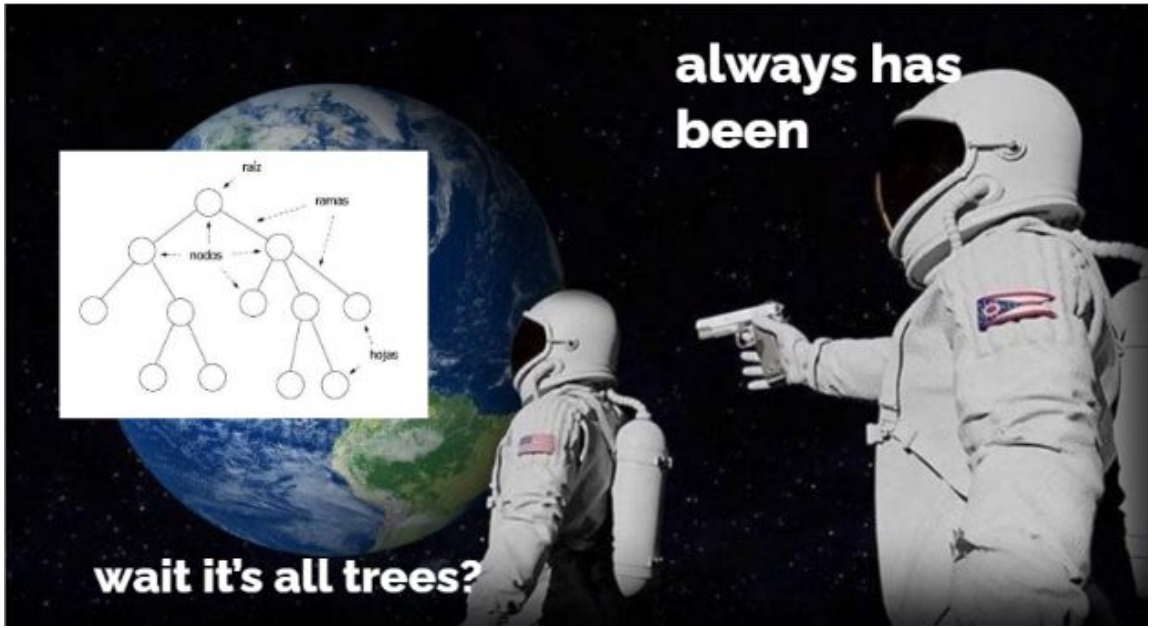
Ayudantia ABB y AVL
08-09-2023

Menti: 31730287 (K200)
38641524 (K204)



Qué es un árbol

- EDD con nodos
- Un nodo contiene un **par de llave-valor**
- Cada nodo tiene asociados sus hijos **mediante punteros**, que pueden ser **cero o más**
- Cada nodo tiene **a lo más 1 padre**, en caso de no tener es la **raíz**
- **No hay ciclos**
- Se arman con **normas pre-establecidas** (siguen ciertas reglas que afectan las operaciones)

Árbol binario de búsqueda (ABB)

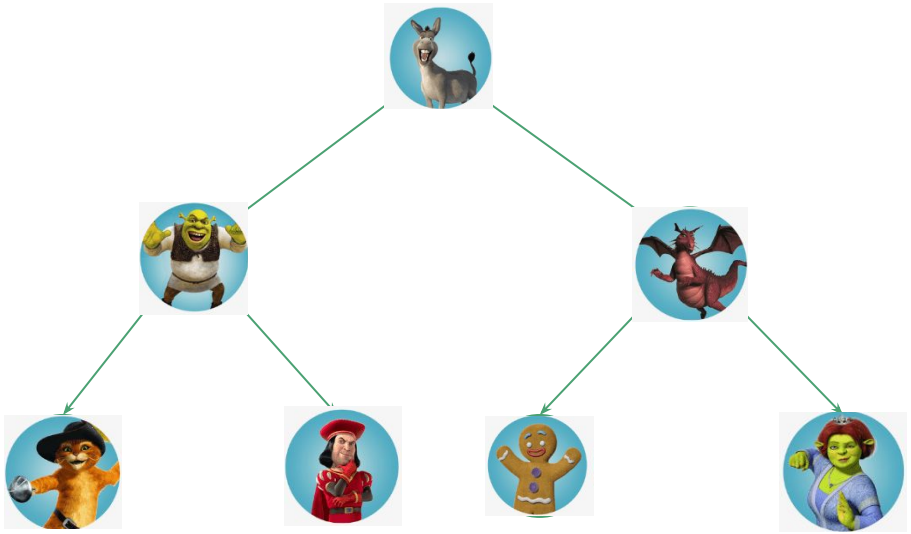
- En inglés **BST**
- Cada nodo tiene asociados **dos** ABB , **mediante punteros**
- Sea x el nodo, entonces:
  - x.key = llave
  - x.value = valor
  - x.p es el padre del nodo, como máximo puede tener 1. En caso de no tener padre es la raíz
  - **tiene a lo más dos hijos**
  - x.left es el nodo izquierdo el cual cumple que **x.left<x**
  - x.rigth es el nodo derecho el cual cumple que **x.rigth>x**
  - en caso de **x.rigth ==null y x.left == null, entonces x es hoja**

¿Como se ve un nodo en C?

```
struct node {
    int key;
    struct node *left, *right;
};
```

Árbol binario de búsqueda (ABB)

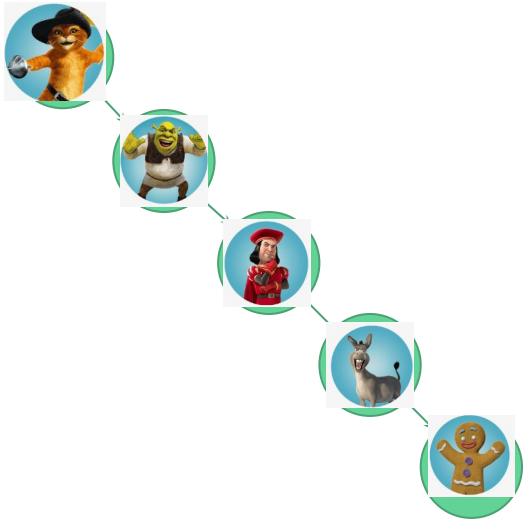
- Altura mínima de la rama más larga es O(log(n)) -> MEJOR CASO



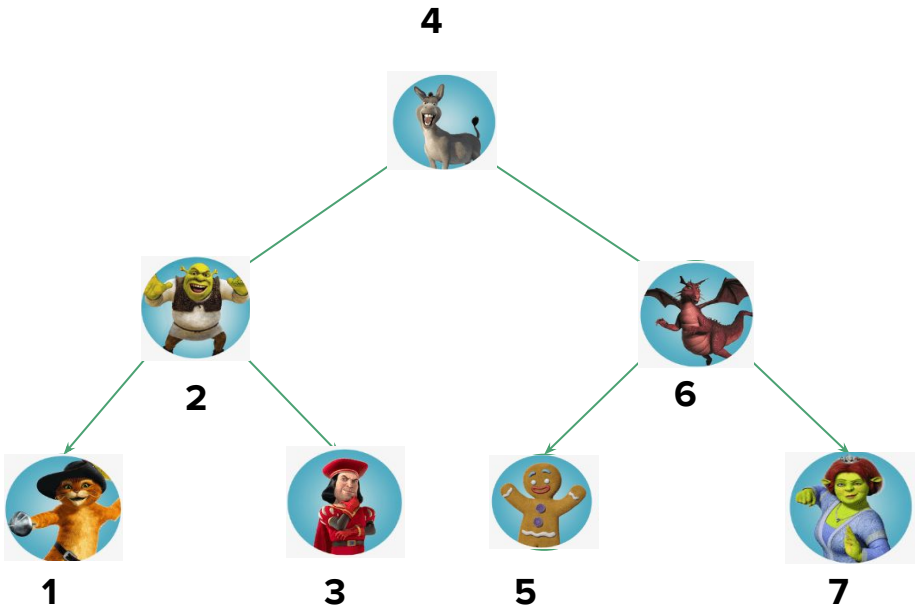
- Cuando está perfectamente balanceado.
- Dado que la diferencia de altura entre el subárbol izquierdo y el subárbol derecho de cada nodo es como máximo 1. con n el número de nodos.

Árbol binario de búsqueda (ABB)

- Altura máxima de O(n), cuando no está balanceado



¿Cómo recorrer árboles?




```
void printfuntion (Node* node)
{
        if (node == NULL)
                        return;
        printfuntion(node->left);
        printf('%d', node->value);
        printfuntion(node->right);
}
```

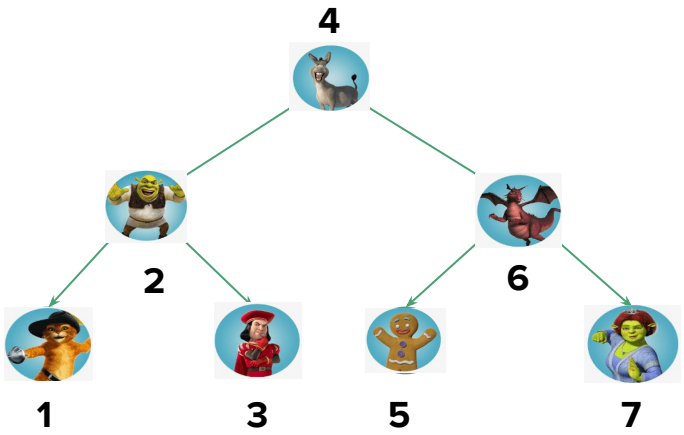
```
void printfuntion (Node* node)
{
        if (node == NULL)
                        return;
        printf('%d', node->value);
        printfuntion(node->left);
        printfuntion(node->right);

}
```

```
void printfuntion (Node* node)
{
        if (node == NULL)
                        return;
        printfuntion(node->left);
        printfuntion(node->right);
        printf('%d', node->value);

}
```

Cómo recorrer árboles?

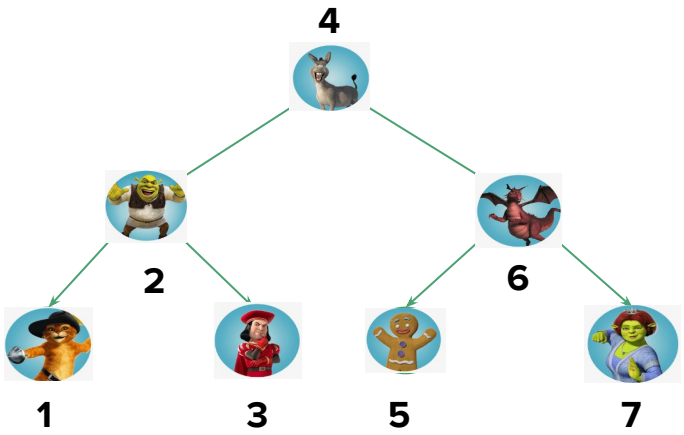


InOrder

```
void printfuntion (Node* node)
{
        if (node == NULL)
                return;
        printfuntion(node->left);
        printf('%d', node->value);
        printfuntion(node->right);
}
```

1234567

Cómo recorrer árboles?



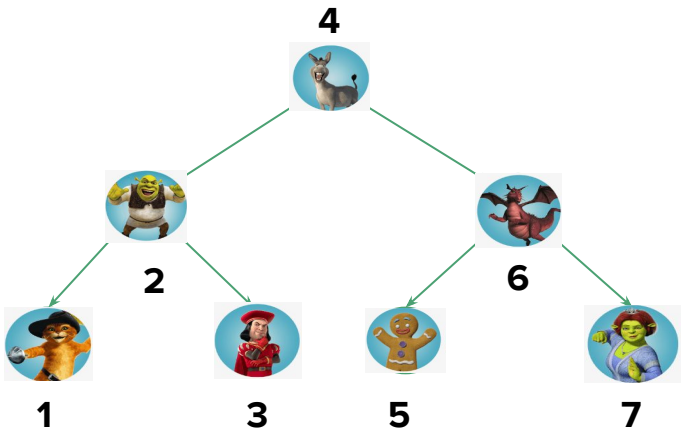
PreOrder

```
void printfuntion (Node* node)
{
        if (node == NULL)
                return;
        printf('%d', node->value);
        printfuntion(node->left);
        printfuntion(node->right);

}
```

4213657

Cómo recorrer árboles?



PostOrder

```
void printfuntion (Node* node)
{
        if (node == NULL)
                return;
        printfuntion(node->left);
        printfuntion(node->right);
        printf('%d', node->value);

}
```

1325764

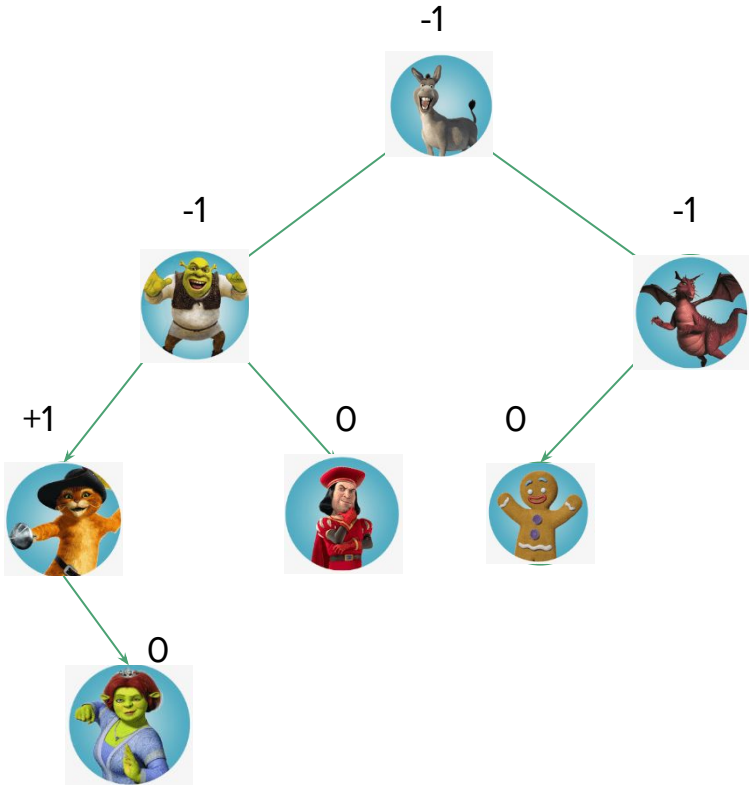
Qué es balance

- Sean n nodos, que la altura sea O(log(n))
- Sea fácil de mantener
- La definición debe ser de carácter recursivo
- y para que sea fácil de mantener se realizan luego de cada operaciones, para que el proceso de balanceo sea <O(log(n))

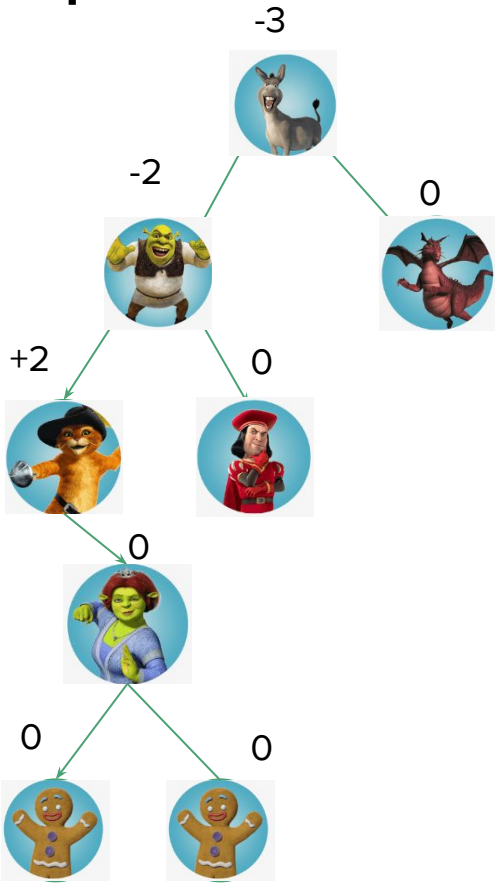
AVL

- Es un tipo de ABB
- Se encuentra balanceado, entonces altura O(log(n))
- La diferencia máxima de sus hijos es de 1
- Cada hijo es AVL (definición recursiva)

¿Que significa que difiera a lo más de 1?

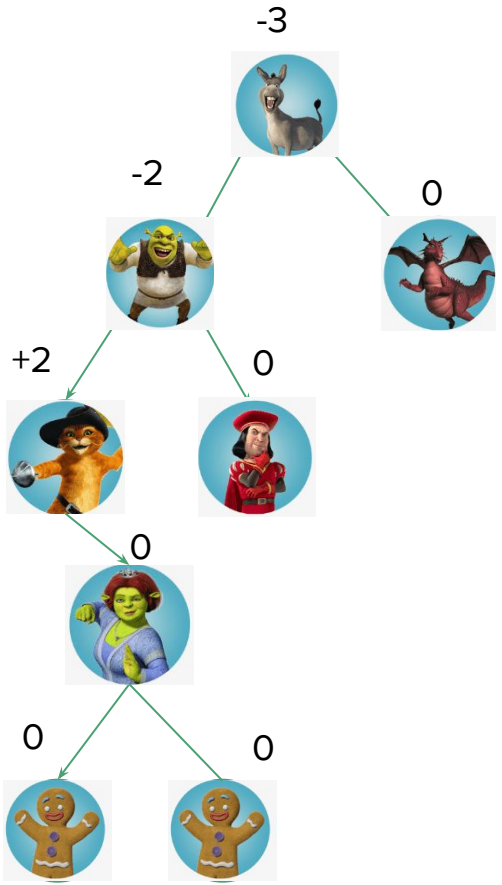


¿Que significa que difiera a lo más de 1?

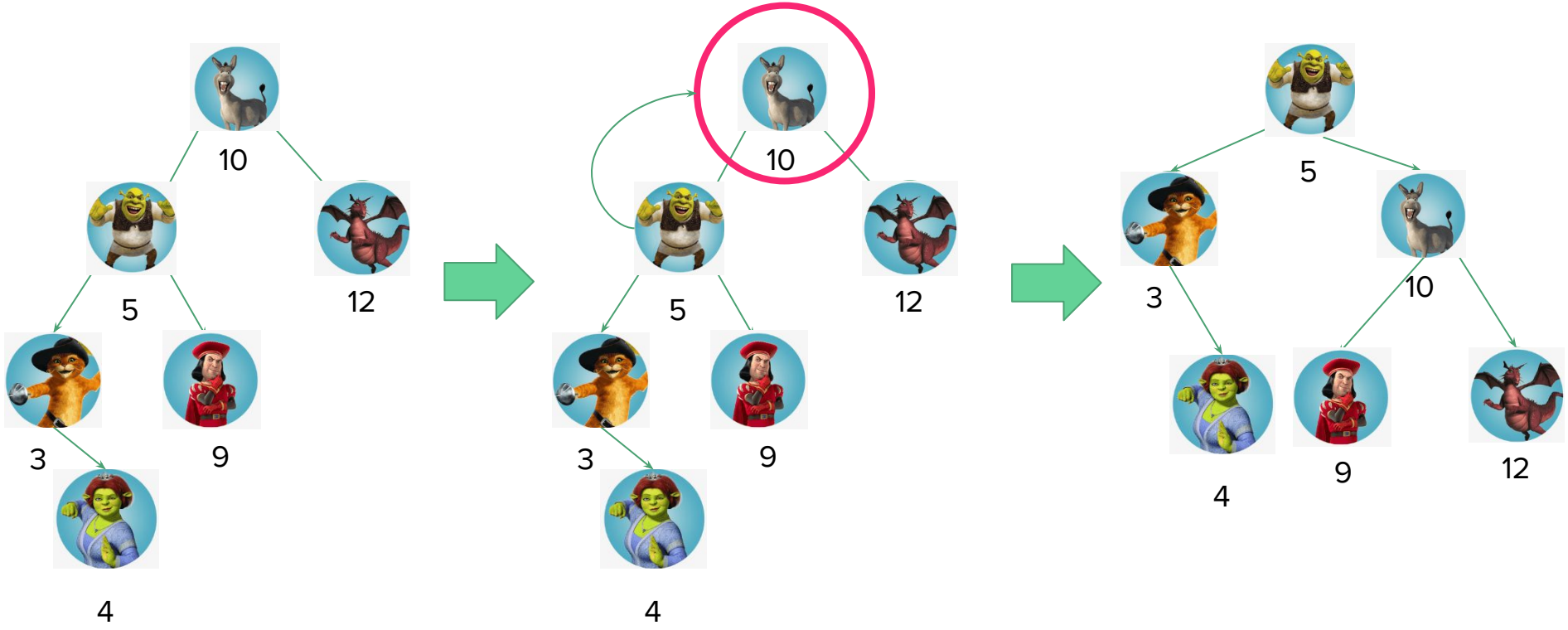


¿Cómo lo arreglamos?

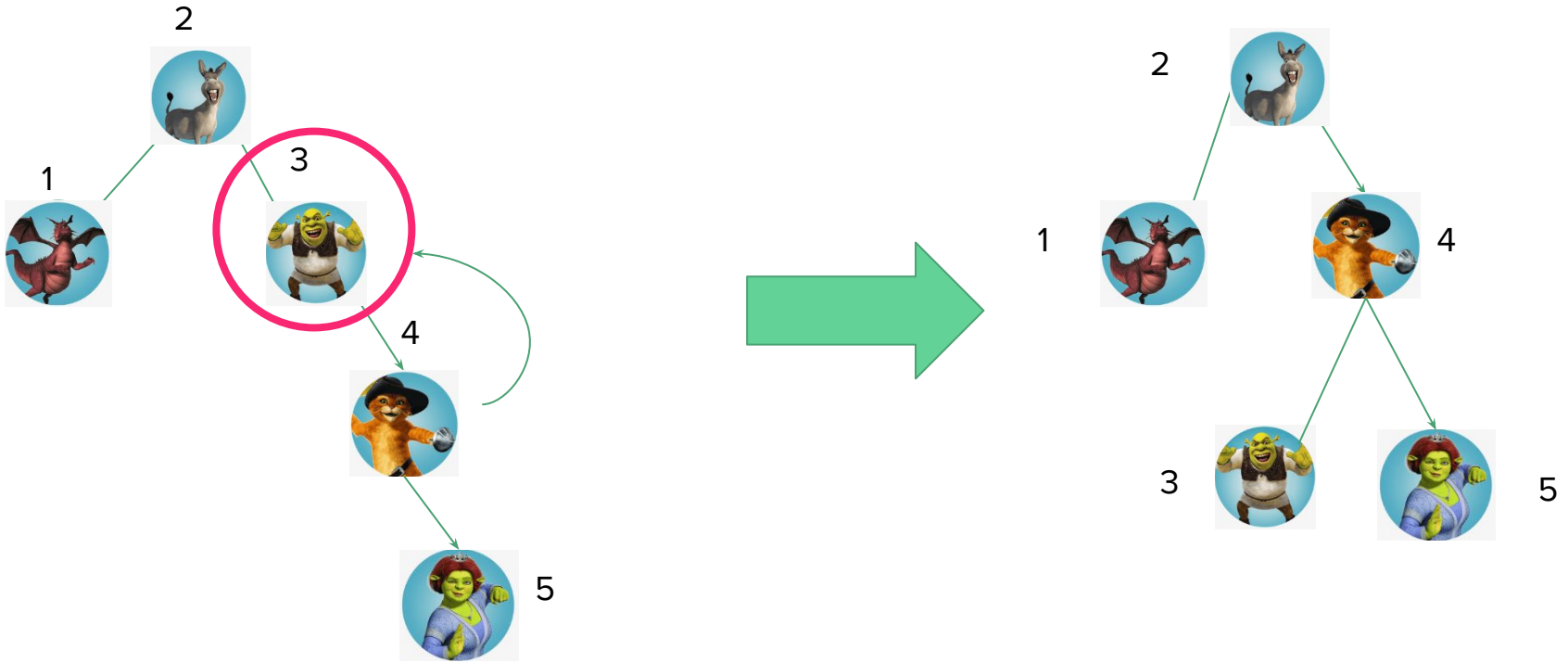
1. Identificar primer nodo desbalanceado de forma ascendente
2. Identificar rotación
3. Rotar
4. Evaluar balance



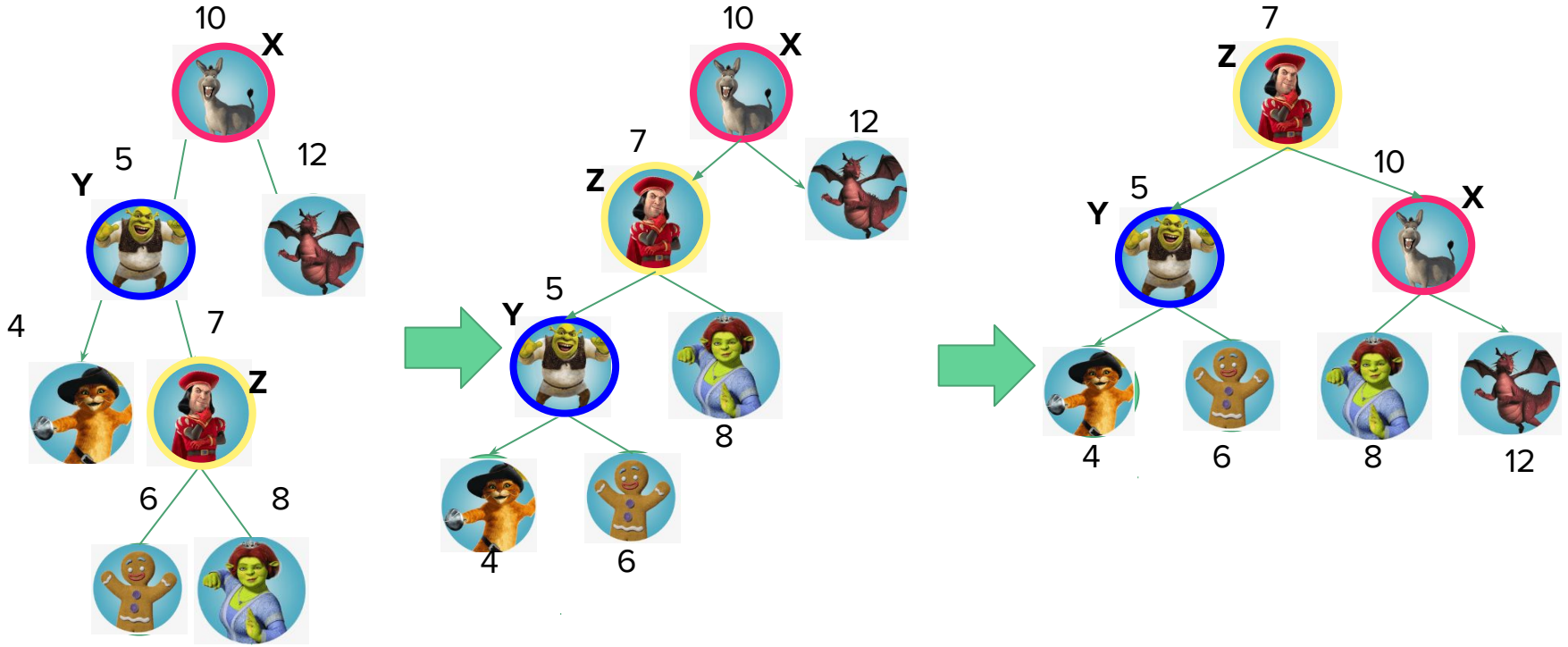
Rotación Simple



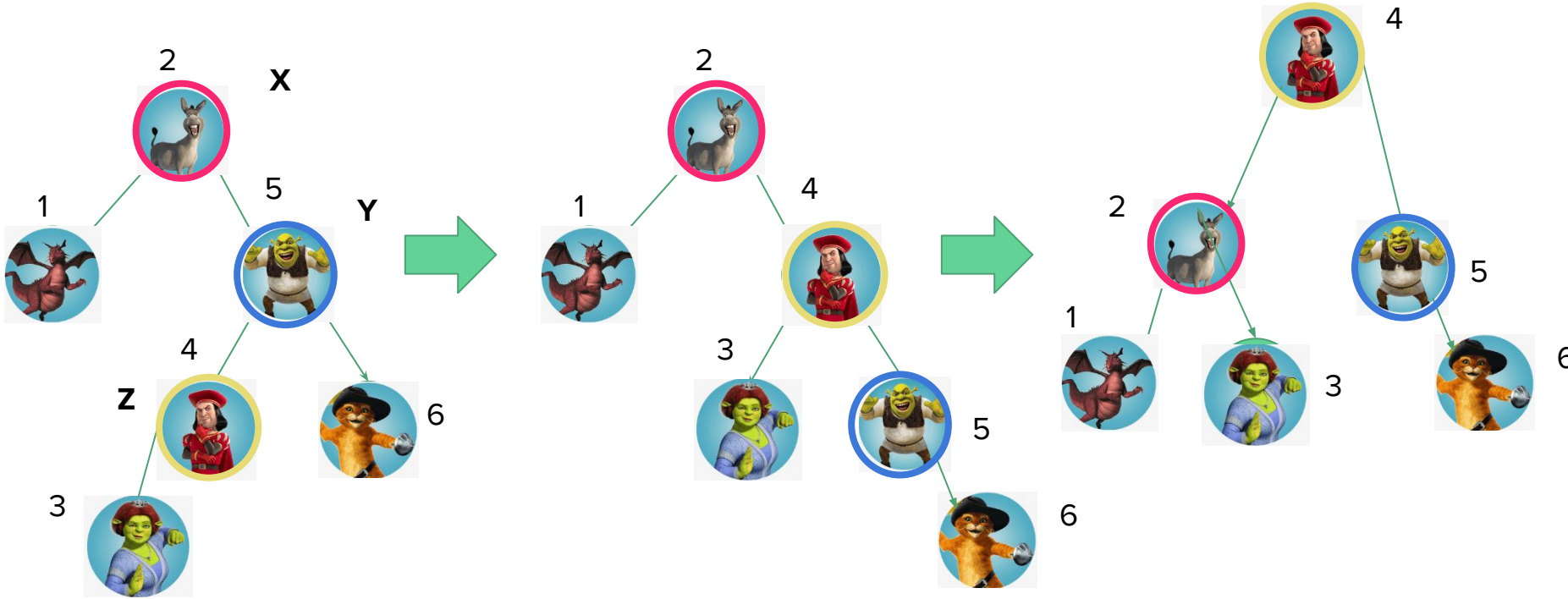
Rotación Simple



Rotación: Doble



Rotación Doble



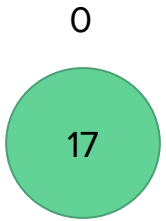
Ejercicio rotación AVL

Considera que tienes un AVL vacío y una lista desordenada:

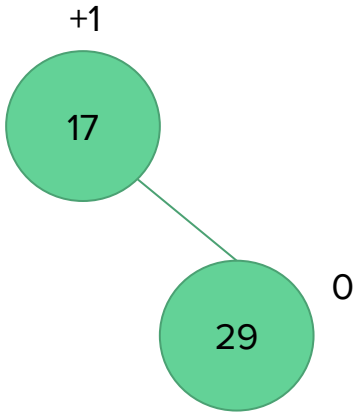
17, 29, 53, 61, 73, 37

Inserta los nodos realizando las rotaciones diferentes

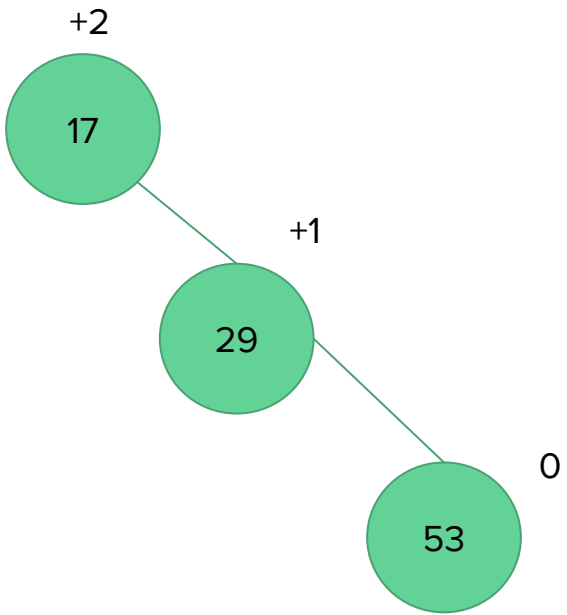
Insertar 17



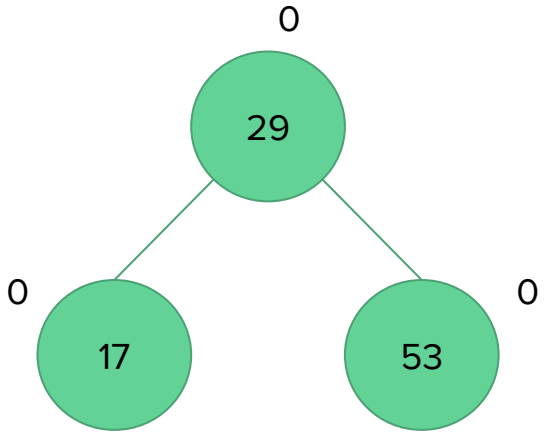
Insertar 29



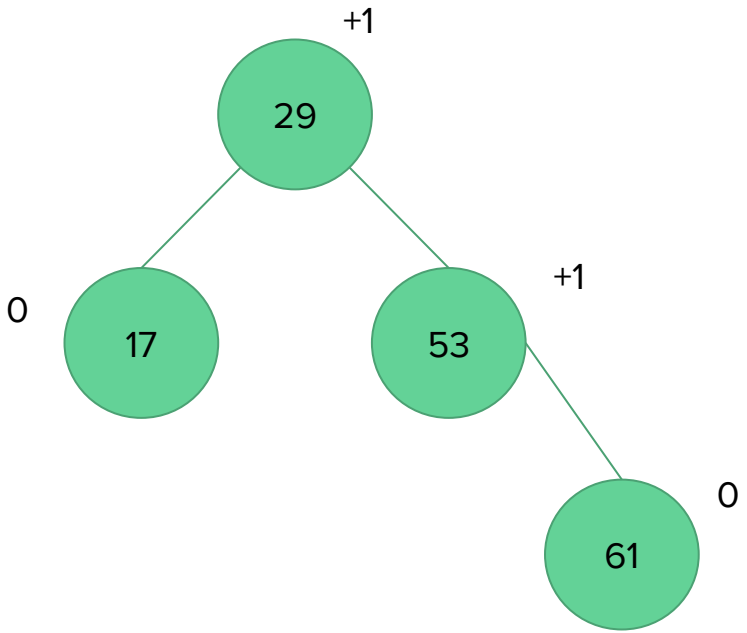
Insertar 53



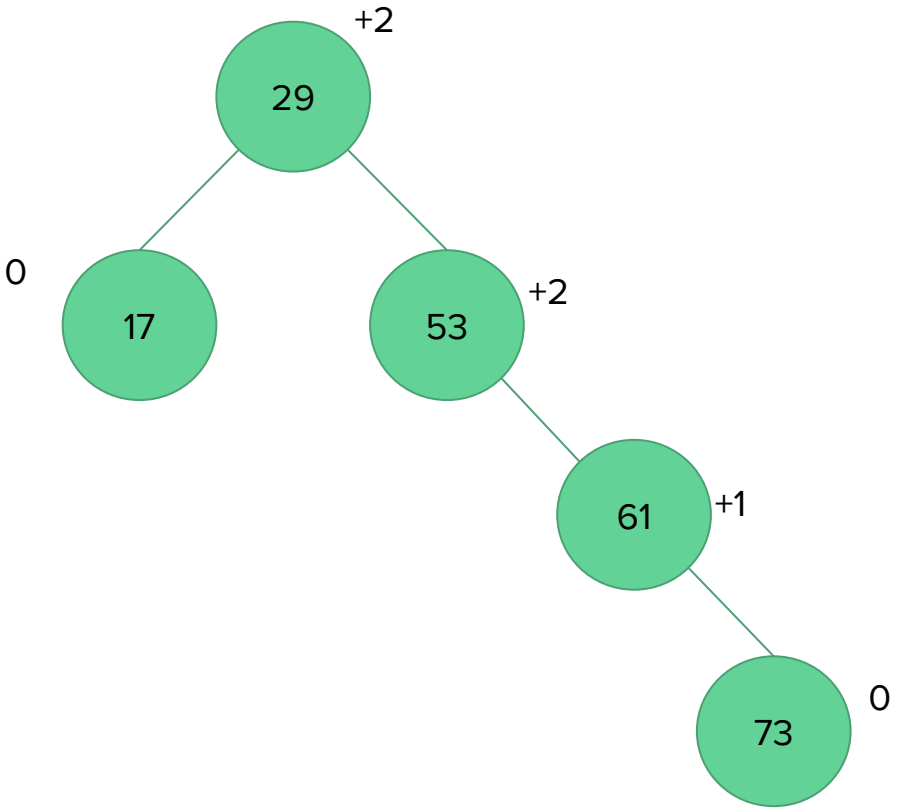
Rotación Simple



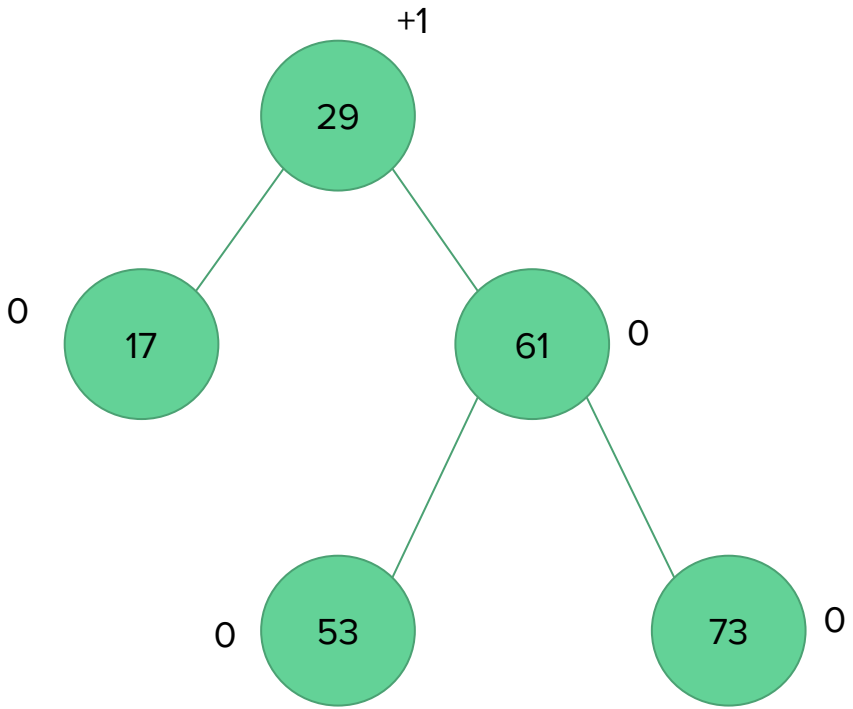
Insertar 61



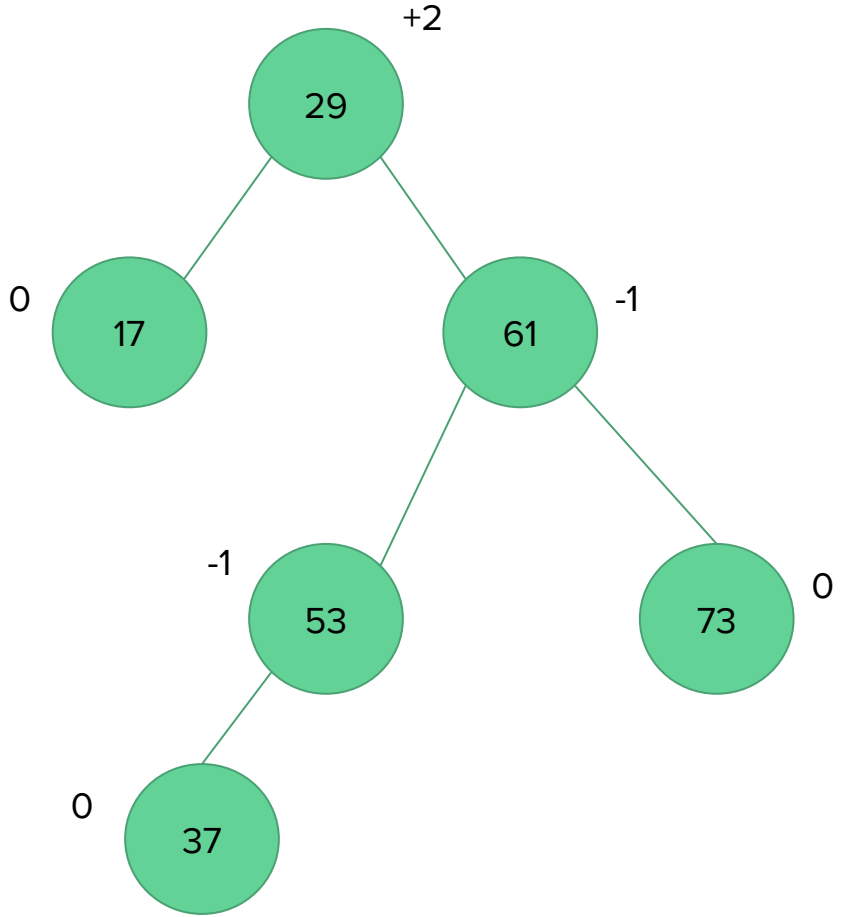
Insertar 73



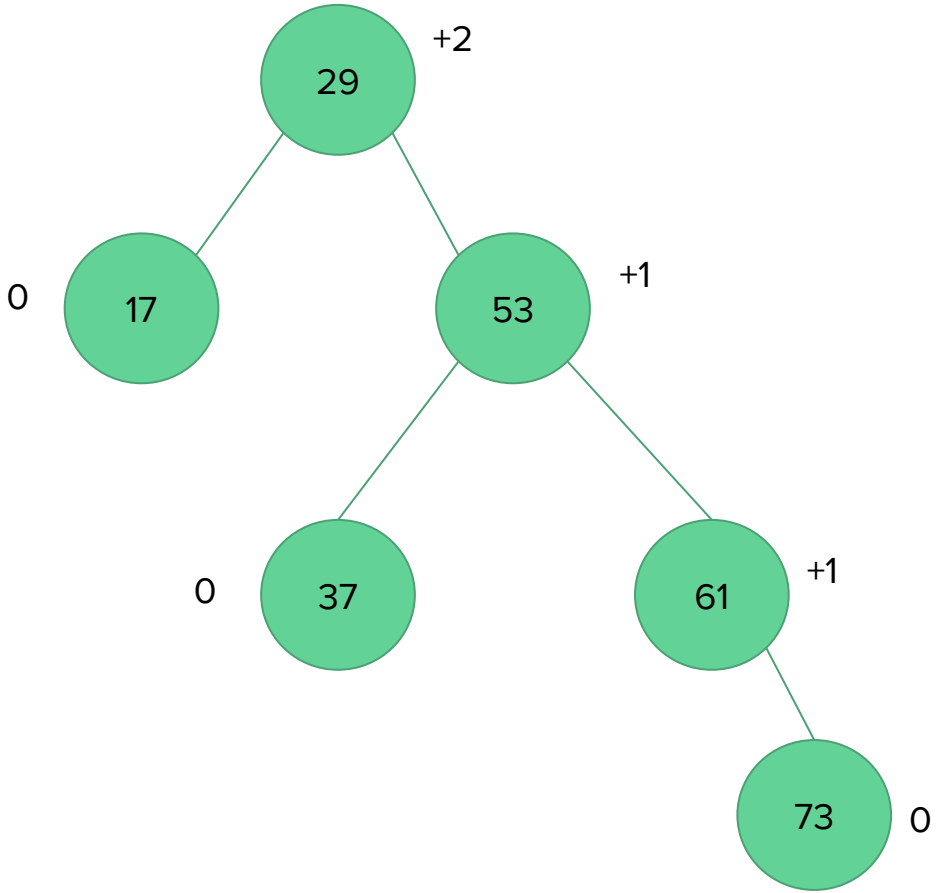
Rotación Simple



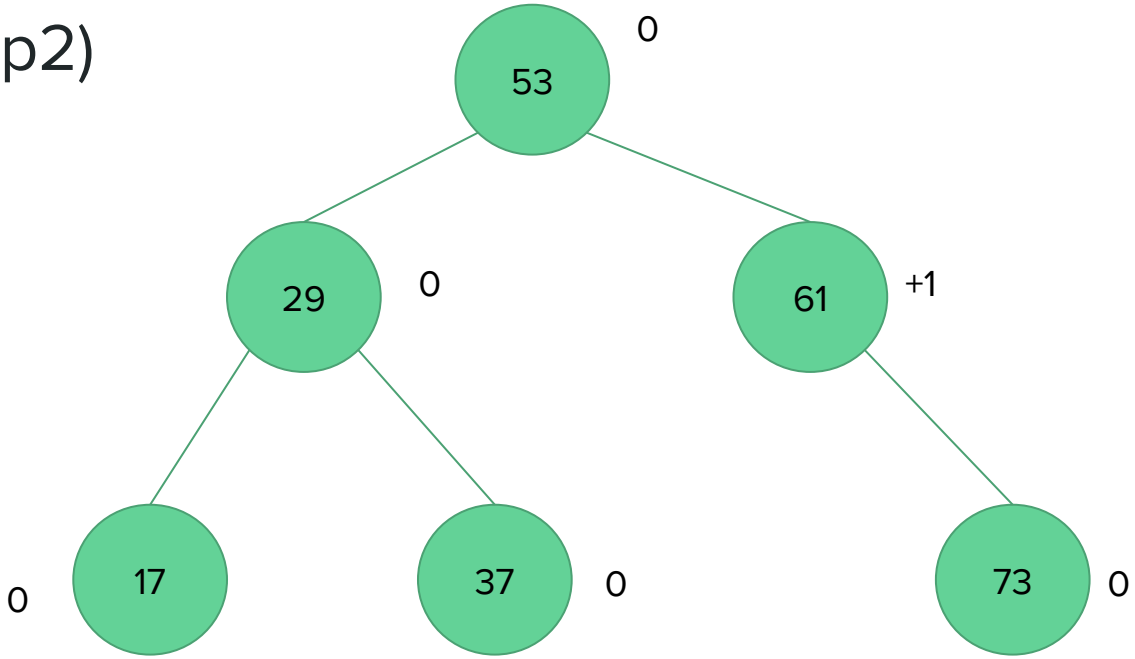
Insertar 37



Rotación doble (p1)

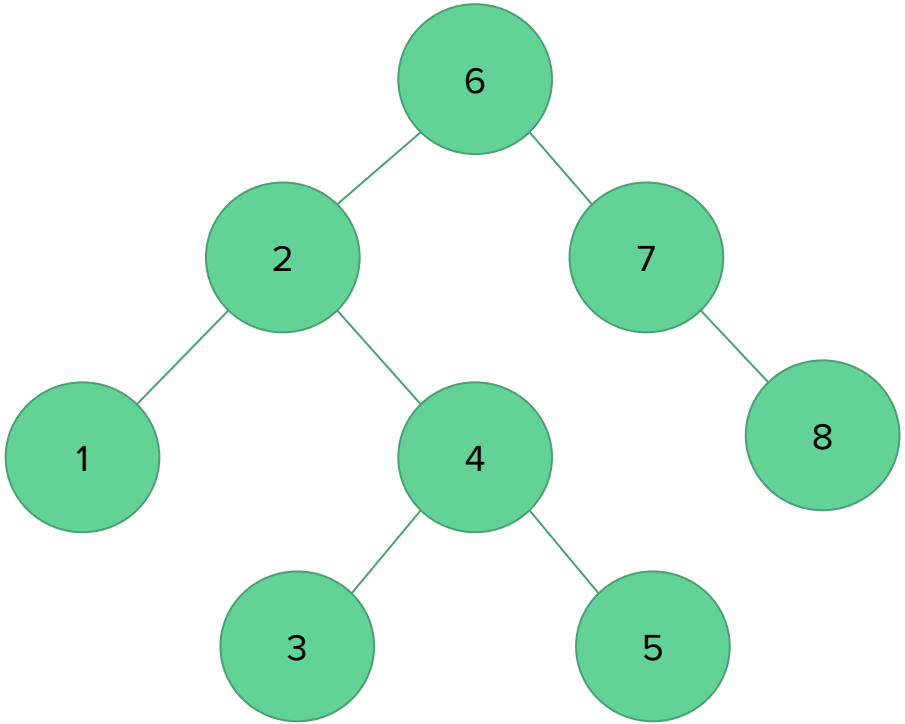


Rotación doble (p2)

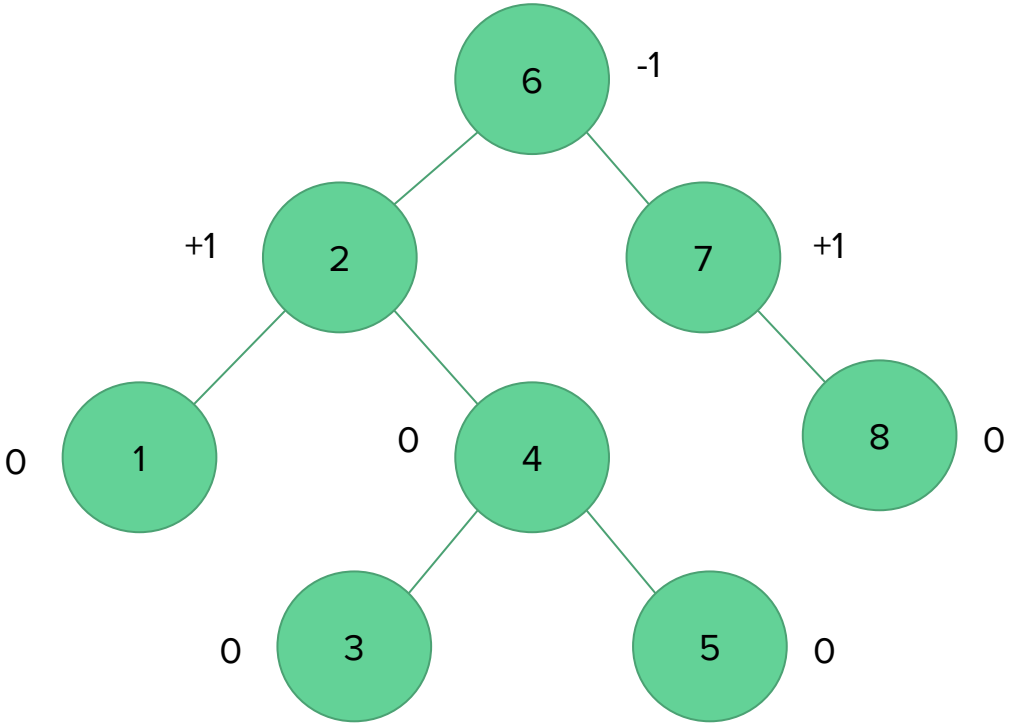


Ejercicio eliminación AVL

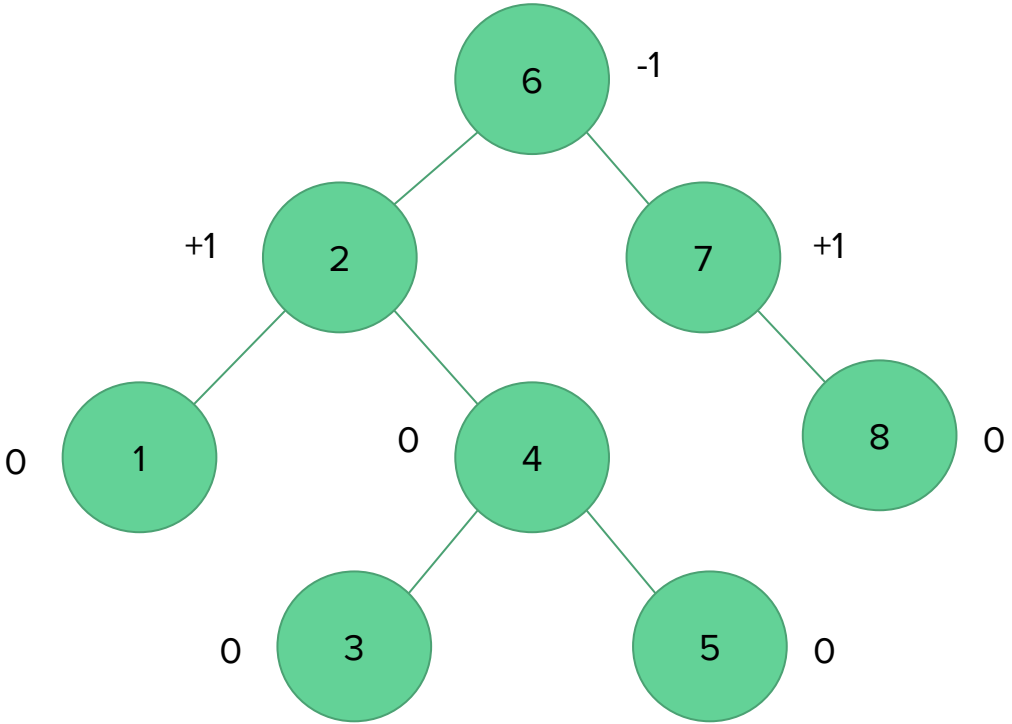
Considera que tienes el siguiente árbol AVL:



Considera que tienes el siguiente árbol AVL:

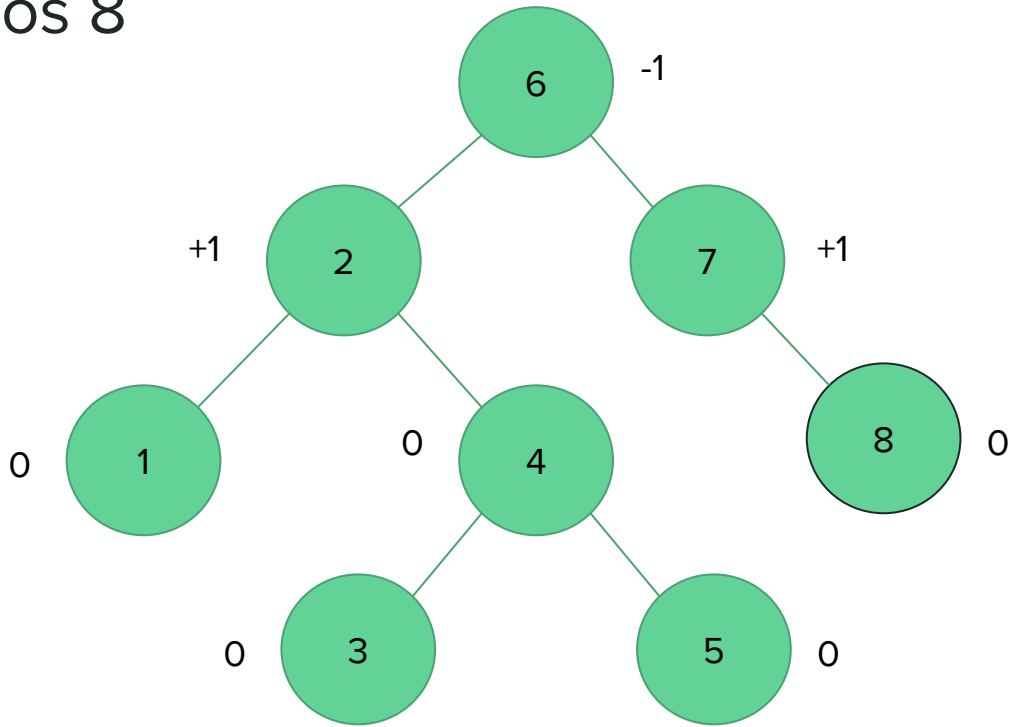


Considera que tienes el siguiente árbol AVL:

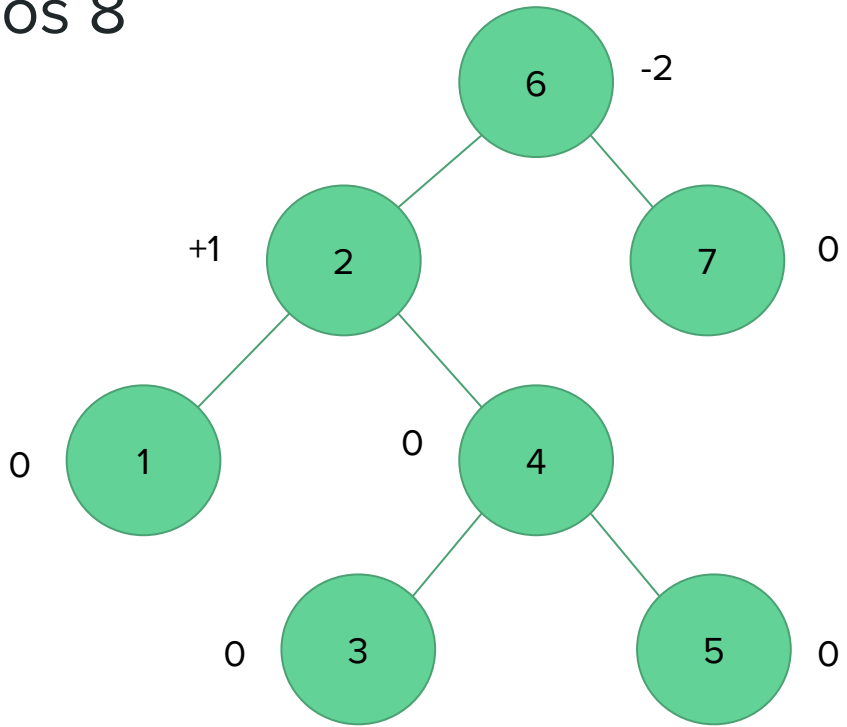


Ejercicio: eliminar el nodo 8

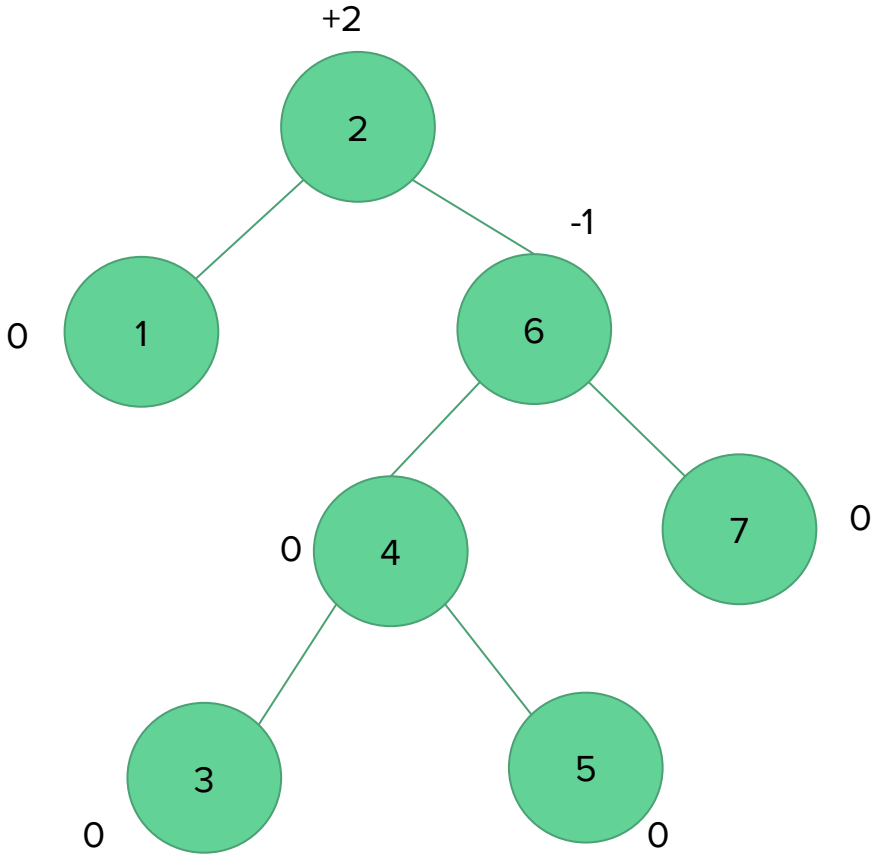
Eliminamos 8



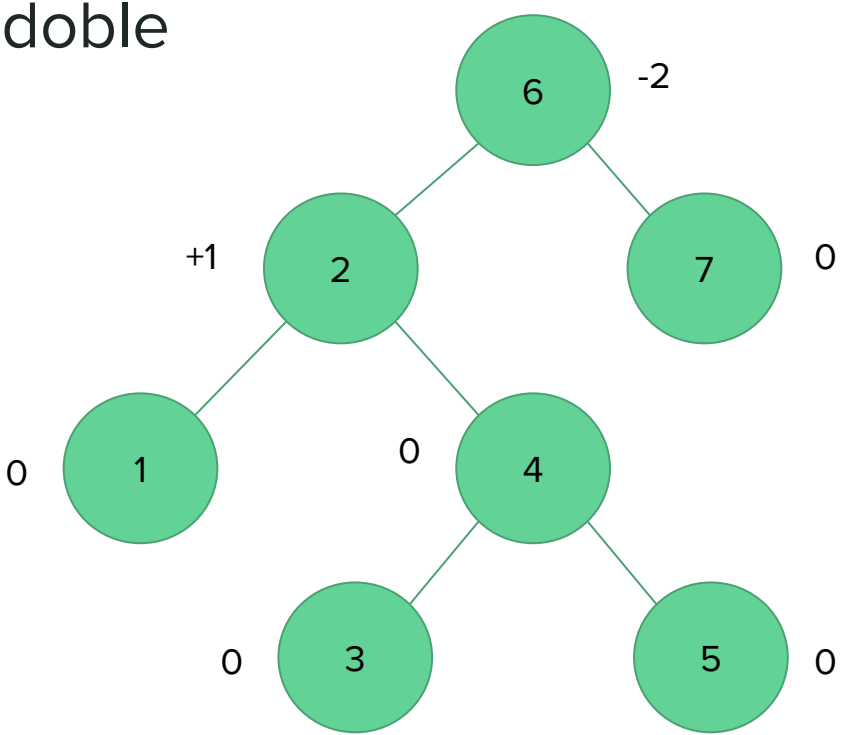
Eliminamos 8



¿Podemos usar rotación simple?



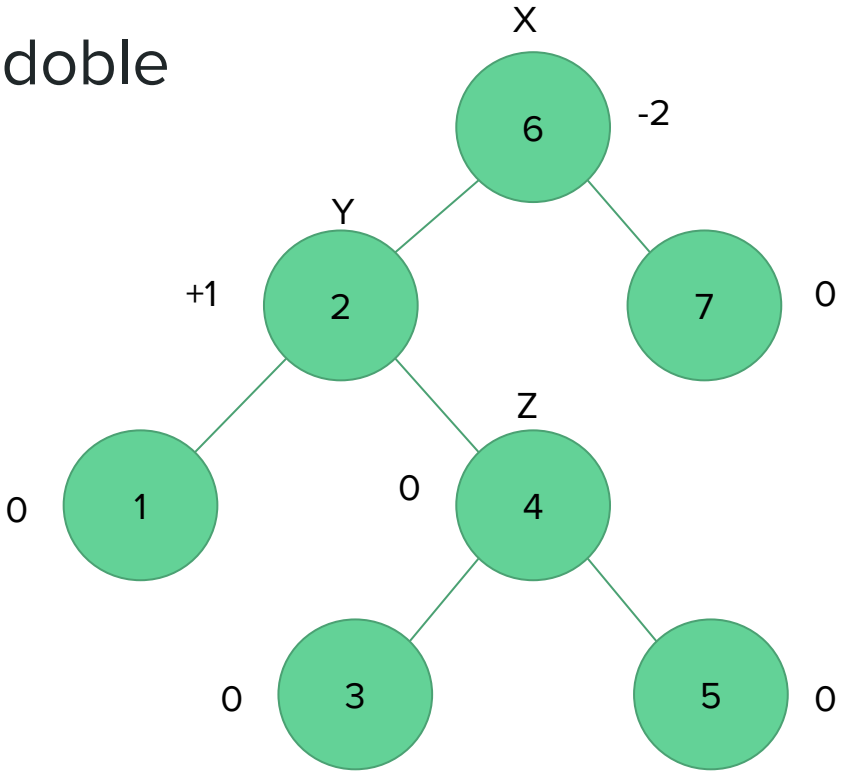
Rotación doble



Recordar:

- **X** es el primer nodo desbalanceado al subir
- **Y** es el hijo de X en el camino de desbalance
- **Z** es el hijo de Y en el camino de desbalance

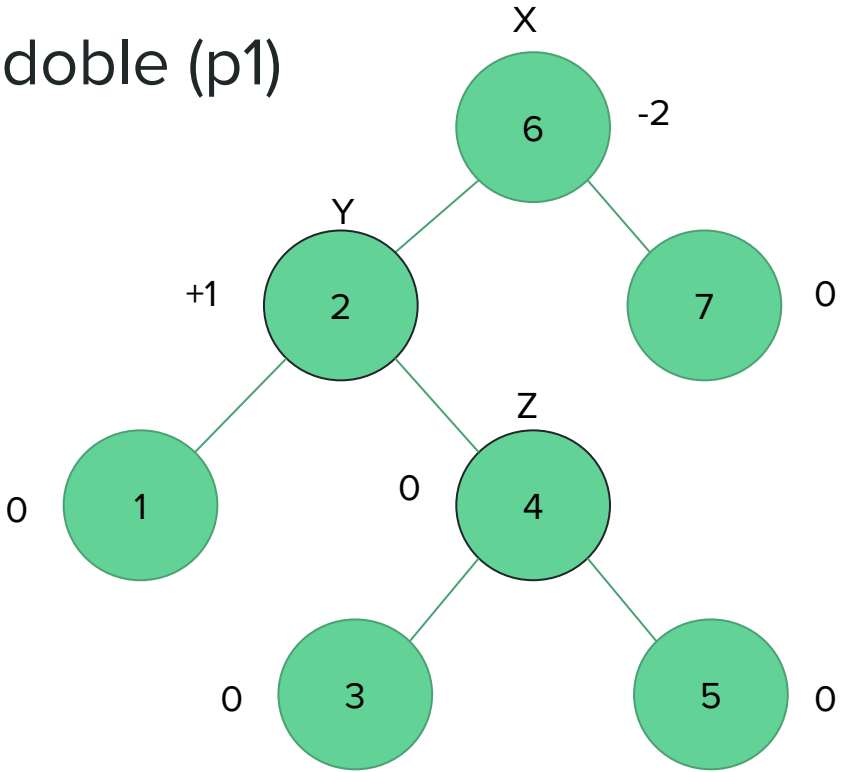
Rotación doble



Recordar:

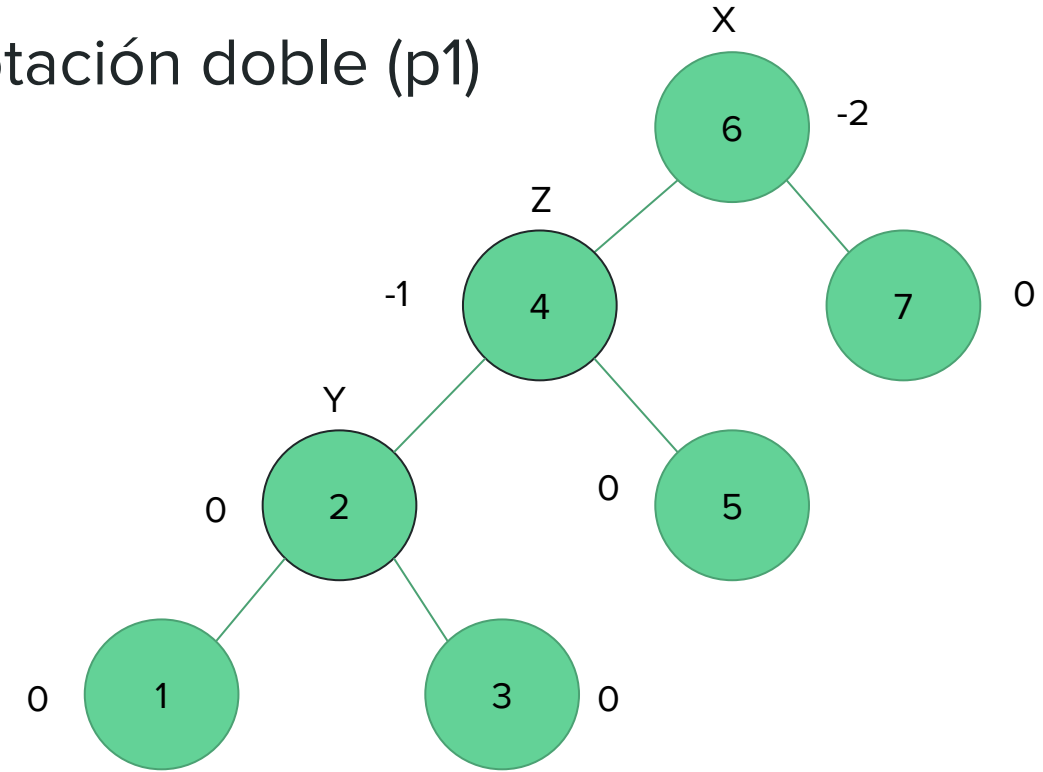
- **X** es el primer nodo desbalanceado al subir
- **Y** es el hijo de X en el camino de desbalance
- **Z** es el hijo de Y en el camino de desbalance

Rotación doble (p1)



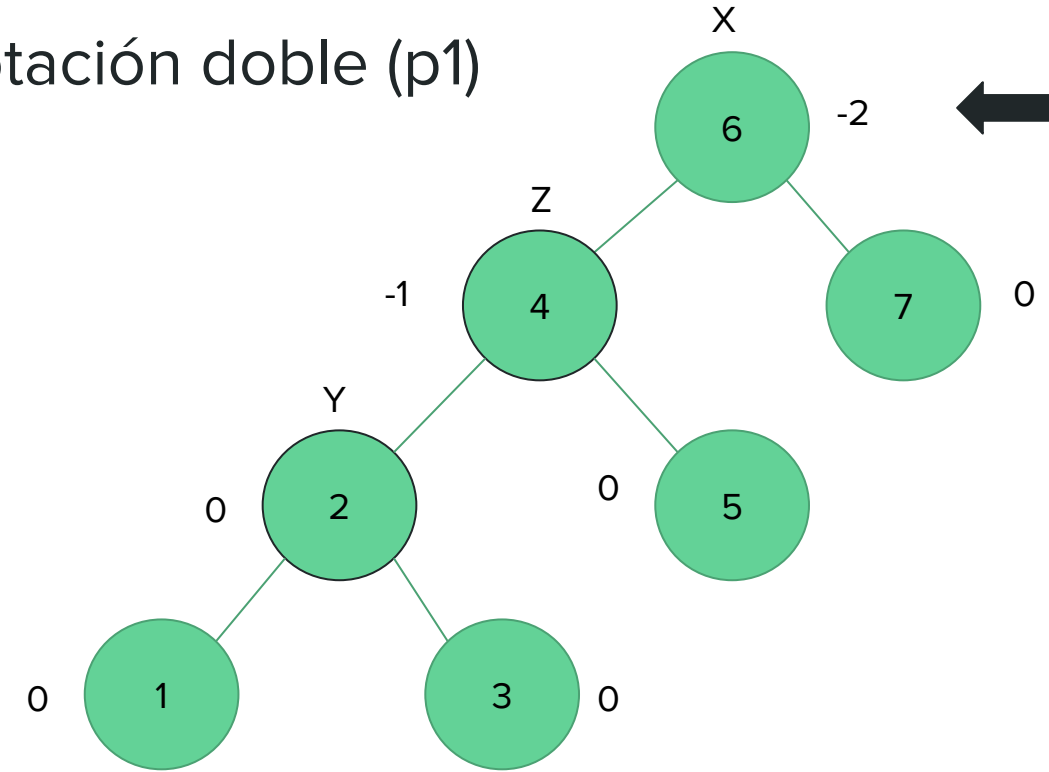
Primero rotamos a la izquierda en torno a Y-Z

Rotación doble (p1)



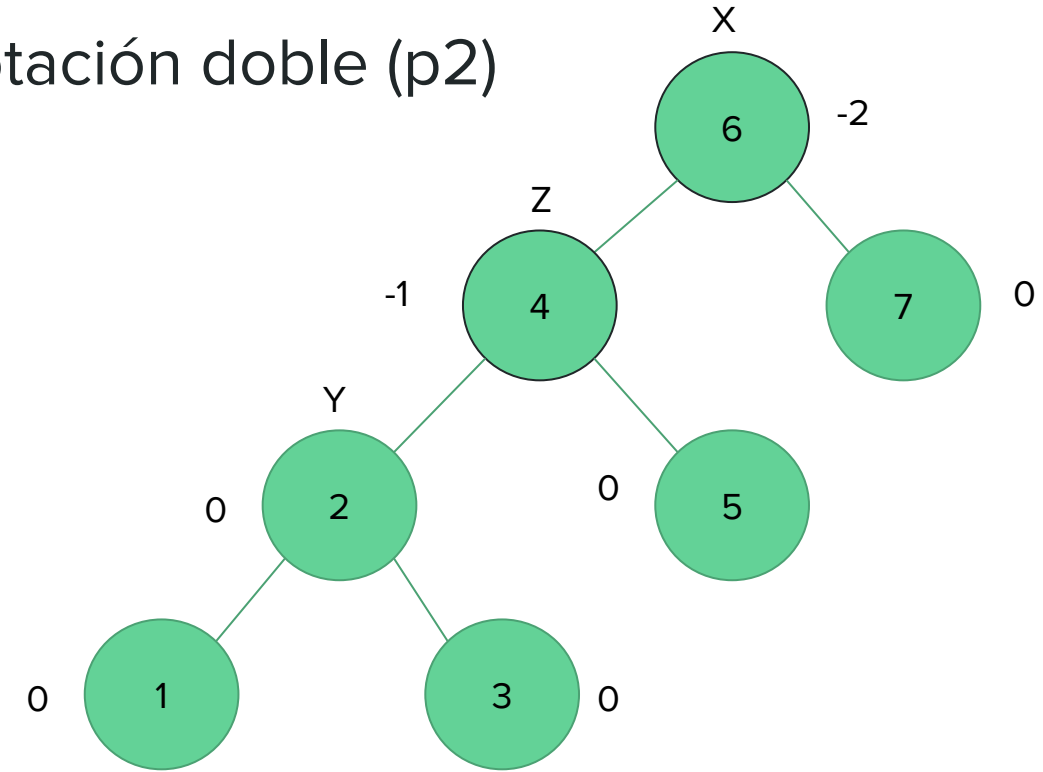
Primero rotamos a la izquierda en torno a Y-Z

Rotación doble (p1)



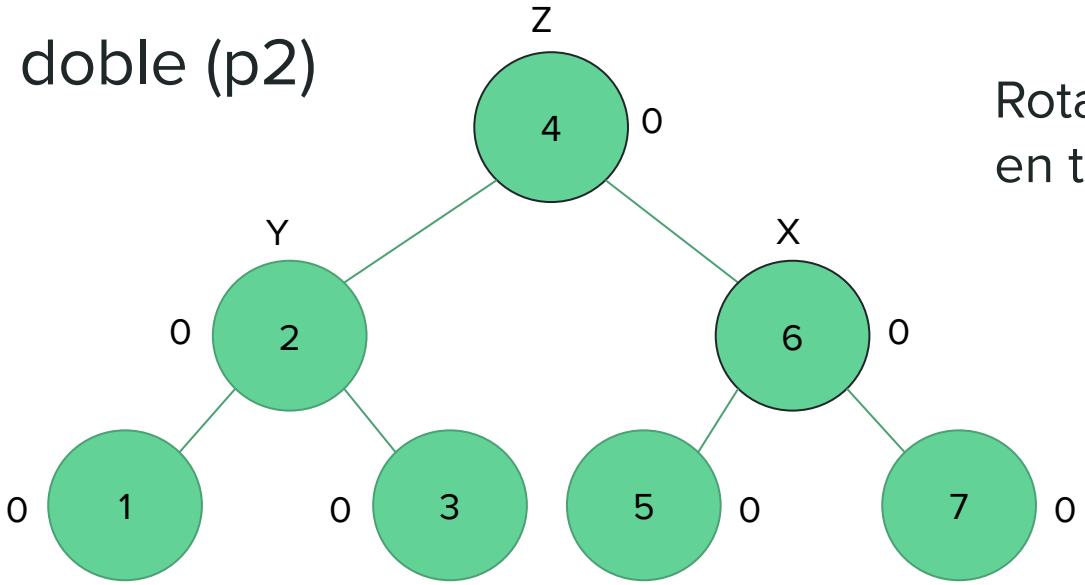
Sigue desbalanceado!

Rotación doble (p2)



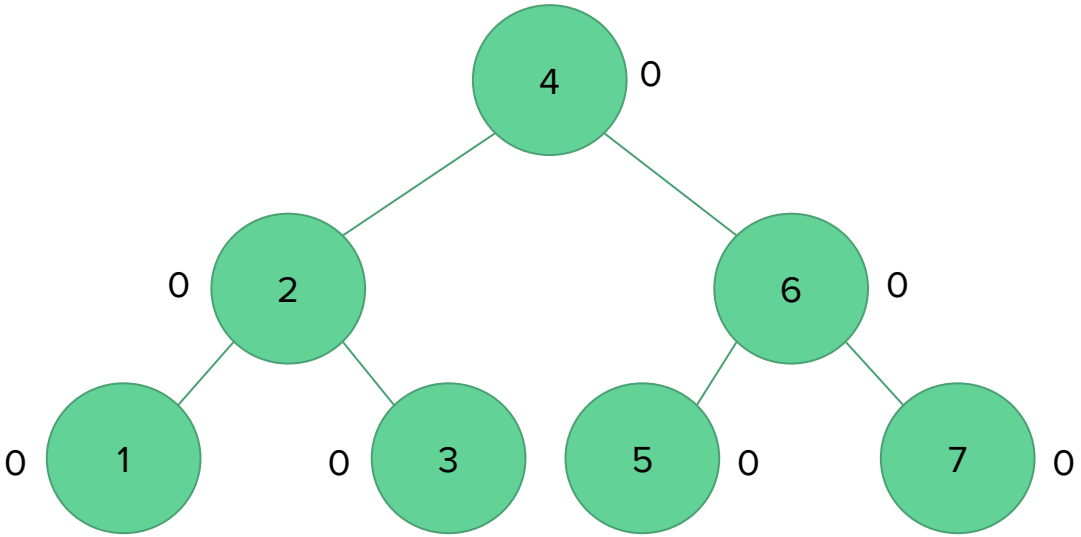
Rotamos a la derecha en torno a X-Z

Rotación doble (p2)



Rotamos a la derecha en torno a X-Z

Perfectamente balanceado!



¡Muchas gracias!