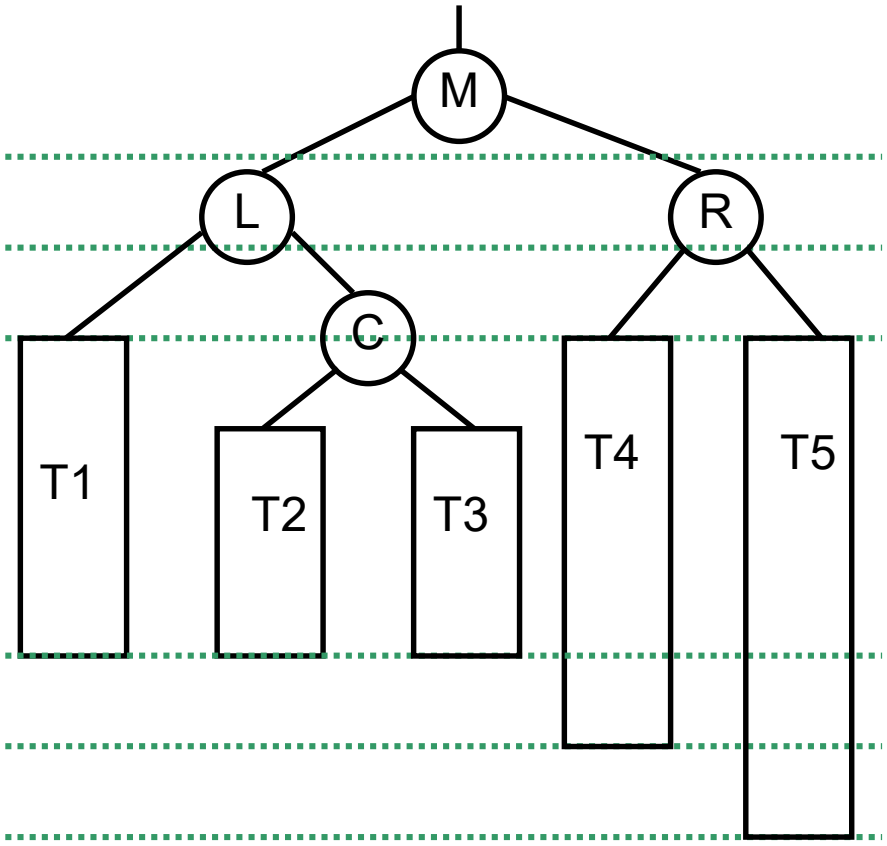
- Ejemplo de borrado:



Borrar A, y

la nueva raíz será M

Equilibrado de árboles AVL



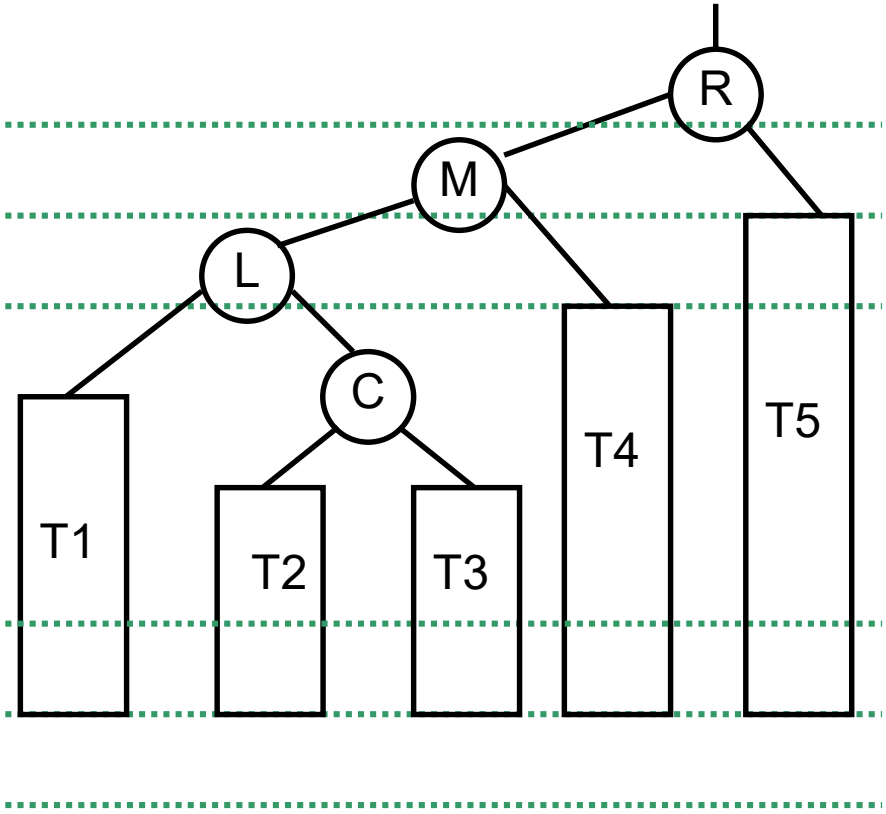
Borrado A,

la nueva raíz es M ,

Árbol resultante demasiado pesado a la derecha

Solución similar a la inserción → aplicar **rotación a la derecha**

Equilibrado de árboles AVL



Borrado A,

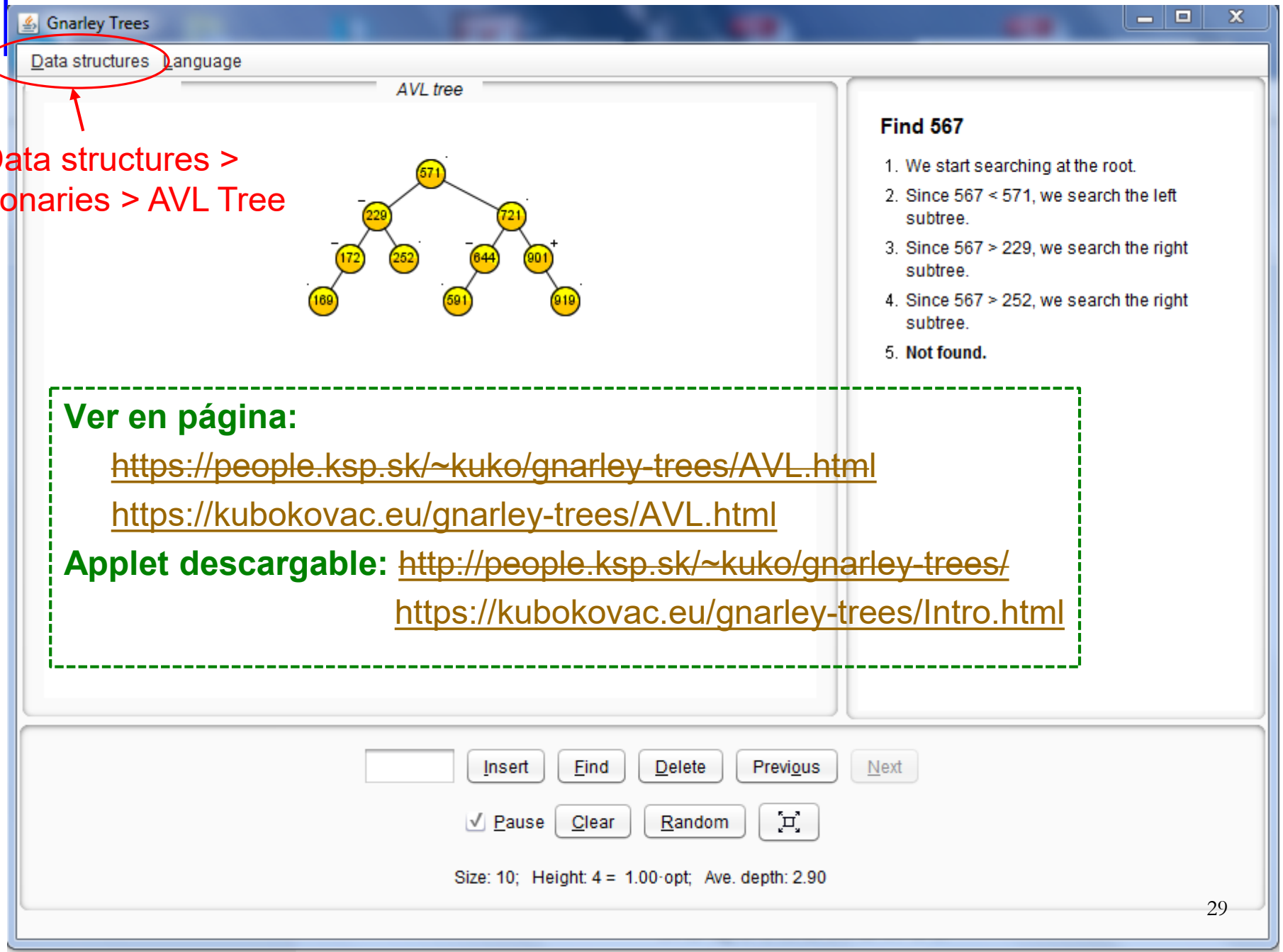
la nueva raíz es M ,

Árbol resultante demasiado pesado a la derecha

aplicada rotación a la **derecha**

El árbol resultante ha perdido altura

En borrado pueden ser necesarias varias operaciones de restauración del equilibrio, y hay que seguir comprobando hasta llegar a la raíz



Data structures > Dictionaries > AVL Tree

Ver en página:

~~https://people.ksp.sk/~kuko/gnarley-trees/AVL.html~~

https://kubokovac.eu/gnarley-trees/AVL.html

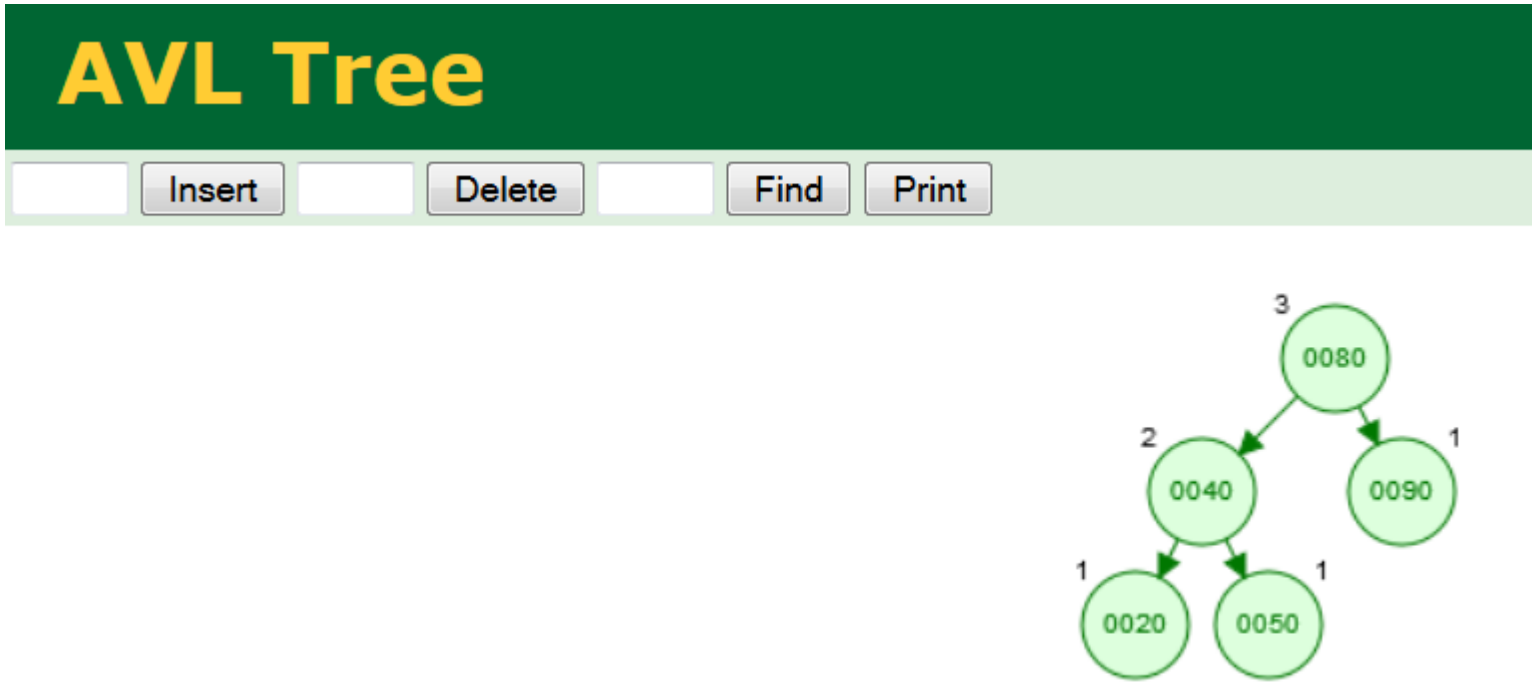
Applet descargable: ~~http://people.ksp.sk/~kuko/gnarley-trees/~~

https://kubokovac.eu/gnarley-trees/Intro.html

Otra opción para visualizar árboles AVL y su funcionamiento:

http://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Utiliza las alturas de los subárboles en lugar del factor de equilibrio



{Ojo: en general las animaciones de esta web consideran las claves como strings, y no evitan las repetidas aunque causen problemas}

Costes de diferentes implementaciones de diccionario

Diccionario (con N claves distintas)	**Vector ordenado** (de tamaño MAX)	**Lista enlazada (ordenada)**	**Árbol Binario de Búsqueda (ABB)**	**AVL**
Coste en memoria	O(MAX)	O(N)	O(N)	**O(N)**
Coste en tiempo (caso peor):				
añadir	O(N)	O(N)	O(N)	**O(log N)**
buscar	O(log N)	O(N)	O(N)	**O(log N)**
borrar	O(N)	O(N)	O(N)	**O(log N)**

Implementación de AVL's

módulo genérico diccionarioAVL ← *{Implementación de diccionario que almacena sus datos en un árbol AVL }*

usa booleanos, pilasGenéricas

parámetros formales

géneros clave, valor

operación *{el género de las claves debe tener definidas funciones de comparación "<" e "=" }*

_<_: clave c1 , clave c2 -> booleano *{devuelve verdad sii e1 es menor que e2}*

_=_: clave c1 , clave c2 -> booleano *{devuelve verdad sii e1 es igual que e2}*

fpf

exporta

tipo diccionario *{Conjunto de pares (clave,valor) en los que no se permiten claves repetidas, y cuenta con operaciones de inserción, búsqueda y borrado relativas a la clave.}*

procedimiento vacío(**sal** d: diccionario);

{Devuelve el diccionario vacío}

función esVacío?(d: diccionario) **devuelve** booleano;

{Devuelve verdad si d es el diccionario vacío, falso en caso contrario}

procedimiento insertar(**e/s** d: diccionario; **ent** c:clave; **ent** v:valor);

{Devuelve el diccionario d, actualizado con el par (c,v) añadido al diccionario si en d no había un par con clave c; si en d ya había un par con la clave c, entonces actualiza el valor que acompaña a dicha clave con v}

procedimiento borrar(**e/s** d: diccionario; **ent** c:clave);

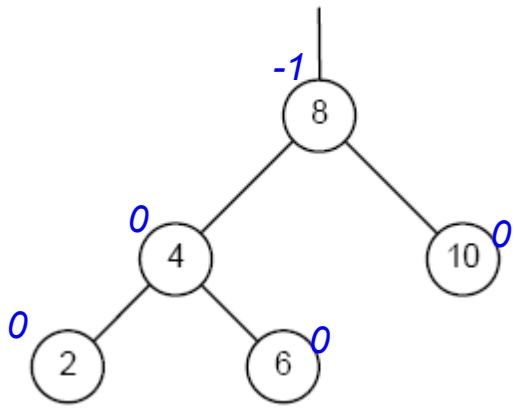
{Dada una clave c, devuelve el diccionario d actualizado tras eliminar el par que tenía dicha clave}

procedimiento buscar(**ent** d: diccionario; **ent** c:clave; **sal** éxito:booleano; **sal** v:valor);

{Dado un diccionario d y una clave c, devuelve verdad en éxito si la clave está en el diccionario, y en ese caso devuelve en v el valor que acompaña a la clave en el diccionario. Si la clave no está en el diccionario, en éxito devuelve falso}

...

Implementación de AVL's



...

implementación

{Representación interna:}

tipos

avl = ↑nodo;

modulo pila_de_avl **concreta** pilasGenéricas(avl);

factorEquil = (pesadolzq, equilibrado, pesadoDer);

nodo = **registro**

laClave:clave;
elValor:valor;

→ *podría ser* ***dato: elemento;***

equilibrio:factorEquil;

izq,der:avl

freg

diccionario=**registro**

cardinal: entero;

raíz: avl; ***{puntero a la raíz de un árbol AVL}***

iter: pila_de_avl.pila***{Para el iterador: pila de punteros a nodos del árbol AVL}***

freg

Coste en memoria O(N)

...

Implementación de AVL's

procedimiento vacío(**sal** d:diccionario)
{... igual que con árboles binarios de búsqueda...}

Coste O(1)

función esVacío(d:diccionario) **devuelve** booleano
{... igual que con árboles binarios de búsqueda...}

Coste O(1)

procedimiento buscar(**ent** d:diccionario; **ent** c:clave;
sal éxito:boolean;
sal v:valor)
{... igual que con árboles binarios de búsqueda (buscando y comparando por clave)...}

En el caso peor: "lineal en la altura del árbol"

…

Por ser un AVL:

Coste O(log N)

Implementación de AVL's

```
{la operación pública:}
procedimiento insertar(e/s d:diccionario; ent c:clave; ent v:valor)
variable alturaModificada,añadido:booleano
principio
  alturaModificada:=falso;
  insertarRec(d.raíz,c,v, alturaModificada, añadido);
  si añadido entonces d.cardinal:=d.cardinal+1 fsi
fin

{operaciones privadas:}
procedimiento insertarRec(e/s a:avl; ent c:clave; ent v:valor;
                          e/s alturaModificada:booleano; sal nuevo:booleano)
principio
  si a=nil entonces
    nuevodato(a);
    a↑.laClave:=c;    a↑.elValor:=v;
    a↑.izq:=nil;      a↑.der:=nil;
    a↑.equilibrio:=equilibrado;
    alturaModificada:=verdad; {pasa de ser árbol vacío a ser hoja, por tanto a crece}
    nuevo:=verdad
 {sino_si ...}
```

Implementación de AVL's

INSERCION		
Era pesado_dch		
Era equilibrado		
Era pesado_izq		

Actualizar Fe y reequilibrar si es necesario (tabla de casos)

```
sino_si c<a↑.laClave entonces {buscamos por el hijo izq}
   insertarRec(a↑.izq,c,v,alturaModificada,nuevo);
   si alturaModificada entonces
     selección
       a↑.equilibrio=pesadoIzq:
         si a↑.izq↑.equilibrio=pesadoIzq entonces
                  rotaciónIzq(a)
         sino rotaciónIzqDer(a)
         fsi;
         alturaModificada:=falso; {después de reequilibrar, la altura de a no ha crecido}
       a↑.equilibrio=equilibrado:
                  a↑.equilibrio:=pesadoIzq; { a, ha crecido por su hijo izq}
       a↑.equilibrio=pesadoDer:
         a↑.equilibrio:=equilibrado; alturaModificada:=falso { a, no ha crecido}
     fselección
   fsi
  sino_si a↑.laClave<c entonces {buscamos por el hijo der}
     ...{simétrico al anterior pero equilibrado el hijo derecho con rotaciónDer y
     rotaciónDerIzq}...
  sino a↑.elValor:=v; nuevo:=falso {actualizar el valor asociado a la clave, no se añade un
     nuevo par, y el árbol no modifica su forma}
  fsi
fin
```

En el caso peor: "lineal en la altura del árbol"…

Por ser un AVL:

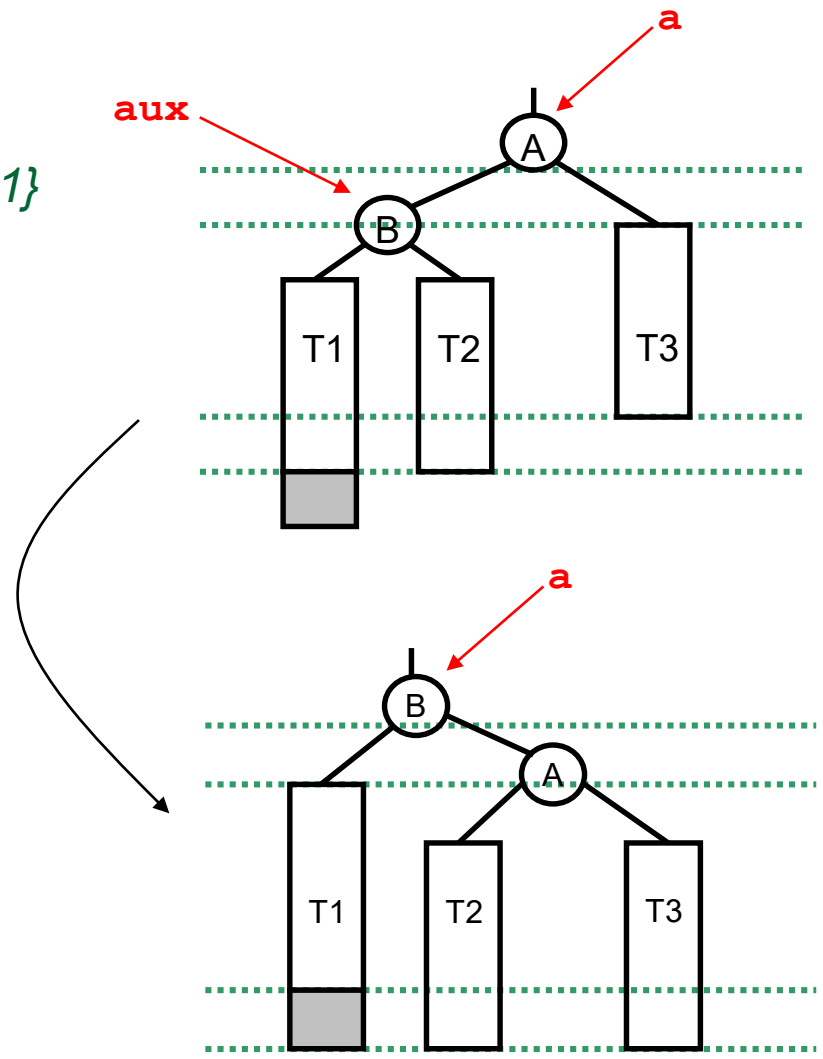
Coste O(log N)

Implementación de AVL's

{ Algoritmos de rotación: }

procedimiento **rotaciónIzq**(**e/s** a:avl) *{caso 1}*
  variable aux:avl;
principio
    aux:=a↑.izq;
    a↑.izq:=aux↑.der;
    a↑.equilibrio:=equilibrado;
    aux↑.der:=a;
    a:=aux;
    a↑.equilibrio:=equilibrado
fin

Coste O(1)

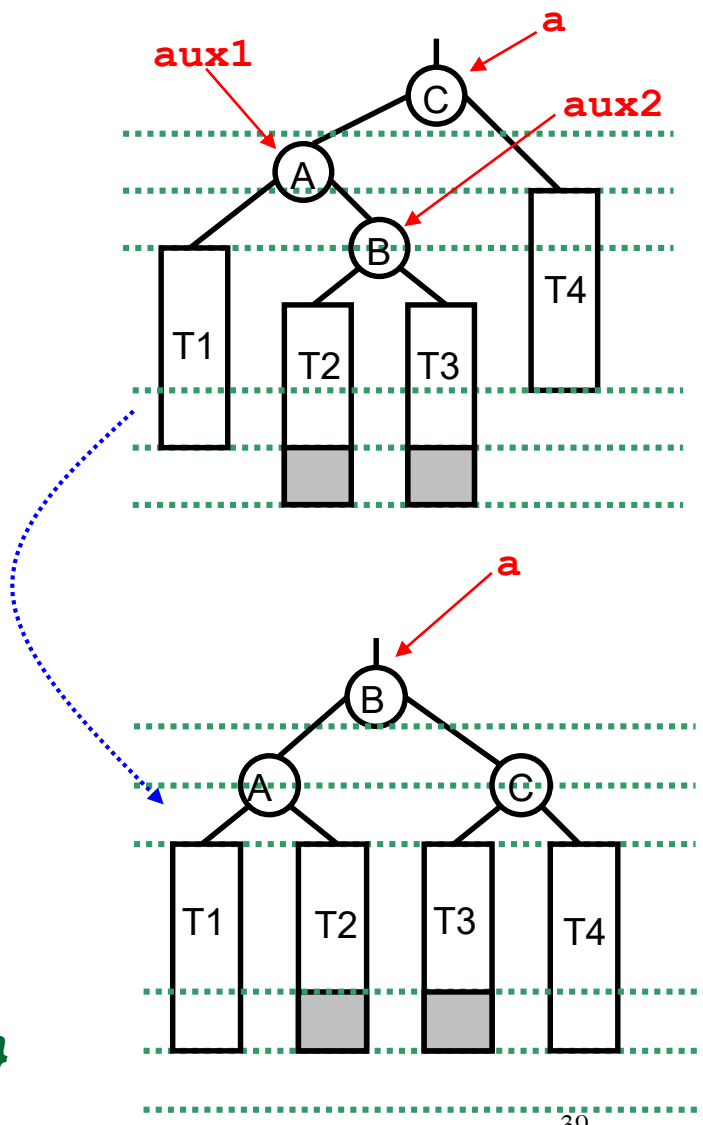


procedimiento ***rotaciónDer****(**e/s** a:avl)*
{ … caso 2: simétrico al caso anterior…}

Coste O(1)

```
procedimiento rotaciónIzqDer(e/s a:avl) {caso 3}
variables aux1,aux2:avl
principio
    aux1:=a↑.izq; aux2:=a↑.izq.↑der;
    aux1↑.der:=aux2↑.izq;  aux2↑.izq:=aux1;
    a↑.izq:=aux2;
    si  aux2↑.equilibrio=pesadoIzq entonces
            aux1↑.equilibrio:=equilibrado;
            a↑.equilibrio:=pesadoDer
    sino_ si aux2↑.equilibrio=equilibrado entonces
            aux1↑.equilibrio:=equilibrado;
            a↑.equilibrio:=equilibrado
    sino
            aux1↑.equilibrio:=pesadoIzq;
            a↑.equilibrio:=equilibrado
    fsi;
    a↑.izq:=aux2↑.der; aux2↑.der:=a;
    aux2↑.equilibrio:=equilibrado;
    a:=aux2
fin
```

{Caso 3 es equivalente a:
1º) rotacionDer de a↑.izq,
2º) rotaciónIzq de a, pero
arreglando los factores de equilibrio}



Coste O(1)

```
procedimiento rotaciónDerIzq (e/s a:avl) {caso 4}
  { … caso 4: simétrico al caso anterior…}
```

Implementación de AVL's

```
procedimiento borrar(e/s d:diccionario; ent c:clave)        {la operación pública}
  variable alturaModificada,borrado:booleano
principio
  alturaModificada:=falso;
  borrarRec(d.raíz,clave, alturaModificada, borrado);
  si borrado entonces d.cardinal:=d.cardinal-1 fsi
fin

{operaciones privadas:    }
procedimiento borrarRec(e/s a:avl; ent c:clave; e/s alturaModificada:booleano;
     sal eliminado: booleano)
variable aux:avl
principio
  eliminado:=falso;
  si a≠nil entonces
    si c<a↑.laClave entonces
          borrarRec(a↑.izq,c,alturaModificada,eliminado);
          si alturaModificada entonces {si a↑.izq ha perdido altura}
            equillzq(a,alturaModificada) {hacer los arreglos necesarios según estudio
                                          (tabla) de casos de borrados}
          fsi
   { sino_si ...}
```

BORRADOS		
Era pesado_dch		
Era equilibrado		
Era pesado_izq		

Implementación de AVL's

BORRADOS		
Era pesado_dch		
Era equilibrado		
Era pesado_izq		

```
sino_si a↑.laClave<c entonces
        borrarRec(a↑.der,c, alturaModificada,eliminado);
        si alturaModificada entonces {si a↑.der ha perdido altura}
          equilDer(a,alturaModificada);
        fsi
sino {se ha encontrado la clave a borrar}
       eliminado:= verdad;
        si a↑.izq=nil entonces
             aux:=a; a:=a↑.der; disponer(aux);
            alturaModificada:=verdad {a, ha perdido altura}
        sino_si a↑.der=nil entonces
             aux:=a; a:=a↑.izq; disponer(aux);
             alturaModificada:=verdad {a, ha perdido altura}
        sino
           borrarMaxClave(a↑.izq,a↑.laClave,a↑.elValor,alturaModificada);
           si alturaModificada entonces {si a↑.izq ha perdido altura}
              equillzq(a,alturaModificada);
           fsi
        fsi
   fsi
  fsi {de si a≠nil }
fin
```

En el caso peor: "lineal en la altura del árbol"

…

Por ser un AVL:

Coste O(log N)

Implementación de AVL's

```
procedimiento equillzq(e/s a:avl; e/s alturaModificada:booleano)
procedimiento equilDer(e/s a:avl; e/s alturaModificada:booleano)
{ ... estudio de casos (similar inserción).
Y se usan los mismos algoritmos de rotación}
```

Coste O(1)

Coste O(1)

```
procedimiento borrarMaxClave(e/s a:avl; sal c:clave; sal v:valor;
                             e/s alturaModificada:boolean) {Precondición: a≠nil}
  variable aux:avl;
principio
    si a↑.der=nil entonces
        c:=a↑.laClave;  v:=a↑.elValor;
        aux:=a;
        a:=a↑.izq;
        disponer(aux);
        alturaModificada:=verdad { a, ha perdido altura}
    sino
        borrarMaxClave(a↑.der,c,v,alturaModificada);
        si alturaModificada entonces {si a↑.der ha perdido altura}
            equilDer(a, alturaModificada);
        fsi
    fsi
fin
```

Coste caso peor O(log N)

BORRADOS		
Era pesado_dch		
Era equilibrado		
Era pesado_izq		

¿Se puede utilizar un AVL para almacenar los datos de un TAD contenedor multiconjunto?

- Si, siempre y cuando no metamos más de un nodo en el AVL con la misma *clave* (=dato que permita distinguir un elemento de otro)
  - Ejemplo: Ejercicio de tablas de frecuencias (Lecciones 2 y 3)
    - Es un contenedor multiconjunto, pero por sus características le basta almacenar su información de forma "resumida"…
    - cada nodo del AVL contendría: un *dato* y su *frecuencia* (frecuencia= contador de cuántas veces se ha "añadido" dicho dato al multiconjunto) →
      - Por tanto, no habría más de un nodo en el AVL con el mismo *dato*

- PERO eso no es factible para cualquier multiconjunto….

¿Se puede utilizar un AVL para almacenar los datos de un TAD contenedor multiconjunto?

- Si, siempre y cuando no metamos más de un nodo en el AVL con la misma *clave* (=dato que permita distinguir un elemento de otro)
  - PERO, ¿Y si hay que almacenar cada dato, aunque tenga igual clave que otro ya existente en el multiconjunto?
    - **En cada nodo del AVL se almacenan** todos aquellos datos con misma *clave*: introduciéndolos **en una estructura de datos auxiliar** almacenada o accesible desde dicho nodo del AVL
      - La elección de cuál sea **dicha estructura auxiliar**, y el coste de utilizarla también al implementar las operaciones del TAD, **afectará también al coste de <u>las operaciones del multiconjunto</u>**: **podrían <u>no</u> tener coste <u>O(logN)</u>**

Otros árboles binarios de búsqueda balanceados

- Árboles *roji-negros* (*Red-Black* Trees)
  - https://en.wikipedia.org/wiki/Red-black_tree
  - http://webdiis.unizar.es/asignaturas/EDA/restringido/varios/rojinegros.pdf (usuario: eda contraseña: tad)
  - Applets o animaciones:
    - https://kubokovac.eu/gnarley-trees/Redblack.html
    - https://people.ksp.sk/~kuko/gnarley-trees/Redblack.html
- …

➢ Rojinegros y otros, por ejemplo en el *capítulo 12 del libro "Weiss, M.A.: Data Structures and Algorithm Analysis in C++, Fourth Edition, Pearson/Addison Wesley, 2014."*

➢ *Capítulo 8 del libro "Z. J. Hernández, J. C. Rodríguez, etc .: Fundamentos de Estructuras de Datos. Soluciones en Ada, Java y C++. Thomson Paraninfo, 2005"*