

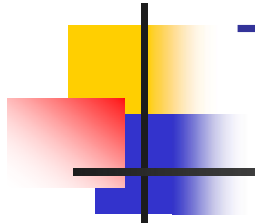


# Estructuras de Datos

---

## **AVL**

Teórico Programación 3  
Curso 2009



# Temario

---

- Motivación
- Árboles equilibrados
- Árboles perfectamente equilibrados
- Árboles AVL

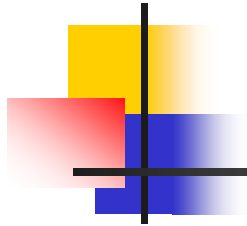


# Motivación (1/4)

---

- ABB con  $n$  de elementos, tiene complejidad  $O(\log n)$  en caso promedio para las operaciones de:
  - ❑ Inserción
  - ❑ Borrado
  - ❑ Búsqueda
  - ❑ Pertenece

pero... ¿qué pasa en el peor caso?



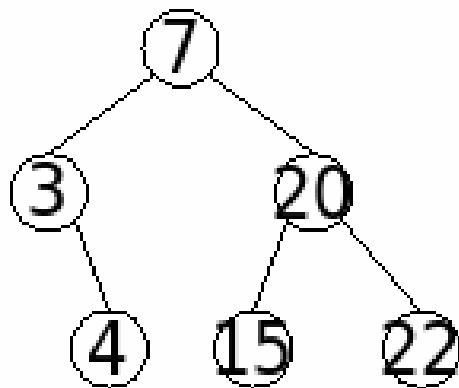
## Motivación (2/4)

---

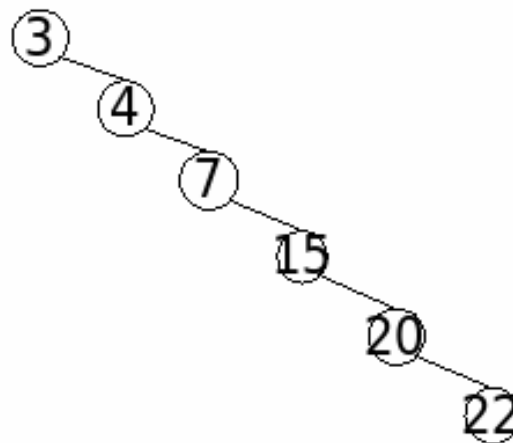
- La complejidad de las operaciones anteriores depende la altura del ABB, y la altura depende de la secuencia (orden) de ingreso a la estructura de los  $n$  elementos

# Motivación (3/4)

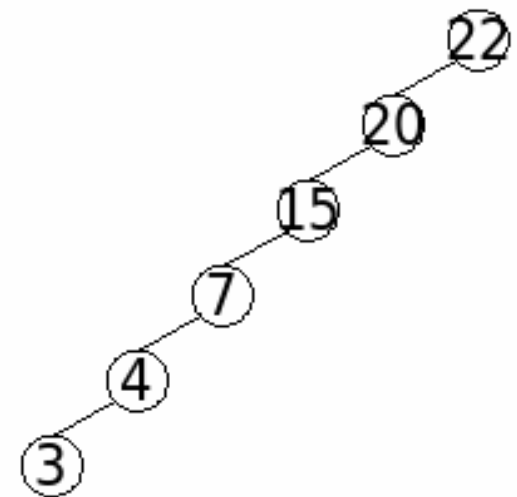
**Secuencia: 7, 3, 4, 20, 15, 22**



**Secuencia: 3, 4, 7, 15, 20, 22**

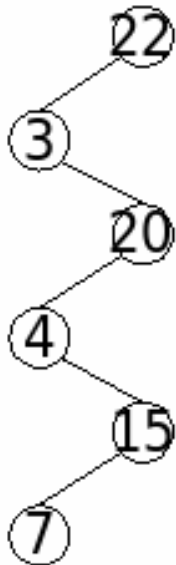


**Secuencia: 22, 20, 15, 7, 4, 3**

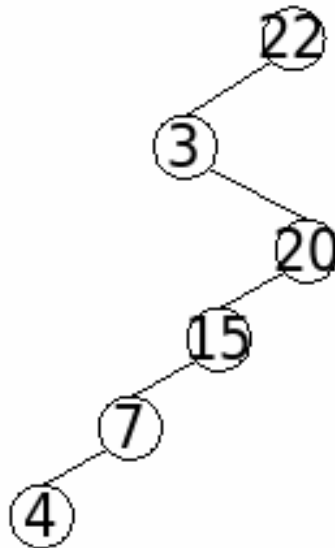


# Motivación (4/4)

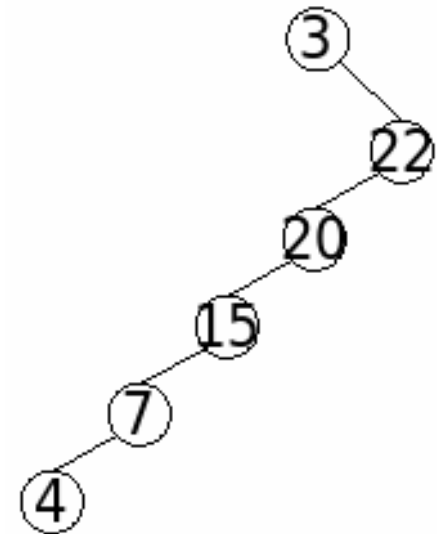
**Secuencia: 22, 3, 20, 4, 15, 7**



**Secuencia: 22, 3, 20, 15, 7, 4**



**Secuencia: 3, 22, 20, 15, 7, 4**





# Árboles equilibrados

---

- Árboles cuya altura máxima es  $O(\log_n)$  independientemente de la secuencia de ingreso de los elementos al árbol
- Para minimizar el problema de los ABB desequilibrados, sea cual sea el grado de desequilibrio que tengan, se puede recurrir a **algoritmos globales** de equilibrado de árboles
  - ❑ Ejemplo: crear una lista mediante la lectura en *inorden* del árbol, y volver a reconstruirlo equilibrado
- Desventaja de estos algoritmos:
  - ❑ *requieren explorar y reconstruir todo* el árbol cada vez que se inserta o se elimina un elemento, de modo que lo que se gana al acortar las búsquedas, (ya que se efectúan menos comparaciones), se pierde equilibrando el árbol



# Árboles perfectamente equilibrados

---

- Basan el equilibrio en la cantidad de nodos
- Se tiene que cumplir en todo momento que:

$$\left| cant\_nodos(T_{izq}) - cant\_nodos(T_{der}) \right| \leq 1$$

- Ventaja
  - la altura siempre es  $\log n$
- Desventaja
  - algoritmos sumamente complejos ya que frecuentemente deben reorganizar el árbol





# AVL - Adelson-Velskii y Landis

---

- Definiciones
- Ejemplo Inserción
- Tipos de rebalanceo o rotaciones
- Algoritmo de Inserción



# AVL

## Definiciones

---

- AVL: árbol binario de búsqueda en el que para cada nodo, las alturas de sus subárboles izquierdo y derecho no difieren en más de 1
- Formalmente
  - Un árbol vacío es un árbol AVL
  - Si  $T$  es un árbol no vacío y  $T_L$  y  $T_R$  son sus subárboles,  $T$  es un AVL  $\Leftrightarrow$

$T_L$  es un AVL

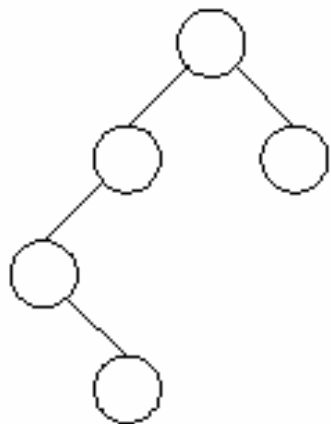
$T_R$  es un AVL

$$|altura(T_L) - altura(T_R)| \leq 1$$

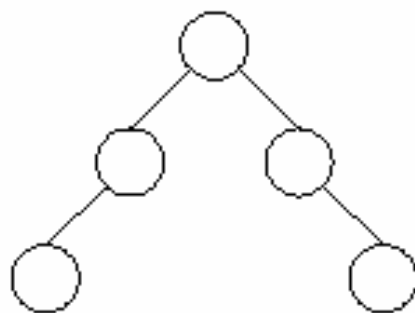
# AVL

## Definiciones

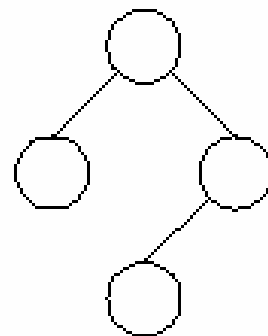
### ■ Ejemplos



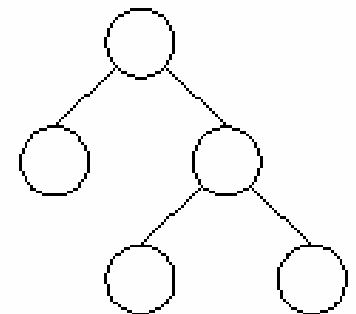
↓  
No AVL



↓  
AVL



↓  
AVL



↓  
AVL que no es  
perfectamente  
equilibrado



# AVL

## Definiciones

---

- **Factor de balanceo** (FB) es la diferencia de altura de los subárboles:

$$FB = altura(T_L) - altura(T_R)$$

- Para todo nodo de un AVL se cumple que  $FB = 1, 0, -1$ 
  - $FB = 1$  si  $altura(T_L) > altura(T_R)$
  - $FB = -1$  si  $altura(T_L) < altura(T_R)$
  - $FB = 0$  si  $altura(T_L) = altura(T_R)$



- Teorema:

Sea  $h$  la altura de un árbol AVL con  $n$  nodos.

Se cumple que:

$$\log(n+1) \leq h \leq 1.44\log(n+2) - 0.328$$

- Es decir que:

$h$  es  $O(\log n)$



- El algoritmo para mantener un árbol AVL equilibrado se basa en reequilibrados locales, de modo que no es necesario explorar todo el árbol después de cada inserción o borrado



# AVL

## Ejemplo Inserción

---

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4

Inicialmente árbol vacío.



8<sup>0</sup>

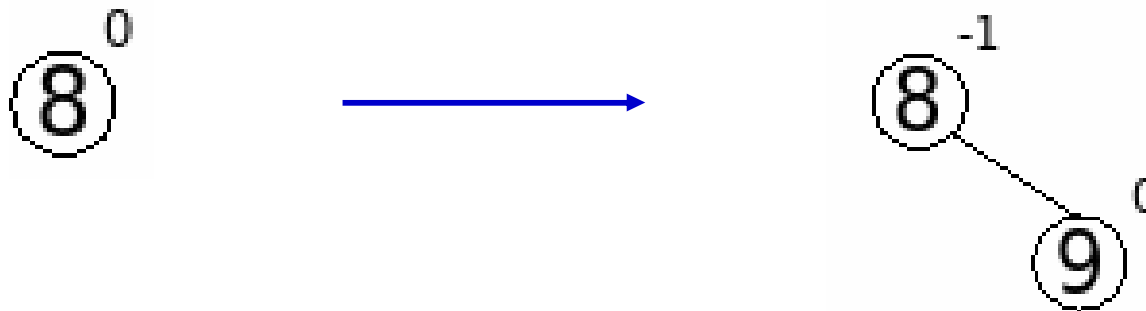


# AVL

## Ejemplo Inserción

---

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4

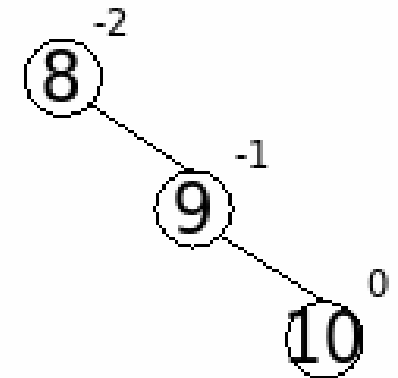
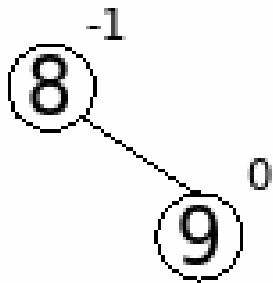




# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



Al insertar el elemento 10, se viola la propiedad de AVL

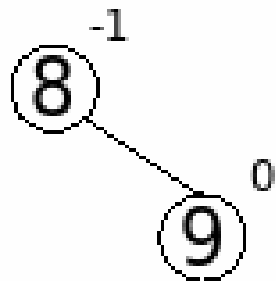
Es necesario rebalancear el árbol

Observar que la altura previa a la inserción era  $h=2$  y luego de la inserción es  $h=3$ .

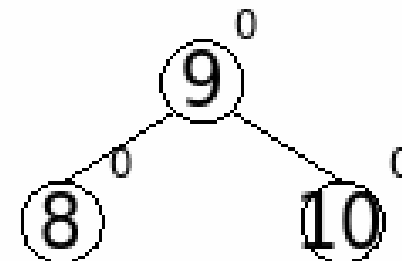
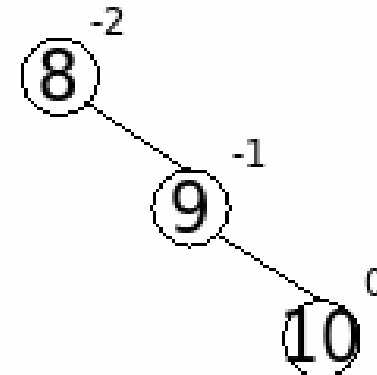
# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



Rebalanceo

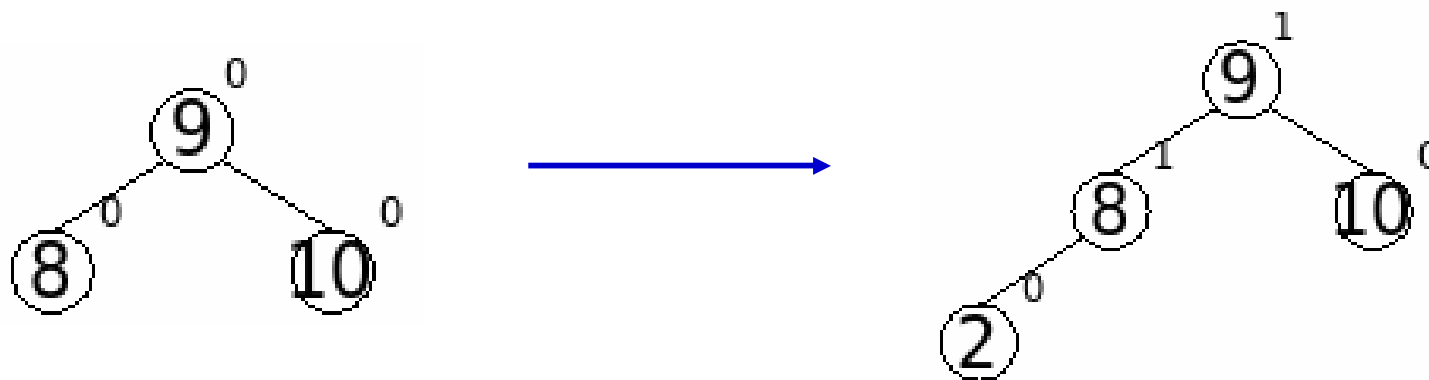


El rebalanceo vuelve la altura a  $h=2$ , que era el valor previo a la inserción.

# AVL

## Ejemplo Inserción

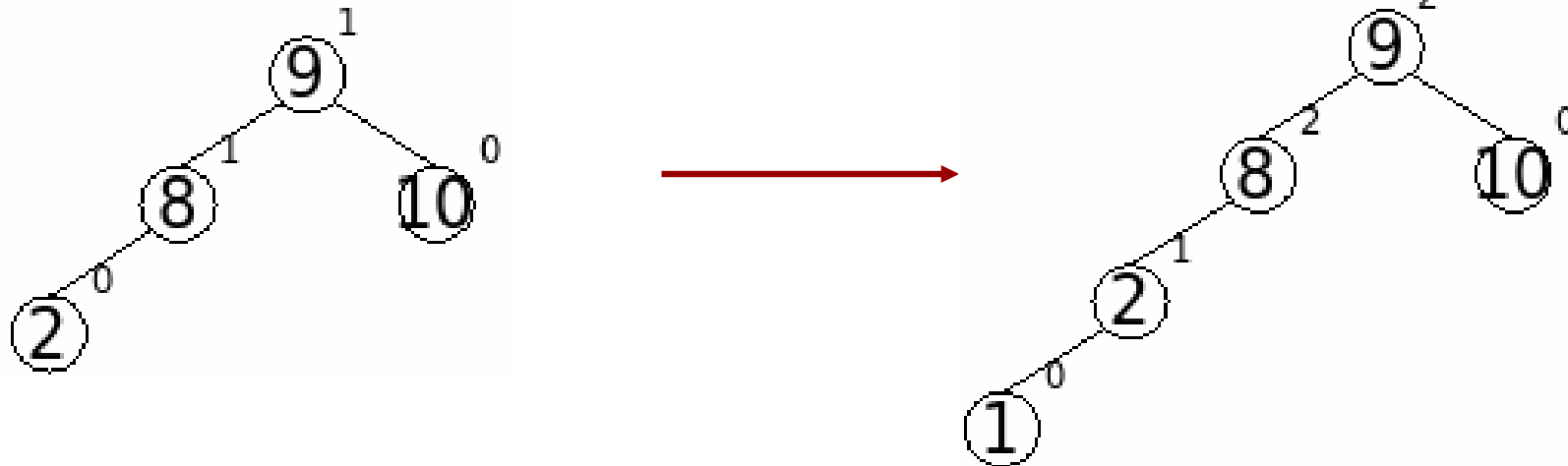
Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



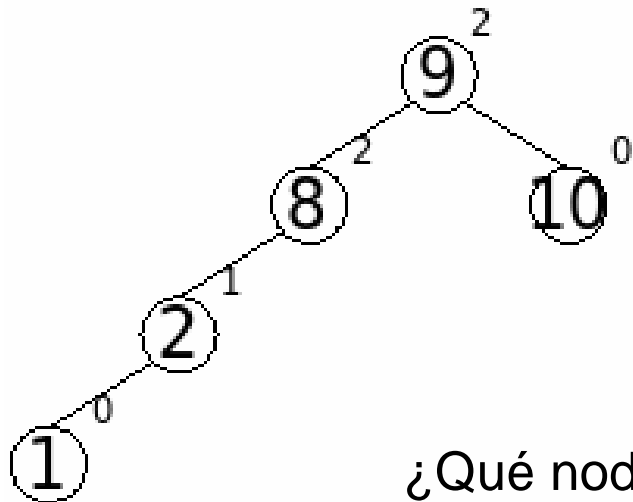
Al insertar el elemento 1, se viola la propiedad de AVL

Es necesario rebalancear el árbol

# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4

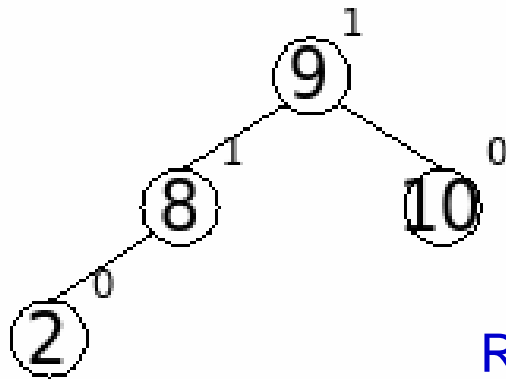


¿Qué nodo se debe rebalancear, el 8 ó el 9?

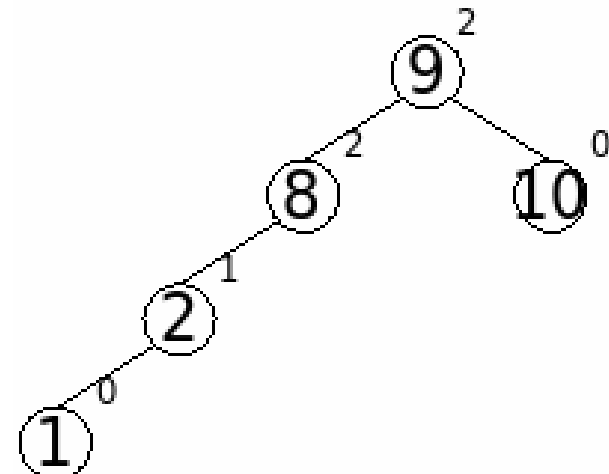
# AVL

## Ejemplo Inserción

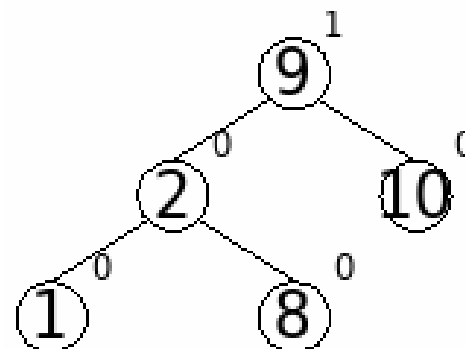
Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



Rebalanceo



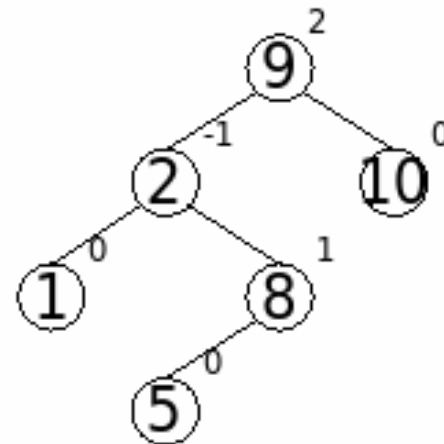
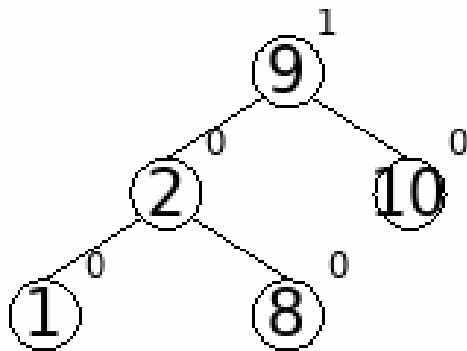
Observación: si se rebalancea el subárbol de raíz 8 queda, rebalanceado también el subárbol de raíz 9 (porque el subárbol izquierdo de raíz 8 vuelve a tener altura  $h=2$  y el derecho tiene  $h=1$ )



# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



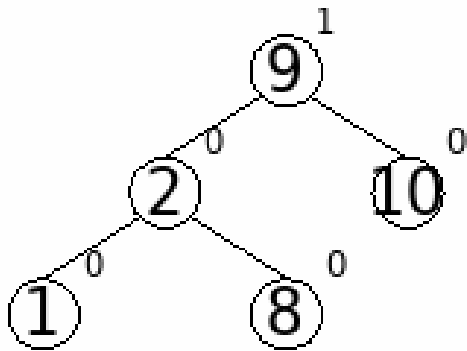
Al insertar el elemento 5, se viola la propiedad de AVL

Es necesario rebalancear el árbol

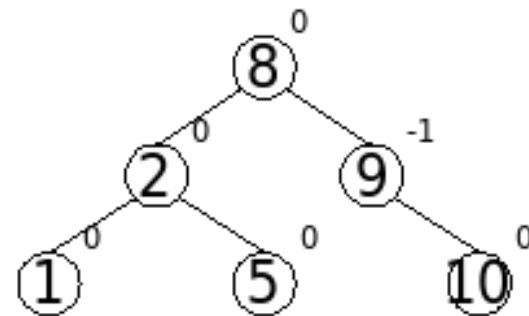
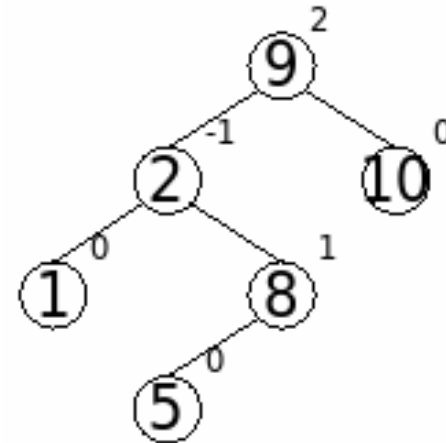
# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



Rebalanceo

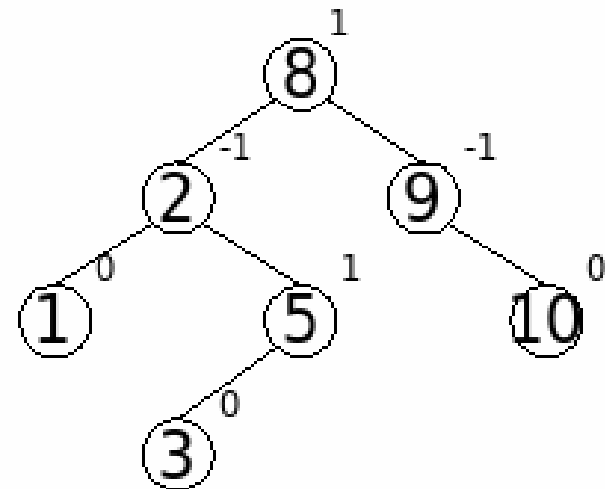
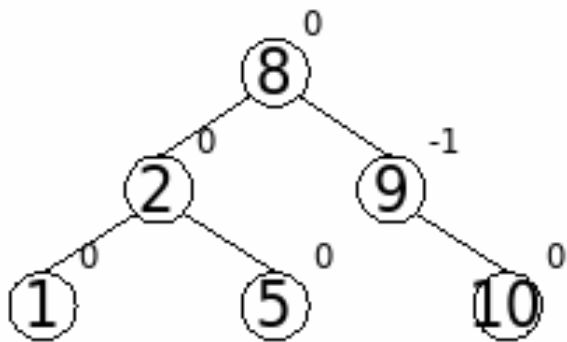




# AVL

## Ejemplo Inserción

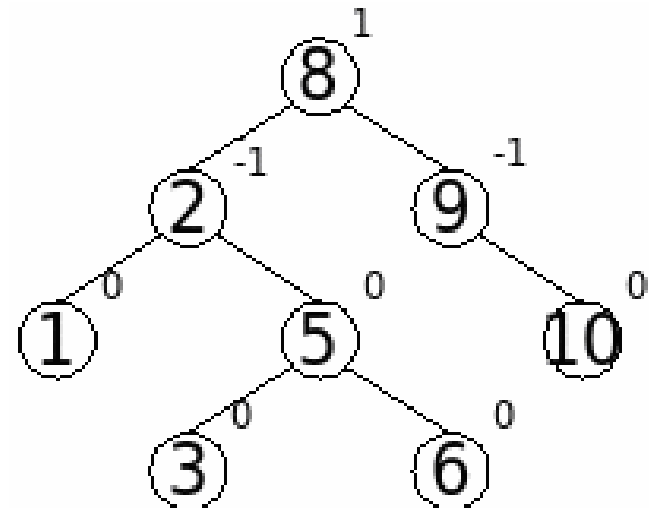
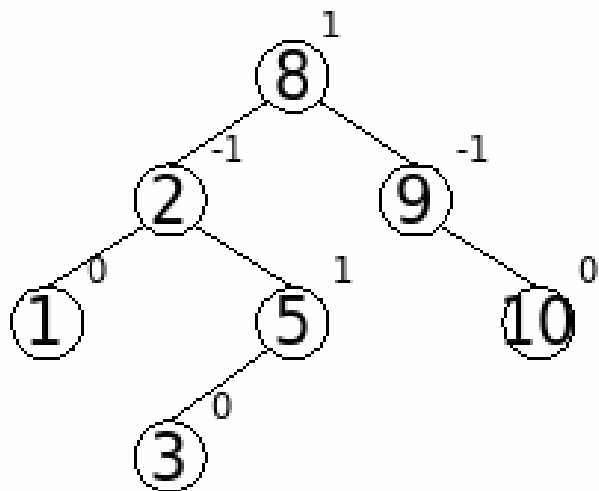
Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



# AVL

## Ejemplo Inserción

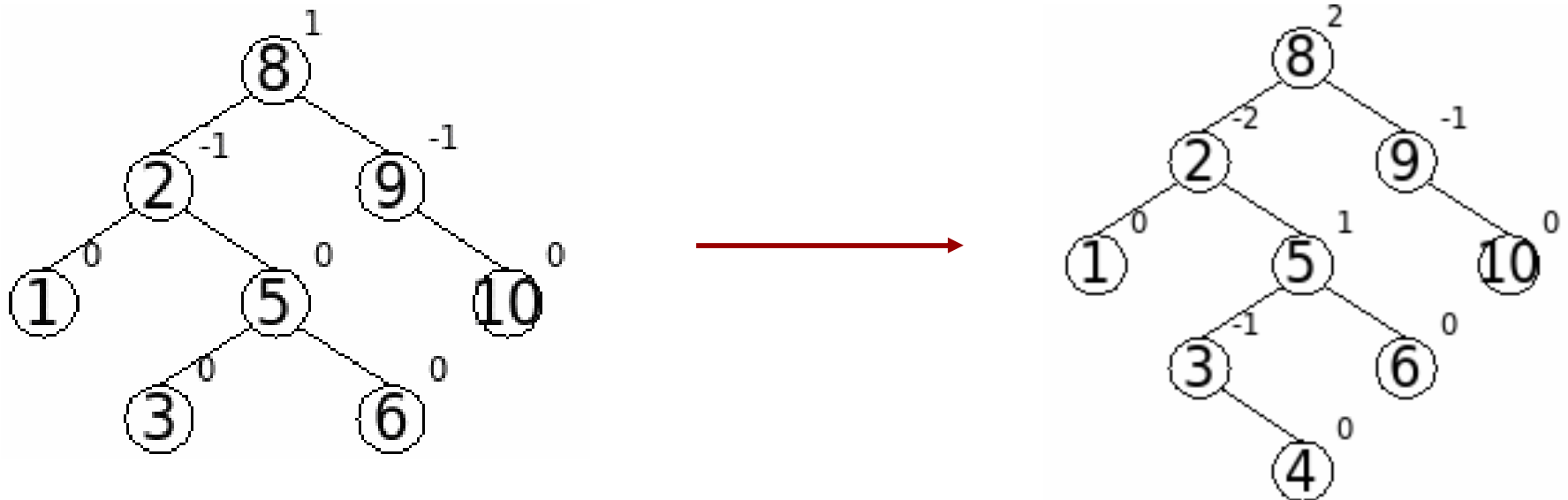
Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, 4



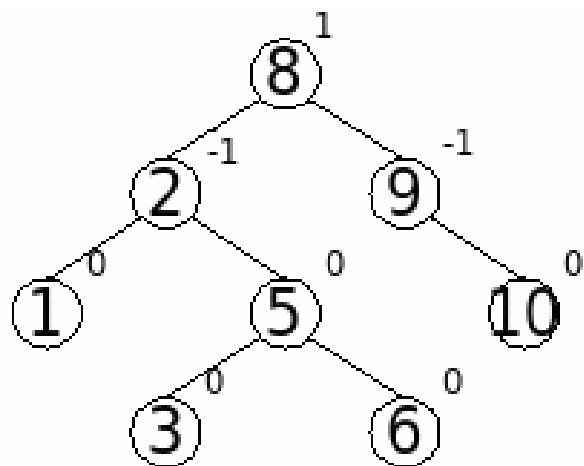
Al insertar el elemento 4, se viola la propiedad de AVL

Es necesario rebalancear el árbol

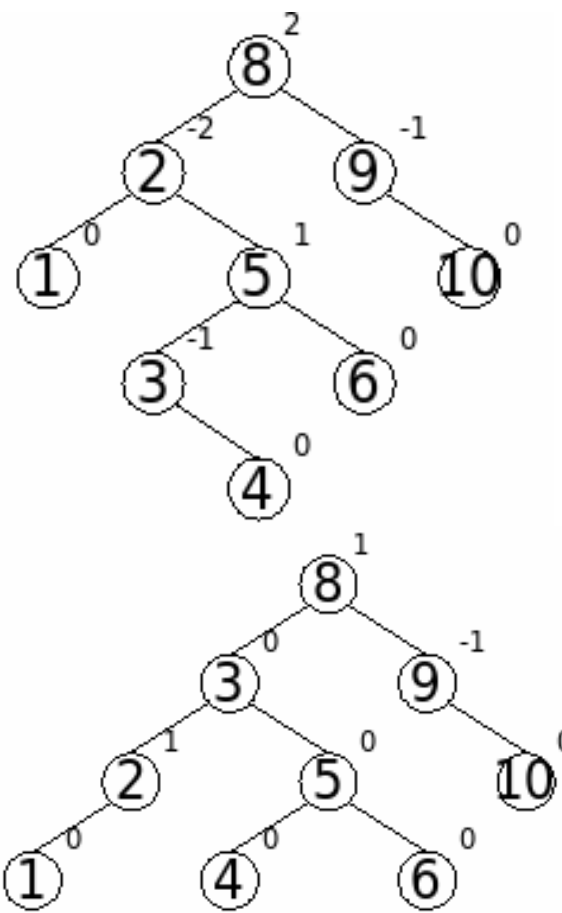
# AVL

## Ejemplo Inserción

Elementos a insertar: 8, 9, 10, 2, 1, 5, 3, 6, .



Rebalanceo





# AVL

## Ejemplo Inserción: Observaciones (1/3)

---

- Observaciones

- ❑ El rebalanceo se realiza en el subárbol cuya raíz sea el ancestro más cercano al nodo insertado y cuyo factor de balance se transforme, a consecuencia de la inserción, en 2 ó -2.
- ❑ Luego de rebalanceado el subárbol anterior, no es necesario rebalancear nada más, ya que si la altura de este subárbol antes de la inserción era  $h$  y después de la inserción la altura se incrementa a  $h+1$ , pero luego del rebalanceo vuelve a ser  $h$ .



# AVL

## Ejemplo Inserción: Observaciones (2/3)

- Se quiere insertar un nodo en un árbol  $T$  con subárbol izquierdo  $T_L$  y subárbol derecho  $T_R$ 
  - Caso 1: el nodo a insertar se inserta en  $T_L$  haciendo que su altura se incremente (si no se incrementa, no se produce desbalanceo). Pueden suceder los siguientes casos:

Previo a la inserción	Luego de la inserción
$\text{altura}(T_L) = \text{altura}(T_R), \text{FB} = 0$	$\text{FB} = 1$ puede vulnerarse el equilibrio en un nodo ancestro.
$\text{altura}(T_L) < \text{altura}(T_R), \text{FB} = -1$	$\text{FB} = 0$ no se debe rebalancear.
$\text{altura}(T_L) > \text{altura}(T_R), \text{FB} = 1$	Se debe rebalancear



# AVL

## Ejemplo Inserción: Observaciones (3/3)

- ❑ Caso 2: el nodo a insertar se inserta en  $T_R$  haciendo que su altura se incremente (si no se incrementa, no se produce desbalanceo). Pueden suceder los siguientes casos:

Previo a la inserción	Luego de la inserción
$\text{altura}(T_R) = \text{altura}(T_L), \text{FB} = 0$	$\text{FB} = -1$ puede vulnerarse el equilibrio en un nodo ancestro.
$\text{altura}(T_R) < \text{altura}(T_L), \text{FB} = 1$	$\text{FB} = 0$ no se debe rebalancear.
$\text{altura}(T_R) > \text{altura}(T_L), \text{FB} = -1$	Se debe rebalancear



# AVL

## Tipos de rotaciones

---

- Tipos de rebalanceos o rotaciones
  - ▣ Simples
    - Tipo LL
    - Tipo RR
  - ▣ Dobles
    - Tipo LR
    - Tipo RL





# AVL

## Tipos de rotaciones: Simples

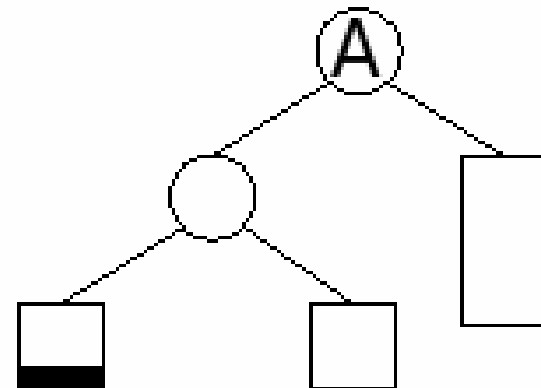
---

- Rotaciones simples involucran 3 subárboles.
- Se clasifican en
  - ▣ Tipo LL
  - ▣ Tipo RR

# AVL

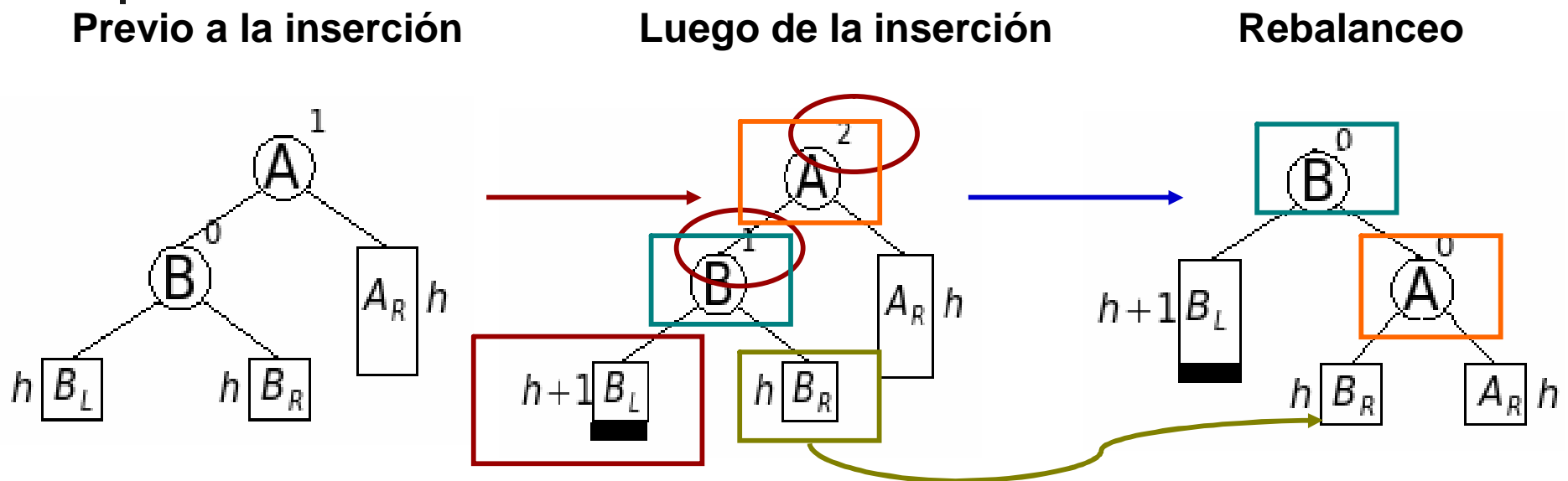
## Tipos de rotaciones: Simples – Tipo LL

- Sea A el ancestro más cercano del nodo insertado, cuyo factor de balanceo luego de la inserción es 2 ó -2.
- El nodo fue insertado en el subárbol izquierdo del subárbol izquierdo de A



# AVL

## Tipos de rotaciones: Simples – Tipo LL



**B se convierte en la raíz y A en hijo derecho de B**

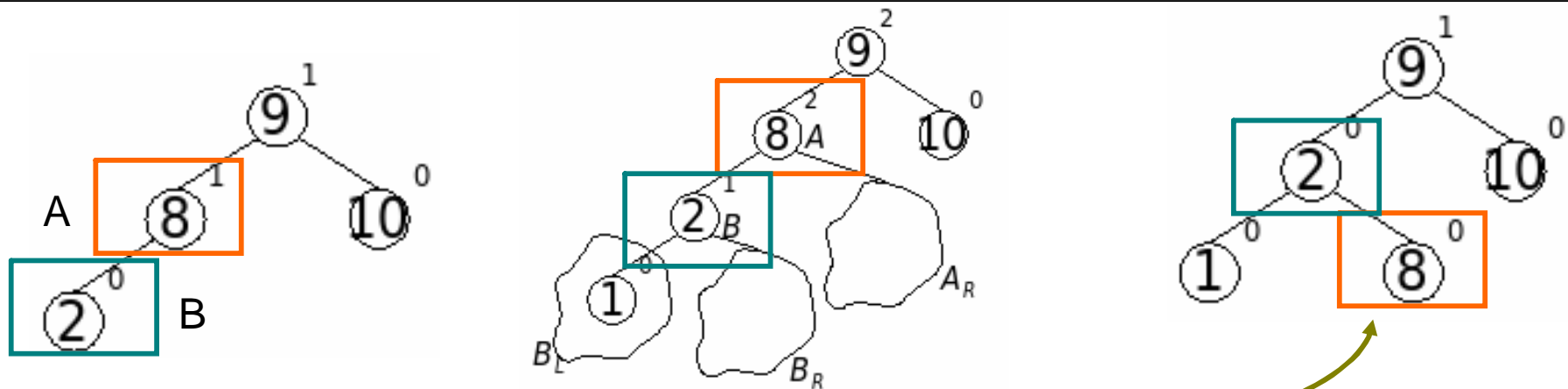
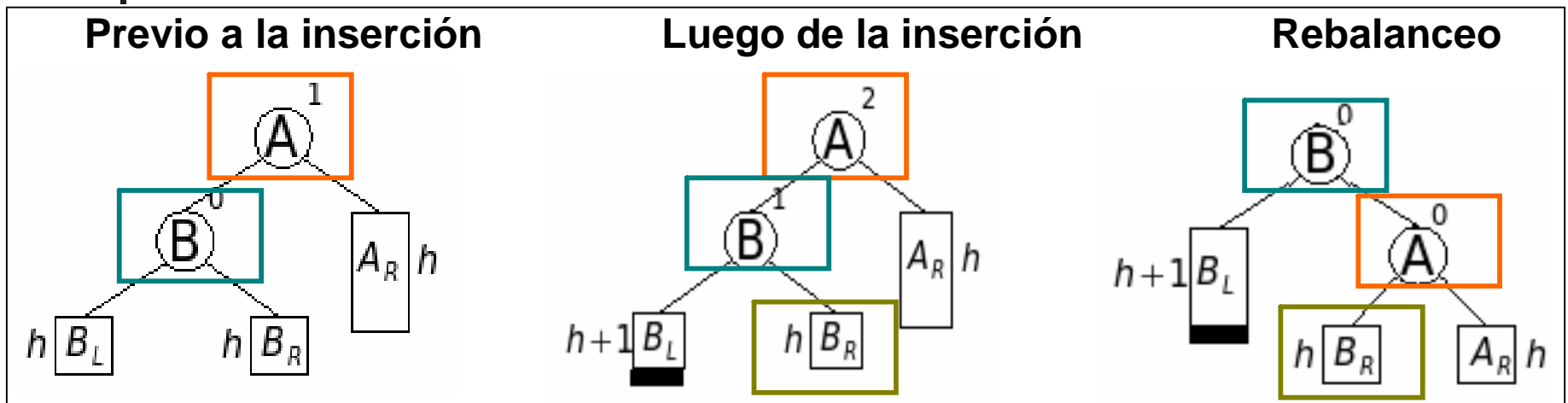
**$B_R$  se convierte en hijo izquierdo de A**

**Notar que**

- $A_R$  sigue siendo hijo derecho de A
- $B_L$  sigue siendo hijo izquierdo de B

# AVL

## Tipos de rotaciones: Simples – Tipo LL

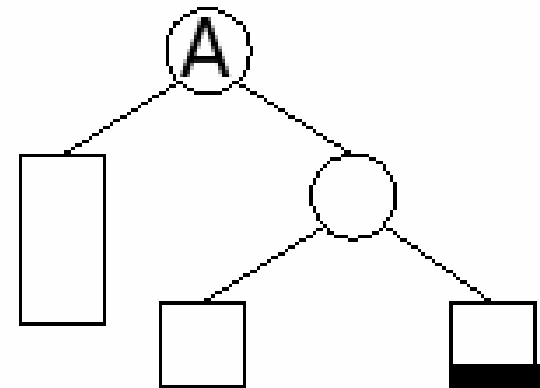


Se inserta el elemento 1

# AVL

## Tipos de rotaciones: Simples – Tipo RR

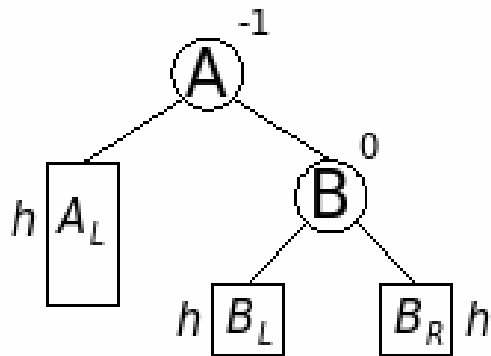
- El nodo fue insertado en el subárbol derecho del subárbol derecho de A.



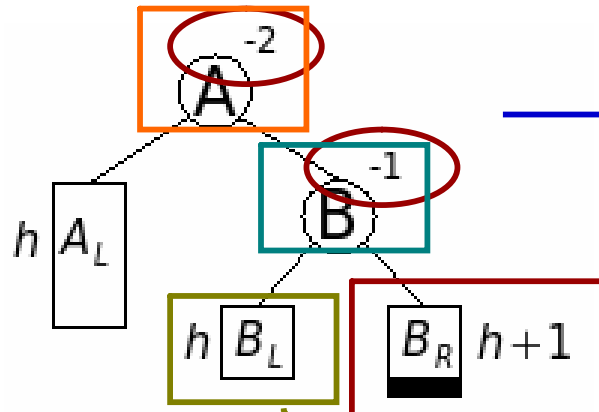
# AVL

## Tipos de rotaciones: Simples – Tipo RR

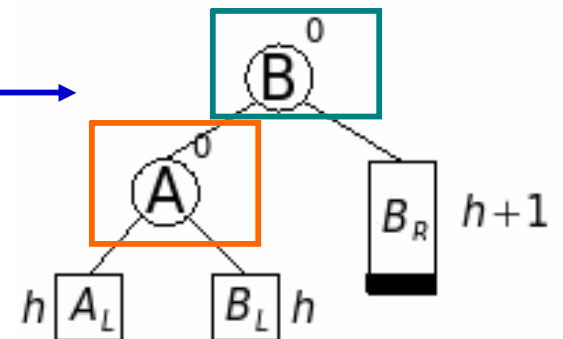
Previo a la inserción



Luego de la inserción



Rebalanceo



**B se convierte en la raíz y A en hijo izquierdo de B**

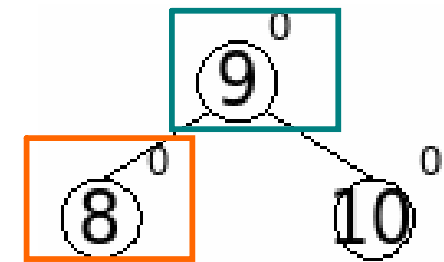
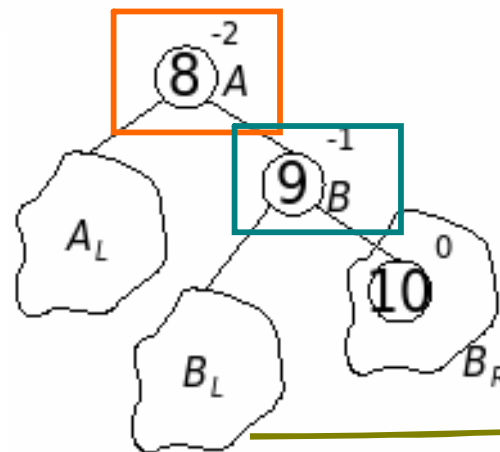
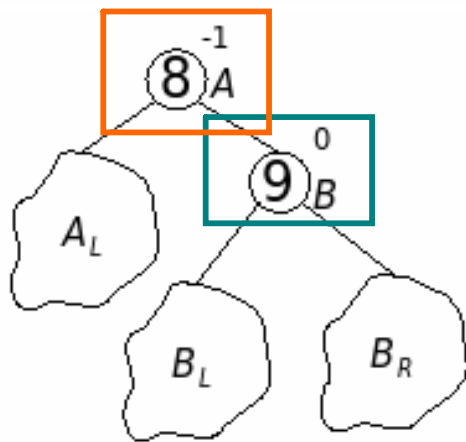
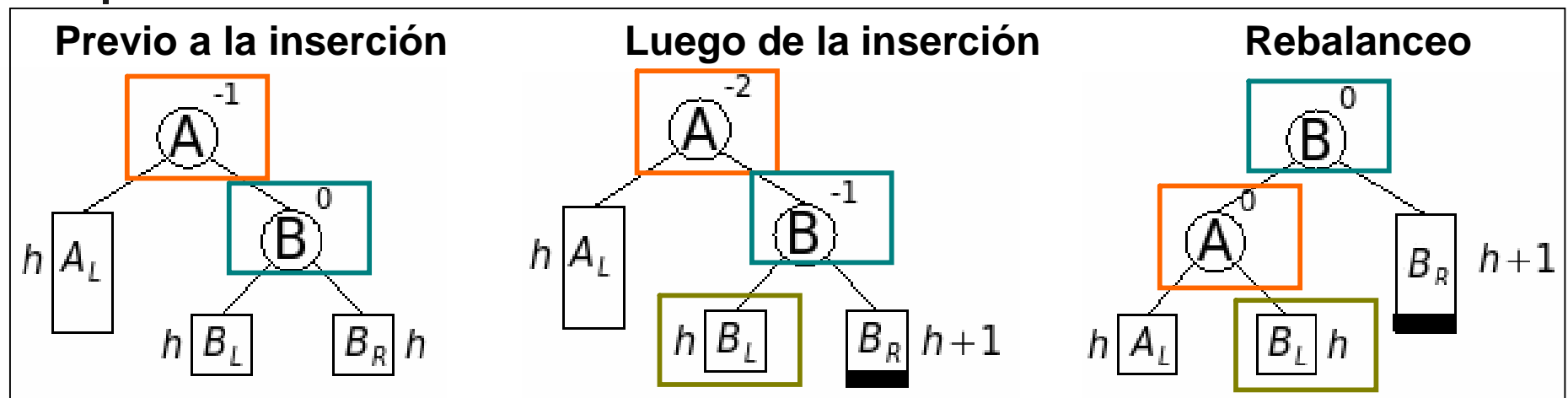
**B<sub>L</sub> se convierte en hijo derecho de A**

**Notar que**

- **A<sub>L</sub> sigue siendo hijo izquierdo de A**
- **B<sub>R</sub> sigue siendo hijo derecho de B**

# AVL

## Tipos de rotaciones: Simples – Tipo RR



Se inserta el elemento 10



# AVL

## Tipos de rotaciones: Dobles

---

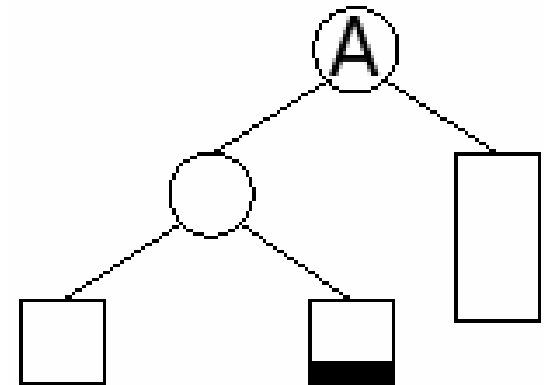
- Rotaciones dobles involucran 4 subárboles.
- Se resuelven aplicando dos rotaciones simples.
- Se clasifican en
  - Tipo LR
  - Tipo RL



# AVL

## Tipos de rotaciones: Dobles – Tipo LR

- El nodo fue insertado en el subárbol derecho del subárbol izquierdo de A.

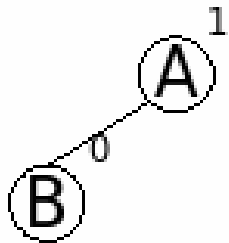


# AVL

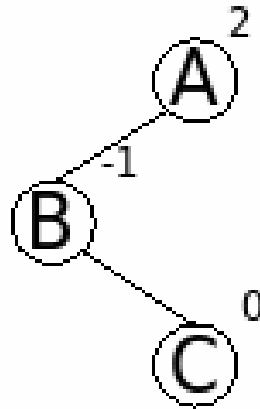
## Tipos de rotaciones: Dobles – Tipo LR

### Caso 1:

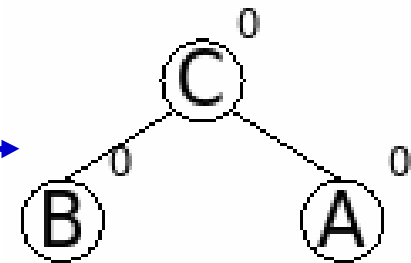
Previo a la inserción



Luego de la inserción



Rebalanceo

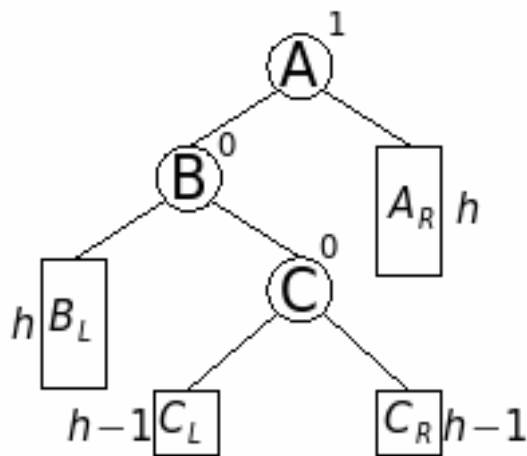


# AVL

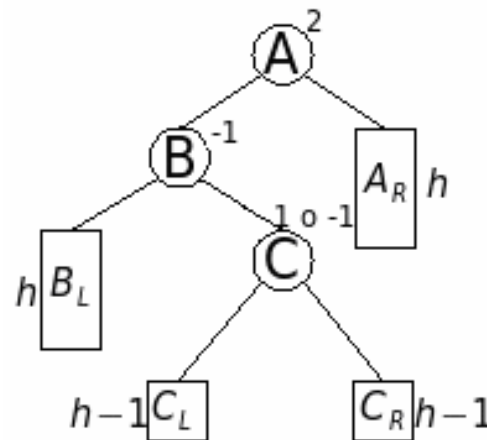
## Tipos de rotaciones: Dobles – Tipo LR

### Caso 2:

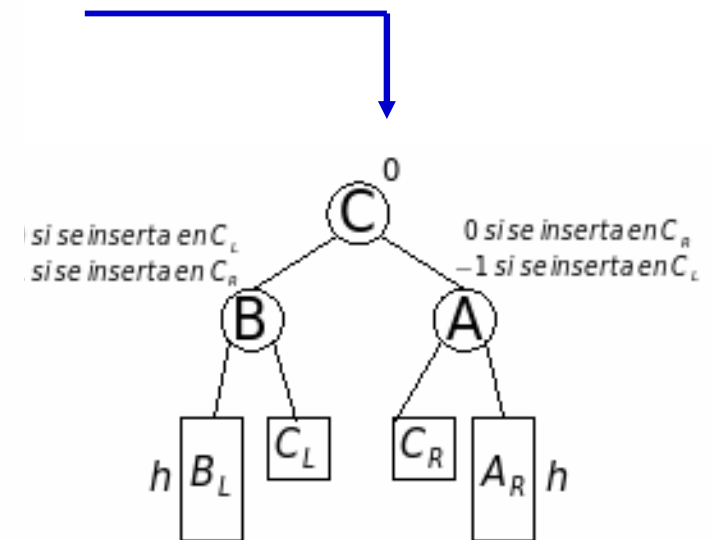
Previo a la inserción



Luego de la inserción



Rebalanceo



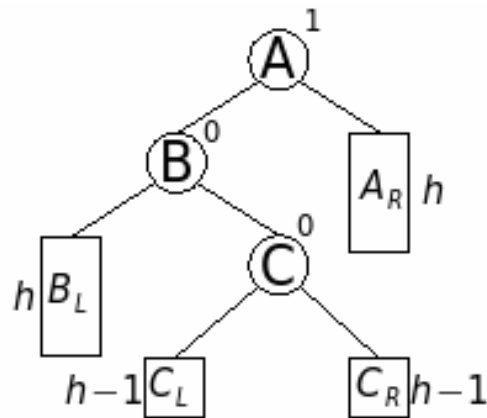
El nodo puede ser insertado en CL o CR.

Si va a CL el FB de C pasa a ser 1, sino va a CR el FB de C pasa a ser -1. En cualquiera de los 2 casos, el FB de B pasa a ser -1 y el de A pasa a ser 2.

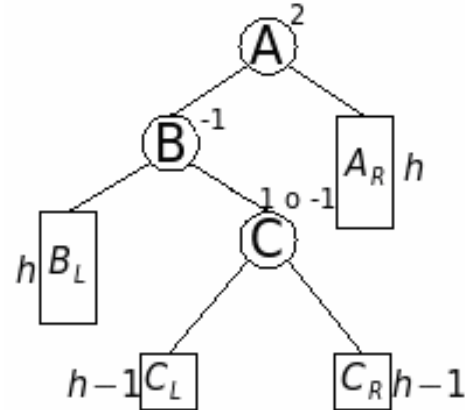
# AVL

## Tipos de rotaciones: Dobles – Tipo LR

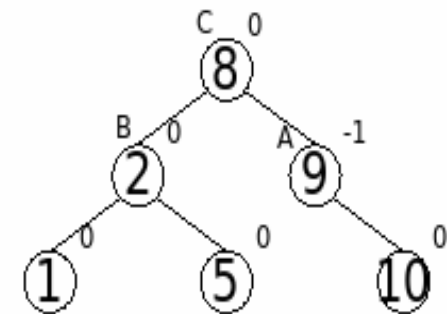
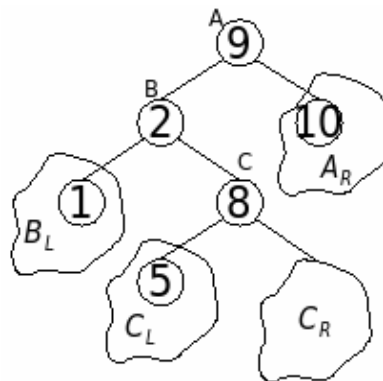
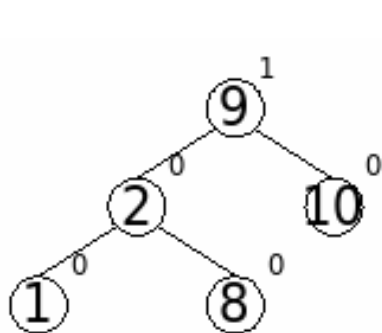
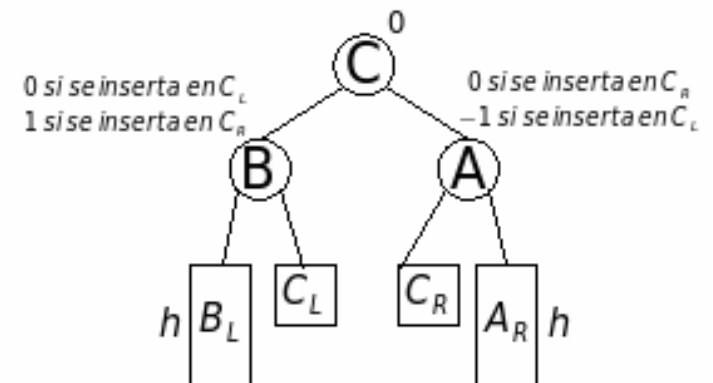
Previo a la inserción



Luego de la inserción



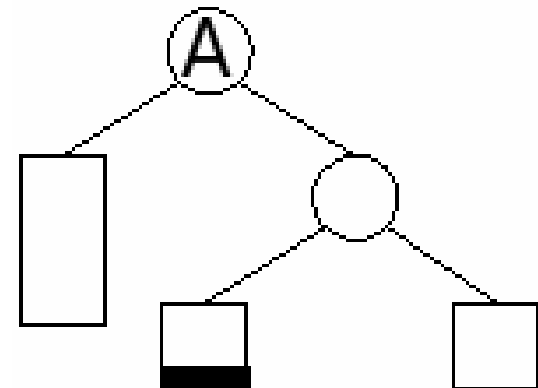
Rebalanceo



# AVL

## Tipos de rotaciones: Dobles – Tipo RL

- El nodo fue insertado en el subárbol izquierdo del subárbol derecho de A.

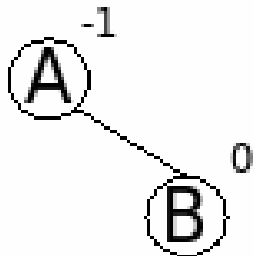


# AVL

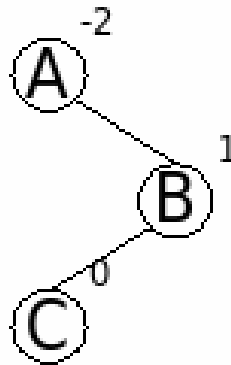
## Tipos de rotaciones: Dobles – Tipo RL

### Caso 1:

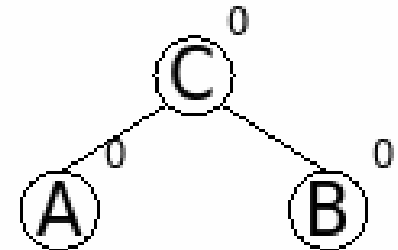
Previo a la inserción



Luego de la inserción



Rebalanceo

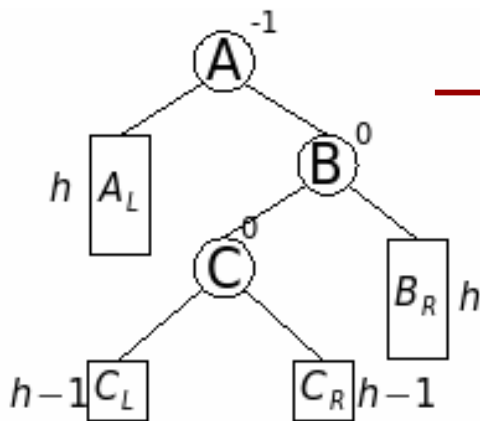


# AVL

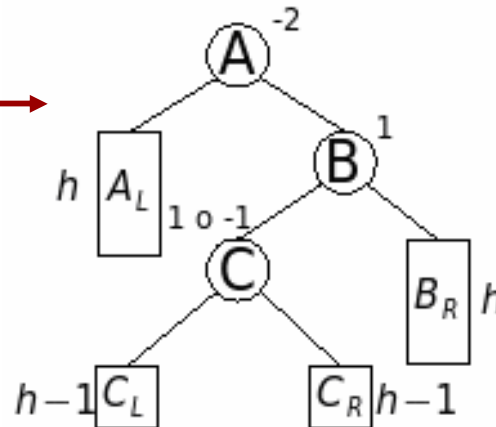
## Tipos de rotaciones: Dobles – Tipo RL

### Caso 2:

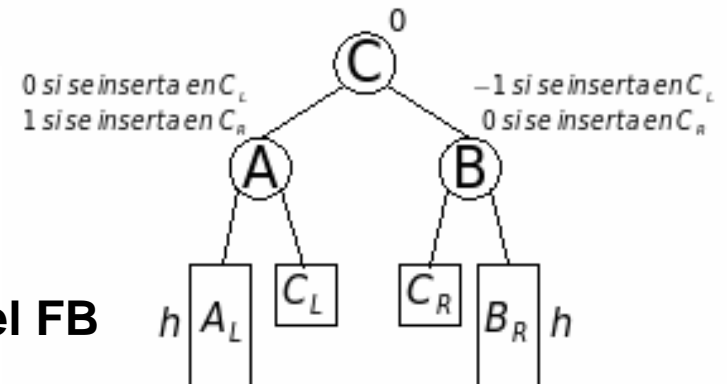
Previo a la inserción



Luego de la inserción



Rebalanceo

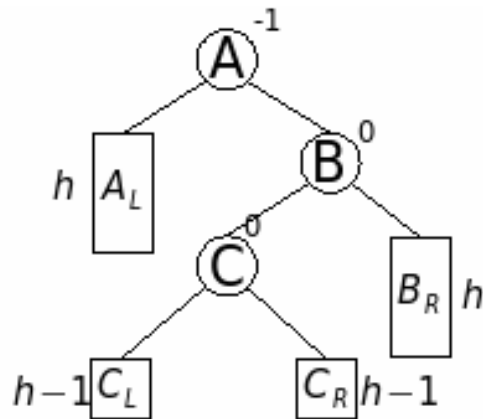


El nodo a insertar puede ir a CL o CR, si va a CL el FB de C pasa a ser 1, si va a CR el FB de C pasa a ser -1. En cualquiera de los 2 casos, el FB de B pasa a ser 1 y el de A pasa a ser -2

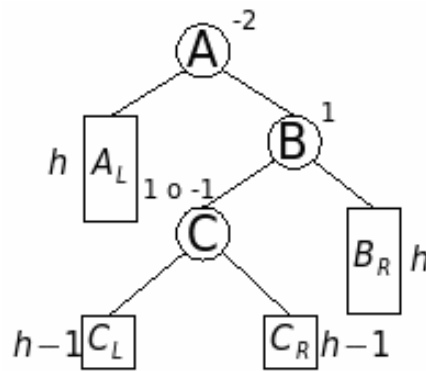
# AVL

## Tipos de rotaciones: Dobles – Tipo RL

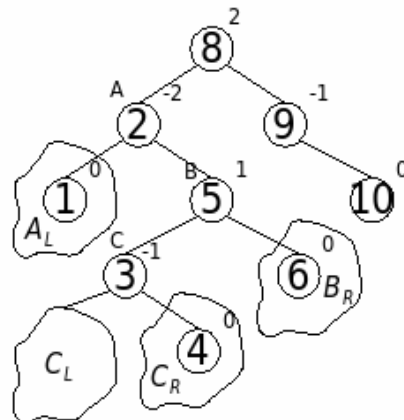
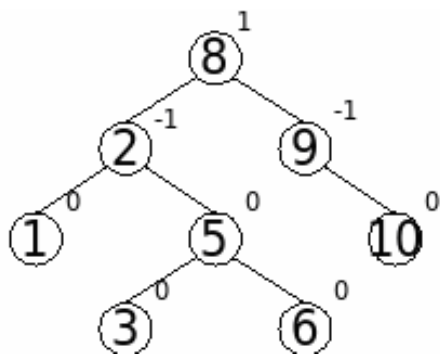
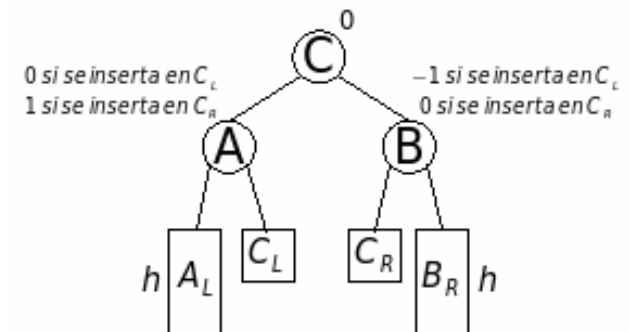
Previo a la inserción



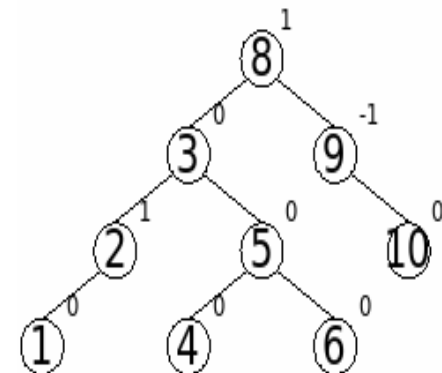
Luego de la inserción



Rebalanceo



amación 3







# AVL

## Algoritmo de Inserción

---

### ■ Observaciones

- ❑ todos los casos de rebalanceo involucran solamente operaciones de intercambios de punteros, por lo cual los algoritmos que implementan los distintos tipos de rebalanceos serán de  $O(1)$ .
- ❑ para poder saber que rebalanceo aplicar es necesario conocer los factores de balance de los nodos involucrados

```
struct nodeAVL
{
    Key clave;
    int FB;
    struct nodeAVL * izq;
    struct nodeAVL * der;
} *AVL;
```



# AVL

## Algoritmo de Inserción

---

- Se divide en 2 etapas:
  - ❑ Insertar el nuevo nodo en el lugar correspondiente
  - ❑ recorrer el camino realizado para insertar el nodo en “vuelta atrás”, chequeando los factores de balanceo.
- Para saber si se produjo desbalanceo se necesita, además de los factores de balanceo, saber si se incrementó o no la altura del subárbol donde se insertó el nodo, para lo cual se utilizará un parámetro booleano.



# AVL

## Algoritmo de Inserción

---

```
void insert(Key x, boolean & aumento, AVL &a){  
    if (Vacio(a)){  
        a = crearNodo(x); //crea un nodo con FB = 0  
        aumento = true;  
    }  
    else  
        //SIGUE!!!!!!
```

```

if (x < a->clave){
    insert(x, aumento, a->izq);
    if (aumento){
        switch(a->FB){
            case -1: //antes de la ins. ALT(TL) < ALT(TR)
                a->FB = 0; //No se produce desbalanceo
                aumento = false; break;
            case 0: //antes de la ins. ALT(TL) = ALT(TR)
                a->FB=1; //mirar los ancestros
                break;
            case 1: //antes de la ins. ALT(TL) > ALT(TR)
                //rebalanceo, el tipo es LL o LR
                if (a->izq->FB == 1) // es LL
                    a = RebalancearLL(a);
                else //es LR
                    a = RebalancearLR(a);
                aumento = false; break;
        } // fin switch
    } // fin if (aumento)
} // fin if (x < a->clave)

```

```

else { insert(x, aumento, a->der);
      if(aumento){
          switch(a->FB){
              case 1: //antes de la ins. ALT(TR) < ALT(TL)
                  a->FB = 0; //No se produce desbalanceo
                  aumento = false; break;
              case 0: //antes de la ins. ALT(TR) = ALT(TL)
                  a->FB = -1; // mirar los ancestros
                  break;
              case -1: //antes de la ins. ALT(TR) > ALT(TL)
                  //rebalanceo, el tipo es RR o RL
                  if (a->der->FB == -1) // es RR
                      a = RebalancearRR(a);
                  else //es RL
                      a = RebalancearRL(a);
                  aumento = false; break;
          } // fin switch
      } // fin if(aumento)
  } // fin else
} // fin if(Vacio(a))
} // fin procedimiento

```



# AVL

## Algoritmo de Rotación Simple

---

```
AVL RebalancearLL(AVL & a)
{
    AVL * auxIzq;
    auxIzq = a->izq;
    a->izq = a->izq->der;
    auxIzq->der = a;
    a->FB = a->FB - 1;
    a = auxIzq;
    a->FB = a->FB - 1;
    return a;
};
```



# Preguntas

---