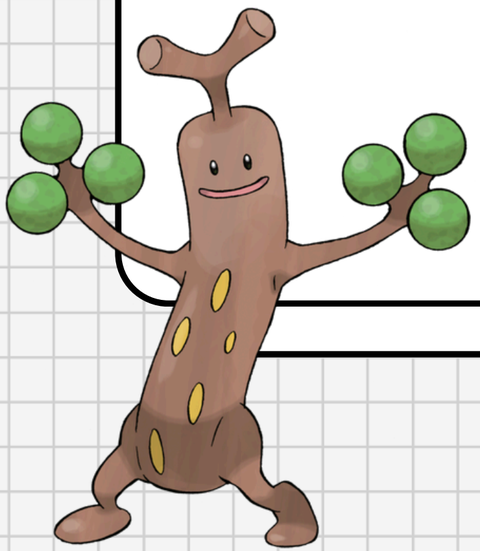
Ayudantía N° 7

# ABB y AVL



# Repaso

Definición

# ¿Qué es un ABB?

Una **árbol binario de búsqueda (ABB)** es una estructura de datos que almacena tuplas (llave, valor) con punteros a sus hojas.

Un ABB se divide en dos ramas, un **hijo izquierdo** y uno **derecho**, que a su vez también son ABBs.

La propiedad más importante de los ABBs es que se cumple que la **llave** almacenada en el **hijo izquierdo** es **menor al padre**, mientras que las **mayores** están en el **hijo derecho**.

Definición

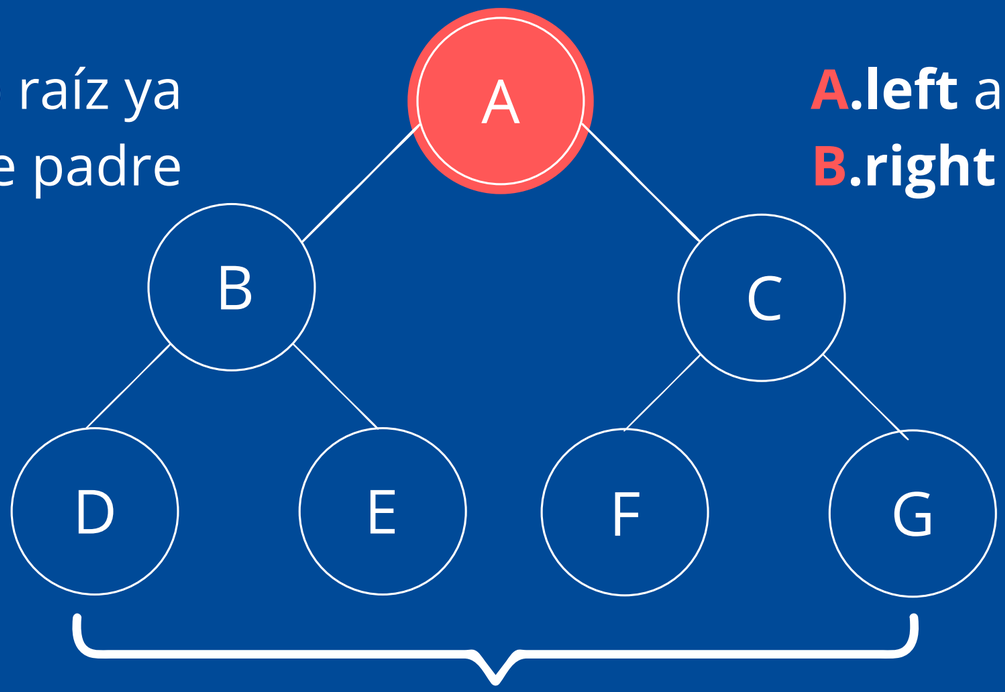
# ¿Qué cumple un árbol binario?

Un AB (de búsqueda o no) cumple con que cada nodo x:

- Si tiene padre, este sea x.p
- Si no tiene padre, x es la raíz
- Tiene dos punteros a nodos hijos izquierdo (x.left) y derecho (x.right), si existen
- Si x no tiene “hijos”, entonces x es una hoja

# Ejemplo de AB

**A** es el nodo raíz ya que no posee padre

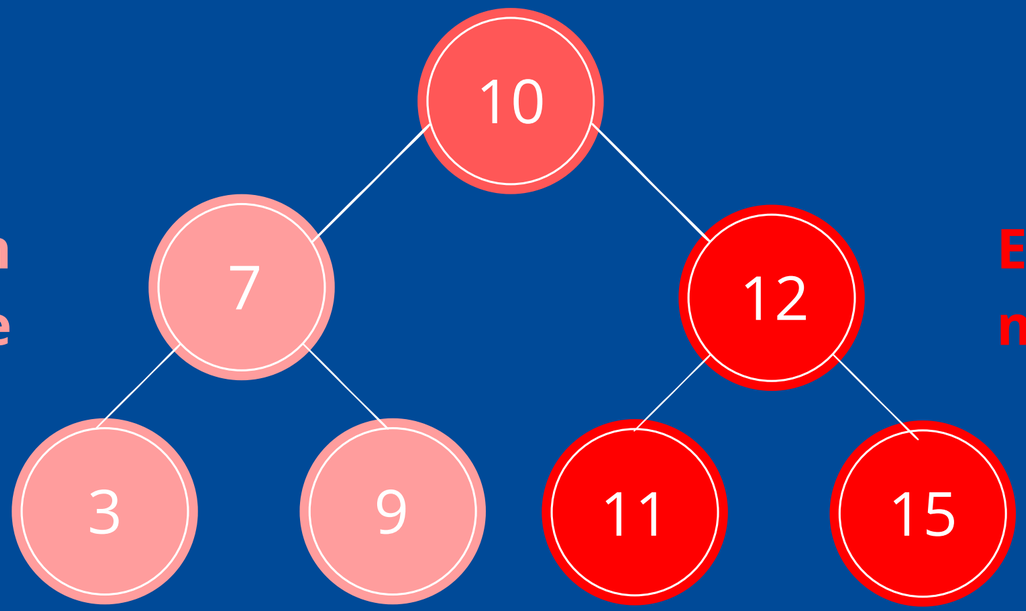



**A.left** apunta a B
**B.right** apunta a C

**D, E, F, G** son hojas (no poseen hijos)

# Ejemplo de ABB

**Estos nodos son menores al padre**

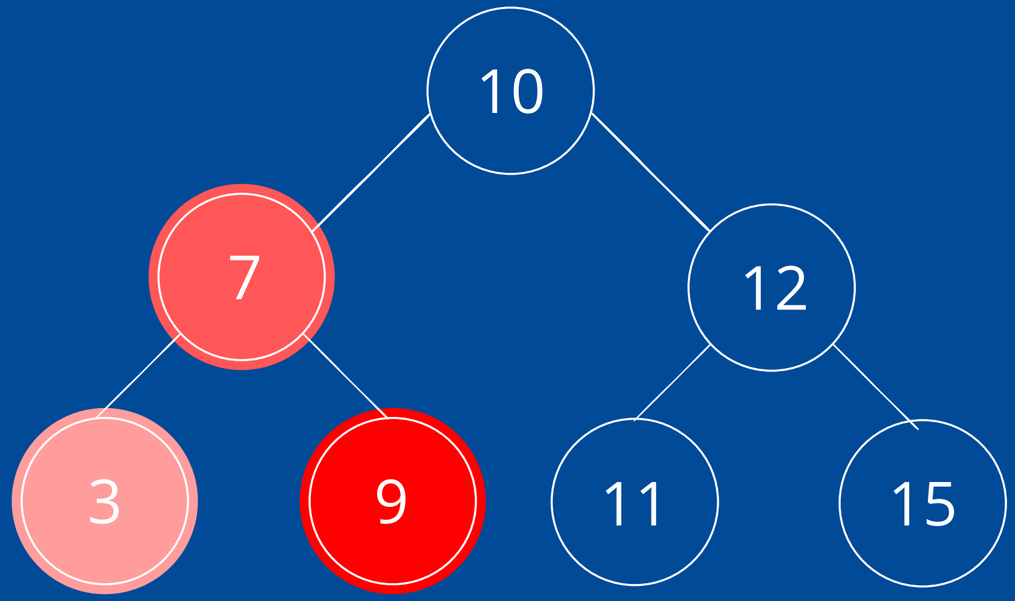



**Estos nodos son mayores al padre**

**Ahora cumplimos propiedades de ABB!**

# Ejemplo de ABB

**Es importante recordar que los hijos también serán ABBs**




**Este nodo es menor al padre**

**Este nodo es mayor al padre**

Pseudocódigo

# Operaciones de ABB

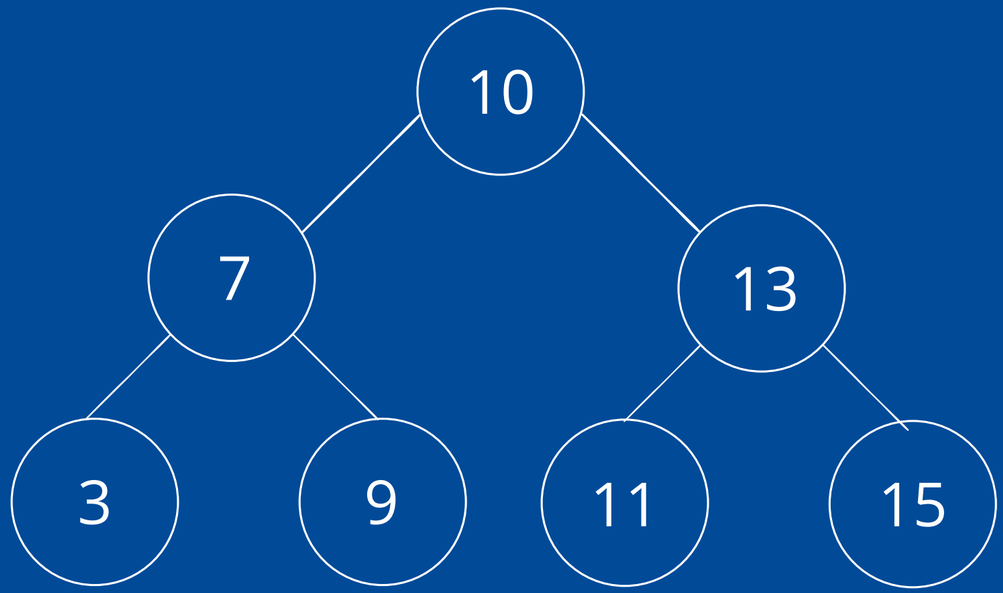
```
Search(A, p, k):
1     if A = ∅ ∨ A.key = k :
2         return (A, p)
3     if k < A.key :
4         return Search(A.left, A, k)
5     return Search(A.right, A, k)
```

Recibimos de input un ABB A, el ABB padre p y la llave que estamos buscando, k.

# Search en ABB

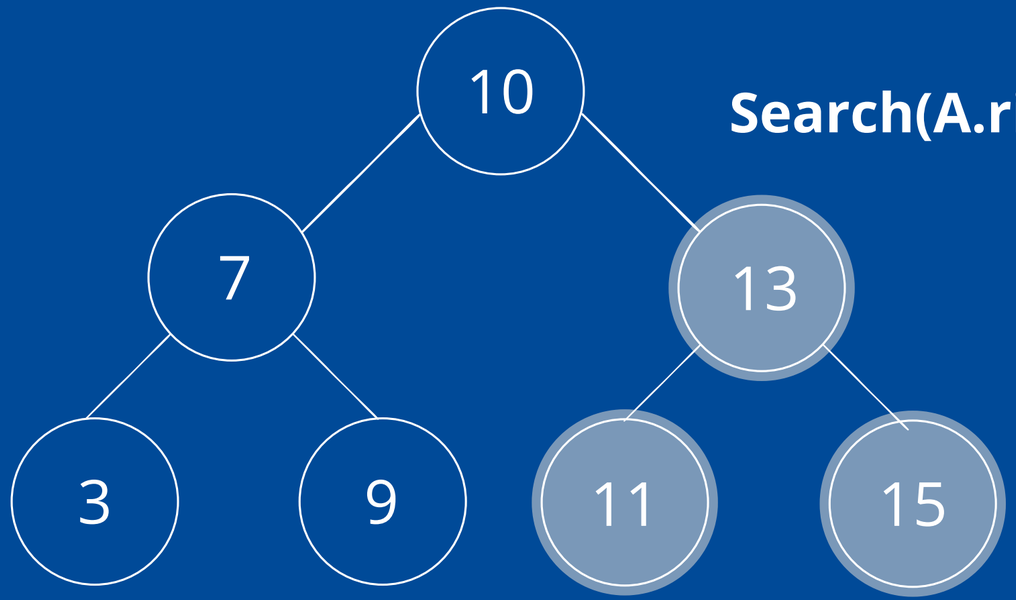
**Digamos que deseo buscar un nodo con key 12**

**Search(A, Ø, 12)**



# Search en ABB

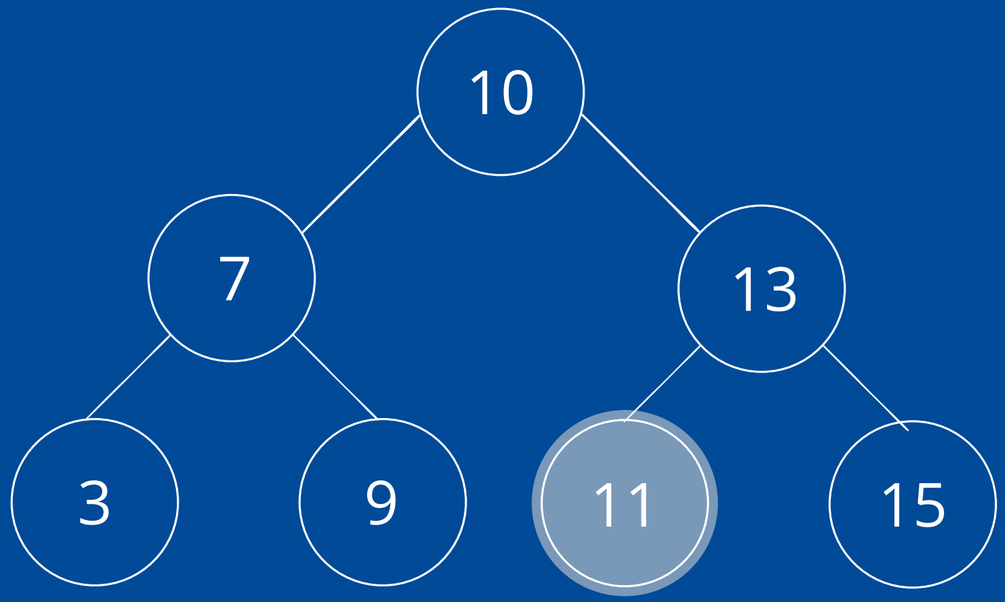
**Dado que 12 es mayor a 10, buscamos en la rama derecha**




**Search(A.right, A, 12)**

# Search en ABB

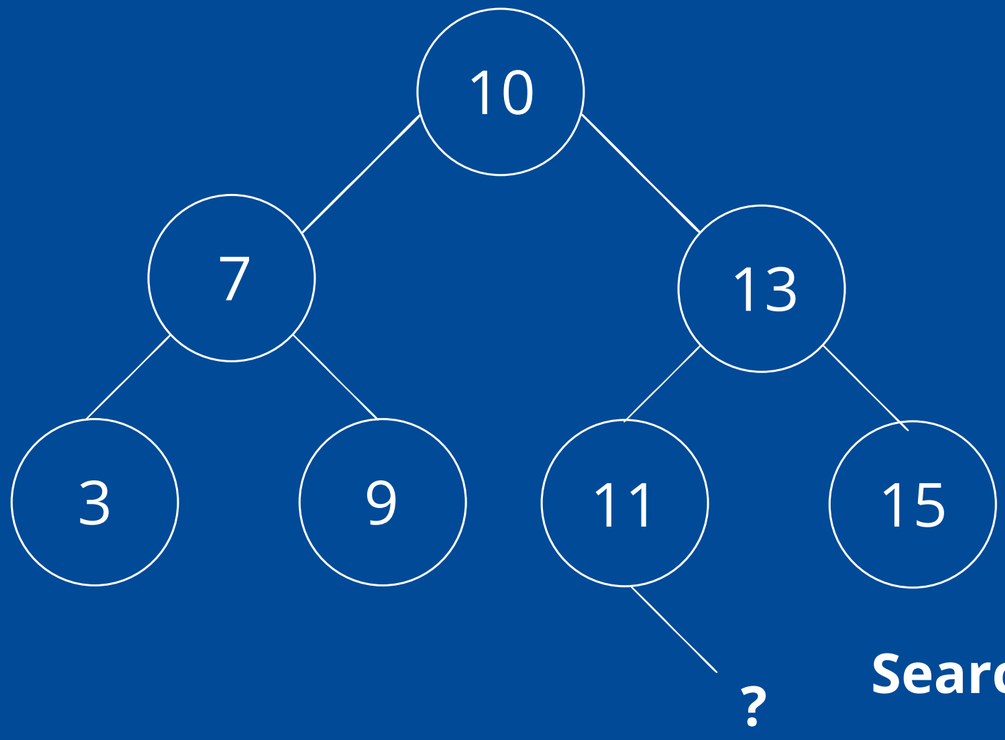
**Dado que 12 es menor a 13, buscamos en la rama izquierda**




**Search(A.right.left, A.right, 12)**

# Search en ABB

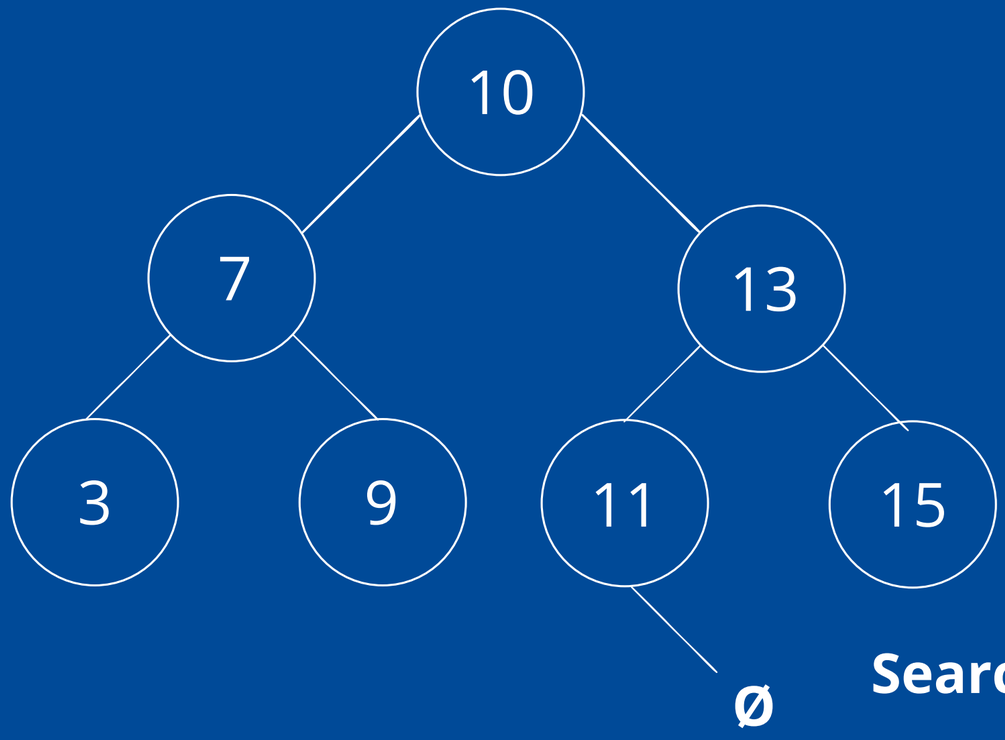
**Como 11 no tiene hijos, A.right.left.right=Ø**




**Search(A.right.left.right, A.right.left, 12)**

# Search en ABB

**Por lo tanto, retornamos (A.right.left.right, A.right.left)**




**Search(A.right.left.right, A.right.left, 12)**

Pseudocódigo

# Operaciones de ABB

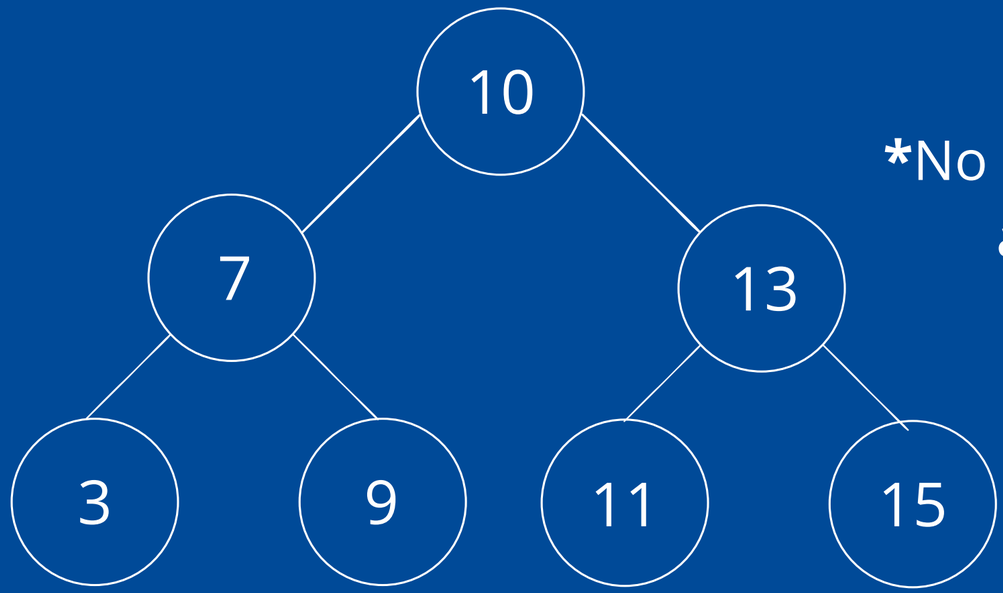
```
Insert (A, k, v):
1    (B, p) ← Search(A, ∅, k)    ▷ versión que indica el padre
2    if B = ∅ :
3        B ← nodo vacío
4        B.key ← k
5        Conectar B al padre p en la posición adecuada
6    B.value ← v
```

Recibimos de input un ABB A, la llave que queremos insertar k y su valor v.

# Insert en ABB

**Digamos que deseo buscar un nodo con key 12**

**Insert(A, 12, v)**

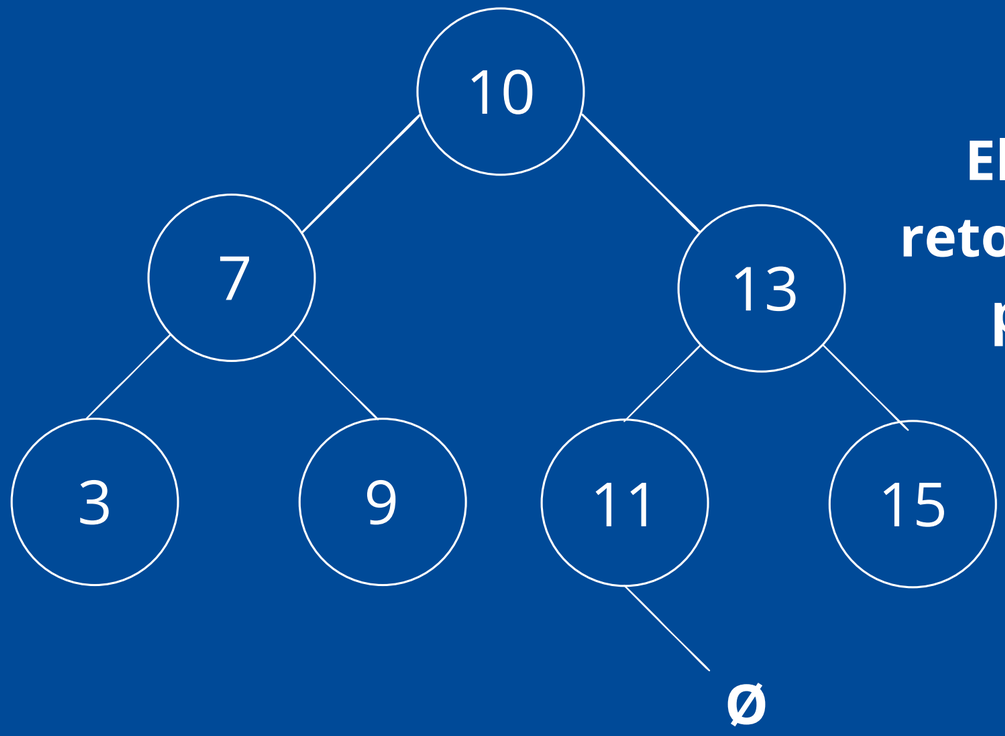



***No le daremos relevancia al valor por ahora**

# Insert en ABB

**Digamos que deseo buscar un nodo con key 12**

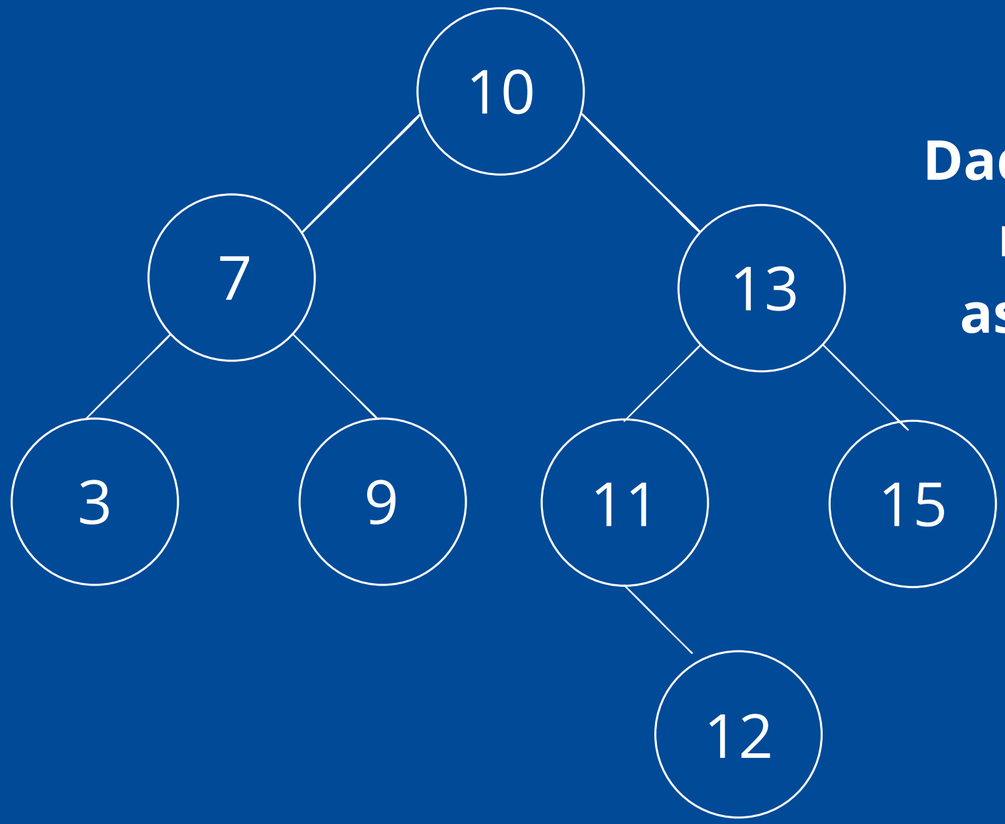
**Insert(A, 12, v)**




**El Search de 12 nos retornó (Ø, A.right.left), por lo tanto B=Ø, p=A.right.left**

# Insert en ABB

**Insert(A, 12, v)**

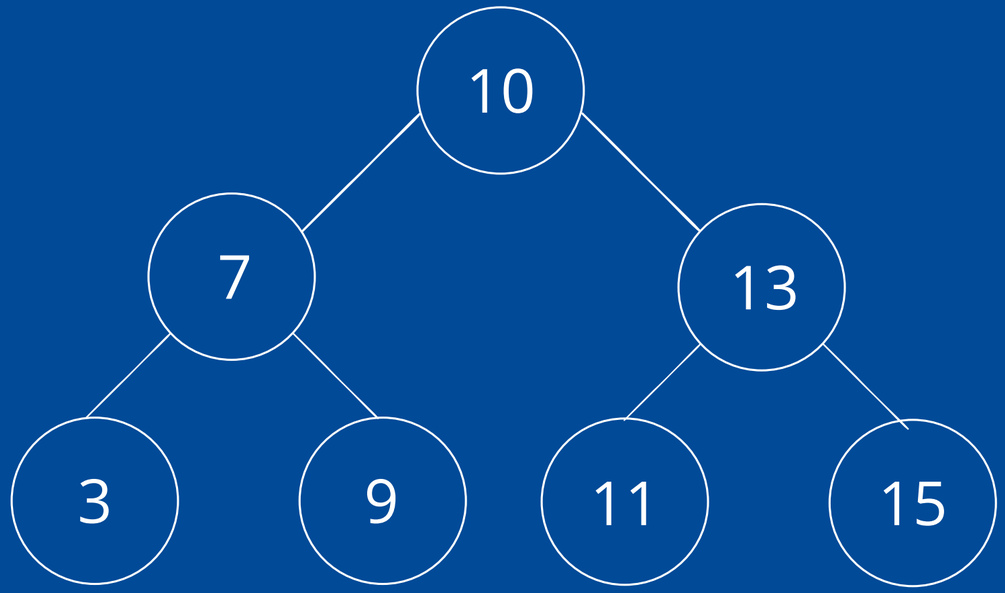



**Dado eso, creamos un nodo nuevo b, le asignamos key 12 y value v**

# Insert en ABB

**Si quisieramos insertar un valor con key ya existente, por convención del código, solamente se actualiza v**

**ej: Insert(A, 13, v)**



Pseudocódigo

# Operaciones de ABB

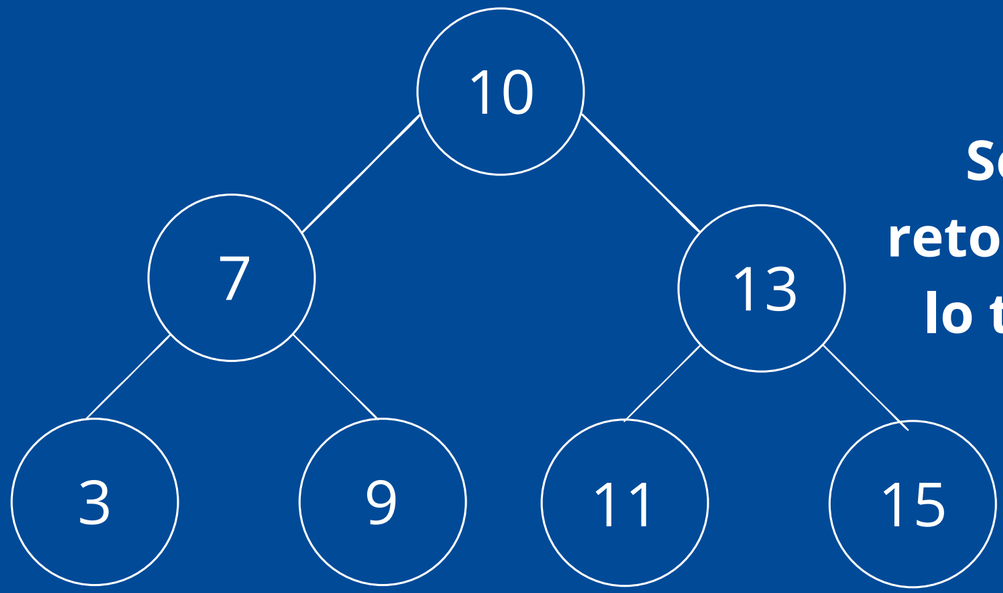
```
Delete (A, k):
1    (D, p) ← Search(A, ∅, k)    ▷ Permite saber el padre de D
2    if D ≠ ∅ :
3        if D es hoja :  D ← ∅ y se elimina la referencia en p
4        elif D tiene un solo hijo H :  D ← H y se actualiza p
5        else:
6            R ← Min(D.right)
7            t ← R.right
8            D.key ← R.key
9            D.value ← R.value
10           R ← t
```

Recibimos de input un ABB A, la llave que queremos eliminar k.

# Delete en ABB

**Digamos que deseo eliminar el nodo con key 13**

**Delete(A, 13)**



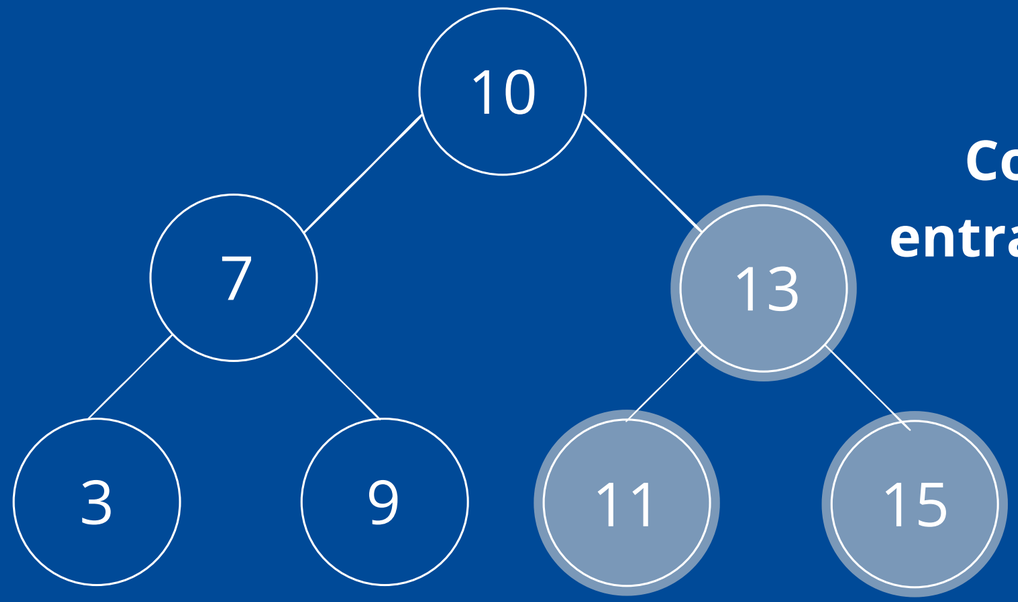

**Search(A, Ø, 13) nos retornará (A.right, A). Por lo tanto D=A.right, p=A**

# Delete en ABB

**Digamos que deseo eliminar el nodo con key 13**

**Delete(A, 13)**

```
else:
    R ← Min(D.right)
    t ← R.right
    D.key ← R.key
    D.value ← R.value
    R ← t
```




**Como D posee 2 hijos, entramos al else del código**

*Diremos que este subarbol es D

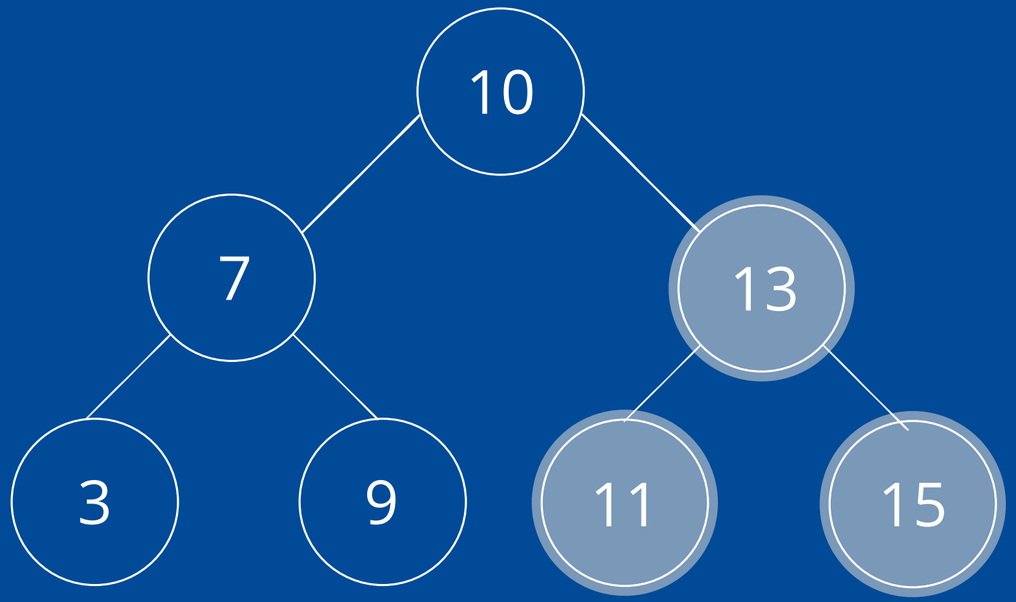
# Delete en ABB

**Digamos que deseo eliminar el nodo con key 13**

**Delete(A, 13)**

**else:**

$R \leftarrow \texttt{Min}(D.right)$
$t \leftarrow R.right$
$D.key \leftarrow R.key$
$D.value \leftarrow R.value$
$R \leftarrow t$




**Entonces,**
**R = D.right**
**t = Ø**
**D.key = 11**
**D.value = v**
**R = Ø**

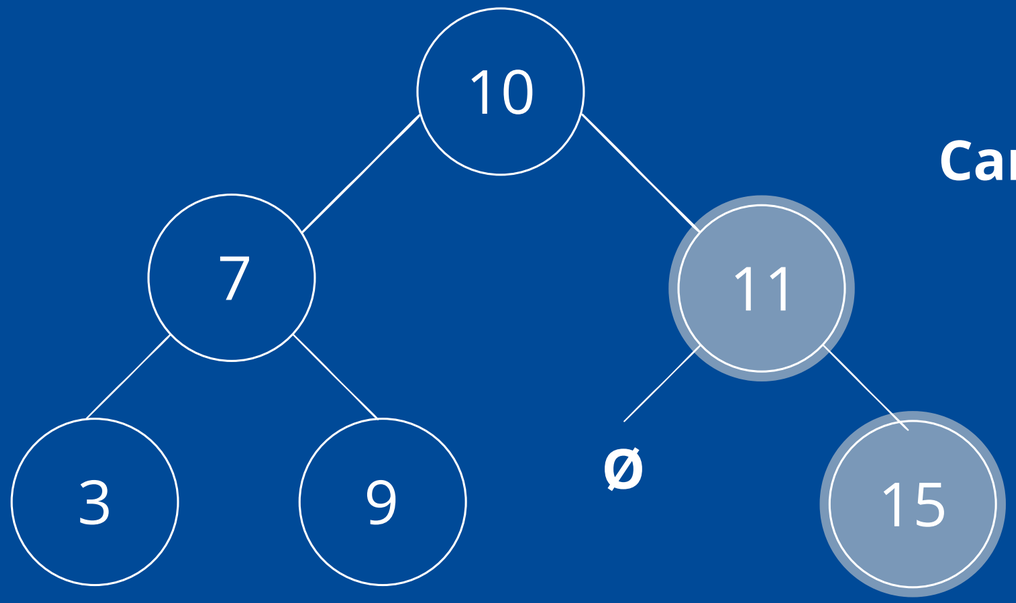
# Delete en ABB

**Digamos que deseo eliminar el nodo con key 13**

**Delete(A, 13)**

**else:**

$R \leftarrow \texttt{Min}(D.right)$
$t \leftarrow R.right$
$D.key \leftarrow R.key$
$D.value \leftarrow R.value$
$R \leftarrow t$




**Cambiamos en el árbol:**

**R = D.right**

**t = Ø**

**D.key = 11**

**D.value = v**

**R = Ø**

Definición

# Árboles AVL-balanceados

Un árbol binario de búsqueda (ABB) está balanceado si:

1. Las alturas entre sus hijos difieren **a lo más en 1** entre sí
2. Cada hijo está AVL-balanceado

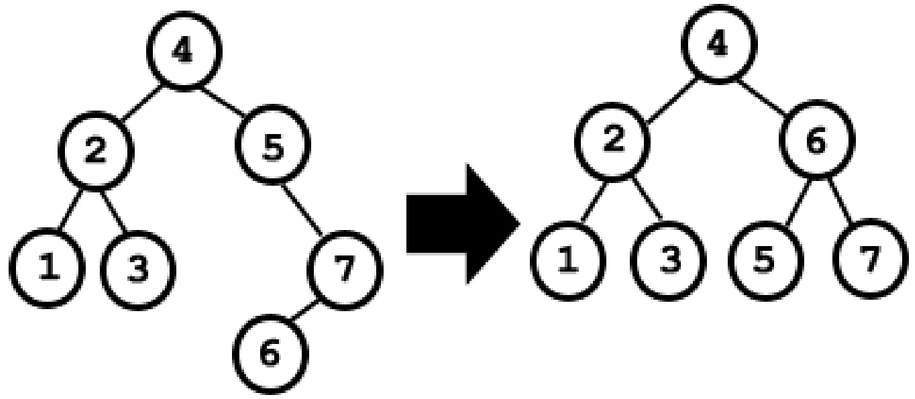
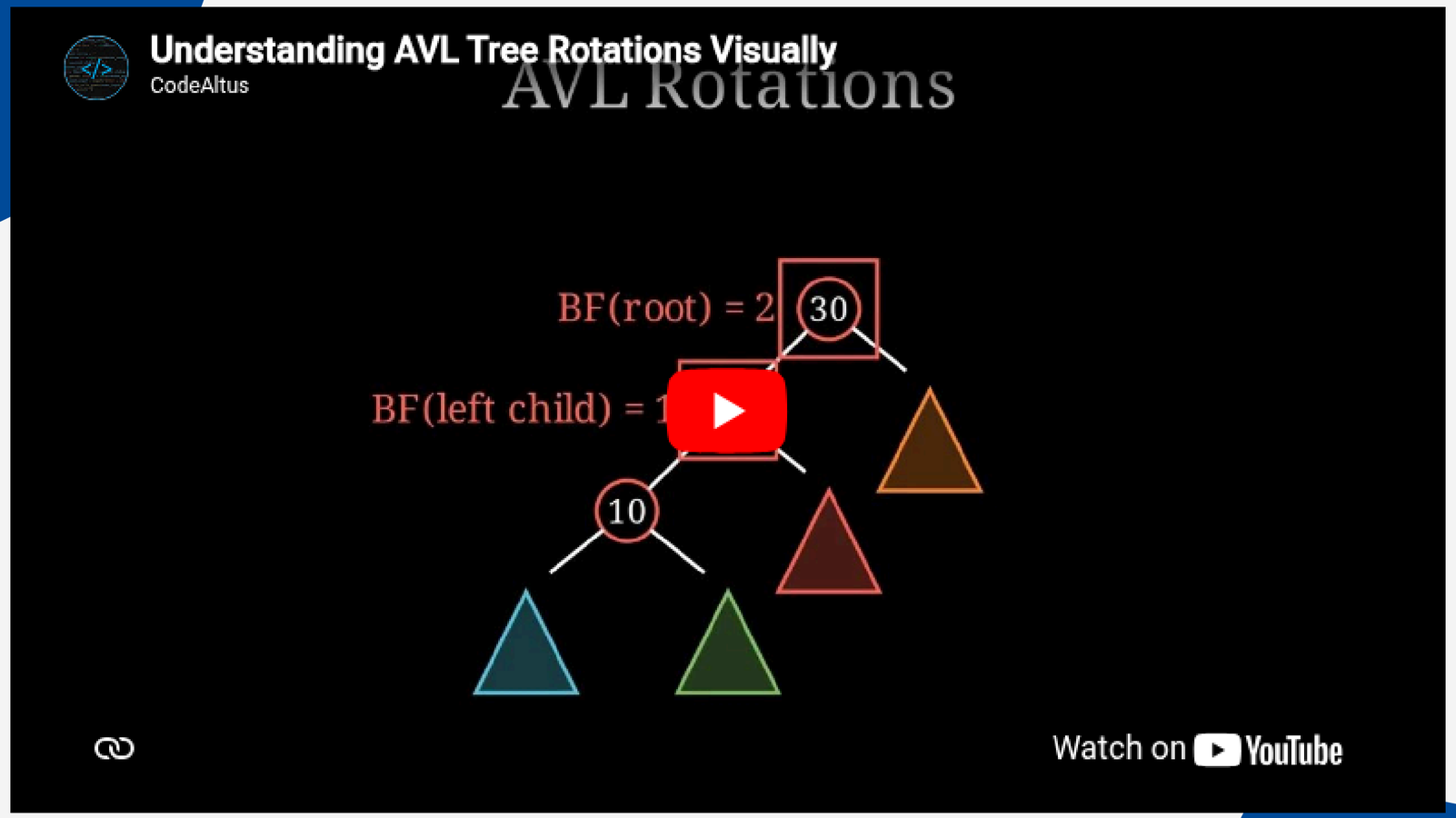



ABB v/s ABB AVL-balanceado

Rep. visual

# Árboles AVL-balanceados



# Ejercicio 1

b) (2 pts) El Profesor Eterovic indica que ha descubierto una propiedad notable de los ABB. Suponga que la búsqueda de la clave k en un ABB termina en una hoja. Consideremos tres conjuntos: A, las claves a la izquierda de la ruta de búsqueda; B, las claves en la ruta de búsqueda; y C, las claves a la derecha de la ruta de búsqueda. El profesor Eterovic afirma que tres claves cualesquiera $a \in A, b \in B$ y $c \in C$ deben satisfacer $a \leq b \leq c$. ¿Será cierto? En caso afirmativo, demuestre formalmente esta propiedad; en caso negativo, muestre con un ejemplo que la propiedad no se cumple.

# Pregunta 2 - I1 2025-01

# Ejercicio 1 - Solución

La propiedad **no es cierta**, por lo que basta mostrar un contraejemplo: considera el ABB con raíz 4 e hijos 3 y 6, en que 6 tiene a su vez hijos 5 y 7. Considera la búsqueda del 7: ruta 4, 6, 7 $\Rightarrow$ a la izquierda quedan las claves 3 y 5, en que 5 > 4

# Ejercicio 2

**4)** Árboles binarios de búsqueda

a) Dado un **ABB** (no necesariamente balanceado), una forma —distinta de la estudiada en clases— de eliminar una clave cuyo nodo tiene ambos hijos es la siguiente: el nodo es ocupado por su (hijo y) subárbol izquierdo, *Ti*, mientras que su subárbol derecho, *Td*, se convierte en el subárbol derecho del nodo más a la derecha de *Ti*.

i) [1 pt.] Justifica que esta eliminación respeta las propiedades de ABB.

ii) [0.5 pts.] Muestra con ejemplos que esta eliminación puede tanto aumentar como reducir la altura del árbol original (dos ejemplos, uno para cada caso).

# Pregunta 4 - I1-2025-2

# Ejercicio 2 - Solución

a) i) A partir de la regla para insertar claves, que, a su vez, cumple la propiedad fundamental de ABB, sabemos que en el proceso de inserción de cualquiera de las claves que están en Ti o Td —llamemos *k* a una clave cualquiera en Ti o Td— pasamos por la clave —llamémosla *j*— que estamos eliminando, y que, por lo tanto, *k* podría haber ido a parar al lugar de *j*, posición en la que habría cumplido la propiedad de ABB con respecto al resto del árbol. Por lo tanto, poner el subárbol Ti en la posición que ocupaba el nodo con clave *j* es válido.

La pregunta entonces es, ¿qué hacemos con Td? De nuevo, por la propiedad de ABB, las claves de Td son todas mayores que las claves de Ti. La única posición que corresponde a claves mayores que todas las claves de Ti, pero al mismo tiempo menores que las otras claves del árbol mayores que *j* es como hijo derecho de la clave más a la derecha de Ti —llamémosla *m*; obviamente, esta posición está "desocupada": *m* sólo puede ser la clave más a la derecha de Ti si no tiene hijo derecho.

# Ejercicio 2 - Solución

a) ii) Para simplificar (y generalizar un poco), eliminamos la raíz del árbol. Entonces, Ti "sube" a esta posición y agregamos Td como hijo derecho del nodo más a la derecha de Ti.

La altura del árbol original, T, era $H(T) = \max\{ H(Ti), H(Td) \} + 1$. La altura del nuevo árbol, T', puede ser desde $H(T') = H(Ti)$ hasta $H(T') = H(Ti) + H(Td)$, dependiendo de la profundidad del nodo más a la derecha de Ti. En el primer caso, H(T') es claramente menor que H(T); en el segundo, H(T') claramente puede ser mayor que H(T).

# Ejercicio 3

b) [1.5 pts.] Considerando que un nodo por sí solo —es decir, un árbol que sólo tiene la raíz— tiene ***profundidad*** 0, dibuja **árboles AVL** de profundidades 2 y 4 tales que *tengan el mínimo número de nodos posible*. Recuerda que un árbol AVL con el mínimo número de nodos está compuesto por una raíz y por dos subárboles AVL, cada uno a su vez con el mínimo número de nodos; comprueba que tus árboles cumplen esta propiedad.

**Pregunta 4 - I1-2025-2**

# Ejercicio 3 - Solución

b) Lo primero que hay que precisar es lo siguiente. Respecto de la frase

"Recuerda que un árbol AVL con el mínimo número de nodos está compuesto por una raíz y por dos subárboles AVL, cada uno a su vez con el mínimo número de nodos…"

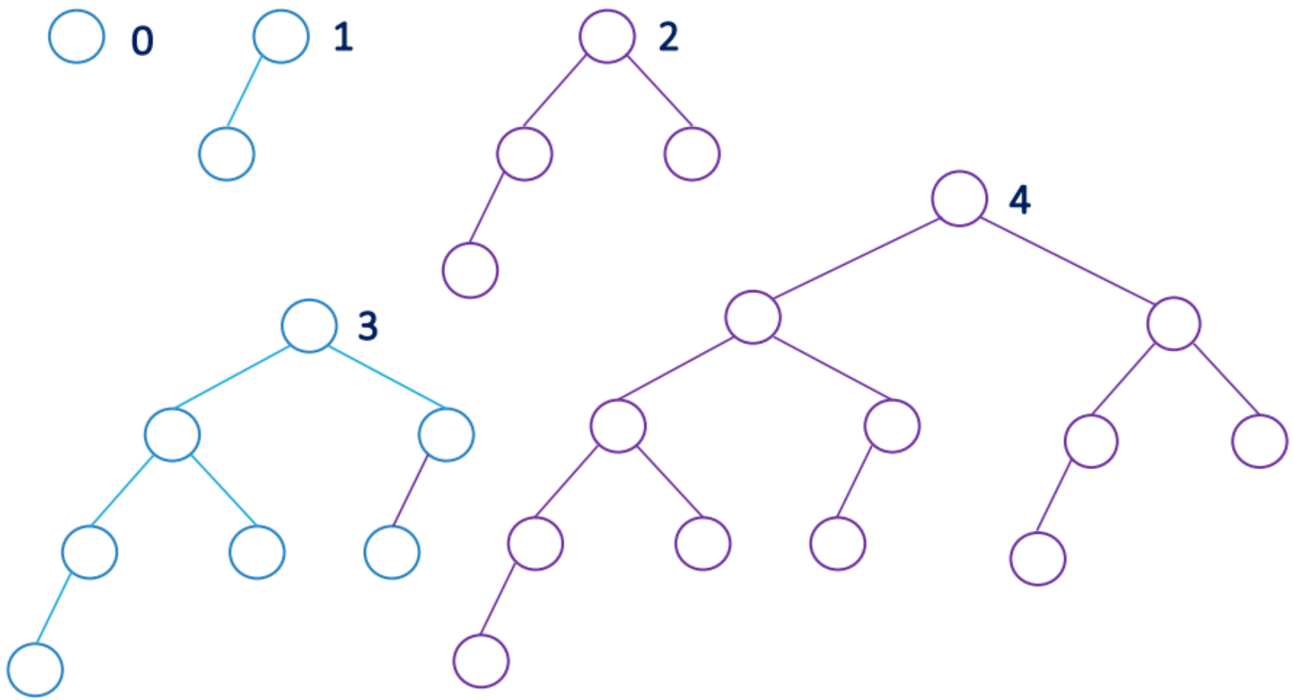
… si el árbol AVL tuviera profundidad $D$, entonces, para que su número de nodos sea el mínimo posible, los dos subárboles AVL mencionados tienen que tener profundidades $D-1$ y $D-2$ (además del menor número posible de nodos). Es decir, el árbol pedido de profundidad 2 está compuesto por un subárbol de profundidad 1 y otro de profundidad 0; y el árbol de profundidad 4 está compuesto por un subárbol de profundidad 3 y otro de profundidad 2.

Así, si un nodo por sí solo es un árbol de profundidad 0, entonces un árbol de profundidad 1 *con el mínimo número de nodos* es un nodo con un único hijo (dos nodos en total); y **el árbol de profundidad 2 pedido** es un nodo (una raíz) con dos hijos: uno de profundidad 1 (dos nodos) y el otro de profundidad 0 (un solo nodo), como los recién descritos.

Ahora podemos construir un árbol de profundidad 3 *con el mínimo número de nodos*, consistente en un nodo (raíz) con dos hijos: el árbol de profundidad 2, del párrafo anterior, y el árbol de profundidad 1, también del párrafo anterior.

Finalmente, **el árbol de profundidad 4 pedido** se construye como un nodo (raíz) con dos hijos: el árbol de profundidad 3 y el árbol de profundidad 2.

# Ejercicio 3 - Solución




En la fig., los números representan las profundidades de los árboles correspondientes. Los árboles pedidos son los de profundidad 2, con 4 nodos, y profundidad 4, con 12 nodos; el número de nodos es el mínimo en cada caso.

# Ejercicio 4

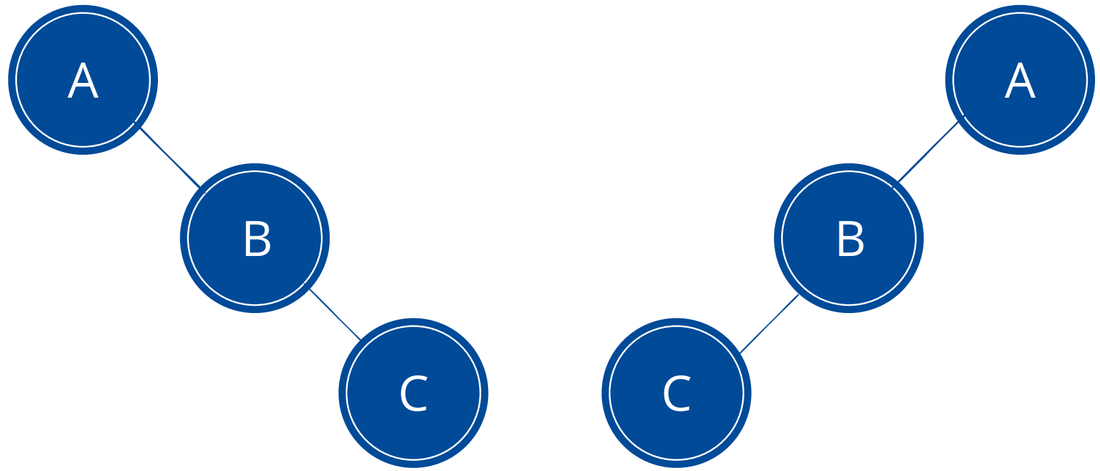
a) (2pts) Considere un árbol AVL inicialmente vacío:

(i) Proponga una secuencia de inserciones de claves tal que requiera lo antes posible una rotación simple para mantener el balance del árbol.

## Pregunta 2 - I1-2025-1

# Ejercicio 4 - Solución

Si insertamos cualquier secuencia de tres números a<b<c o a>b>c quedaremos con un árbol así:



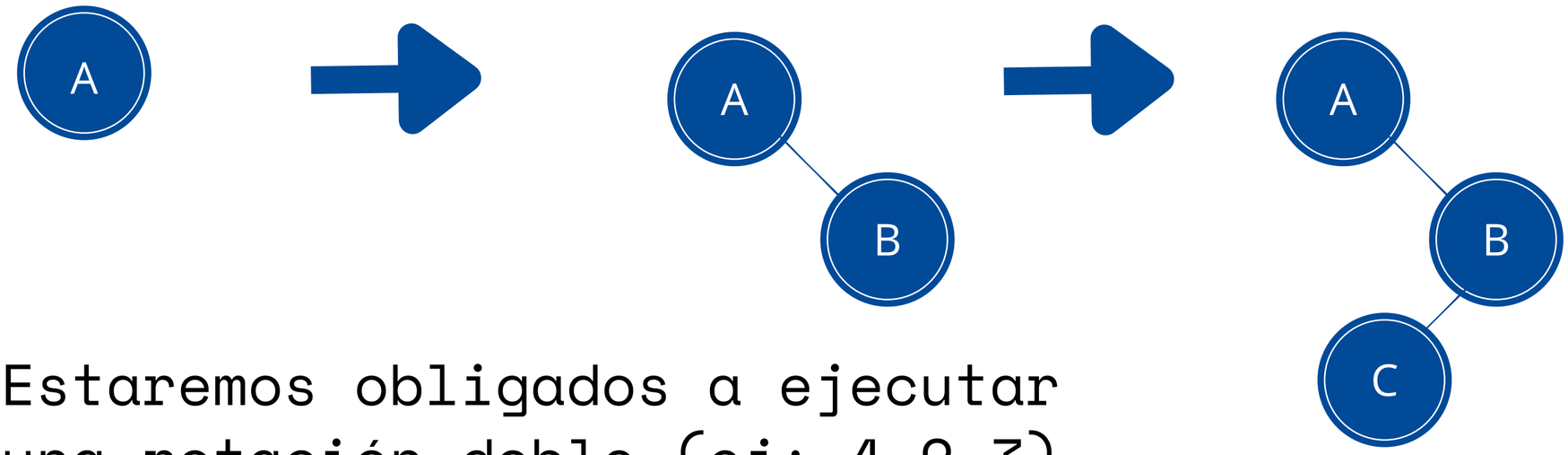

Por lo que estaremos obligados a hacer una rotación

(II) Proponga una secuencia de inserciones de claves tal que requiera lo antes posible una rotación doble para mantener el balance del árbol. Dibuja paso a paso el proceso de inserciones.

## Pregunta 2 - I1-2025-1

# Ejercicio 4 - Solución

Los casos de desbalance que terminan en balanceo doble son izquierda+derecha o derecha+izquierda.
Si hacemos el siguiente paso de inserciones:




Estaremos obligados a ejecutar una rotación doble (ej: 4,2,3)

# Feedback