Ayudantía ABB - AVL y 2-3

MATERIAL DE APOYO

1. Cheatsheet C (notion resumen)

2. Ejercicios de práctica C
   (útil - quizá - para T1 ver el de ABB)

3. Cápsulas de semestres pasados

Dónde encuentro esto?

Links en ReadMe carpeta “Ayudantías” del repo

Árbol binario de búsqueda (ABB)

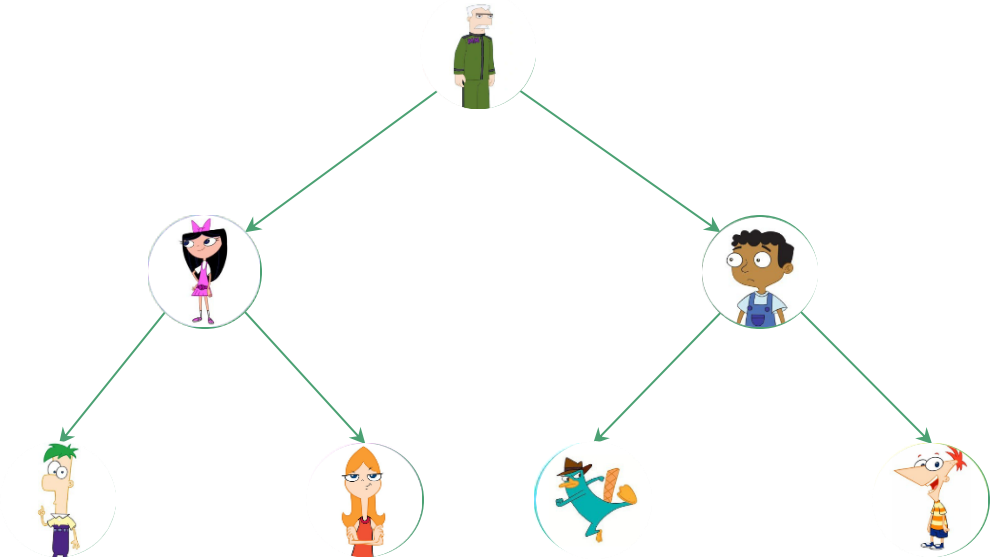
- En inglés **BST**
- Cada nodo tiene asociados **dos ABB , mediante punteros**
- Sea x el nodo, entonces:
  - x.key = llave
  - x.value = valor
  - x.p es el padre del nodo, como máximo puede tener 1. En caso de no tener padre es la raíz
  - **tiene a lo más dos hijos**
  - x.left es el nodo izquierdo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

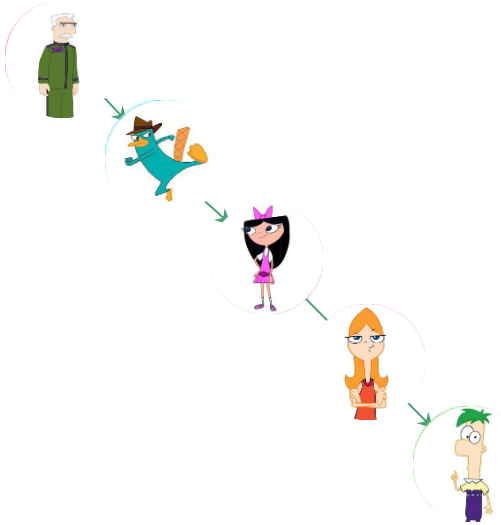
- Altura mínima de la rama más larga es O(log(n)) -> MEJOR CASO



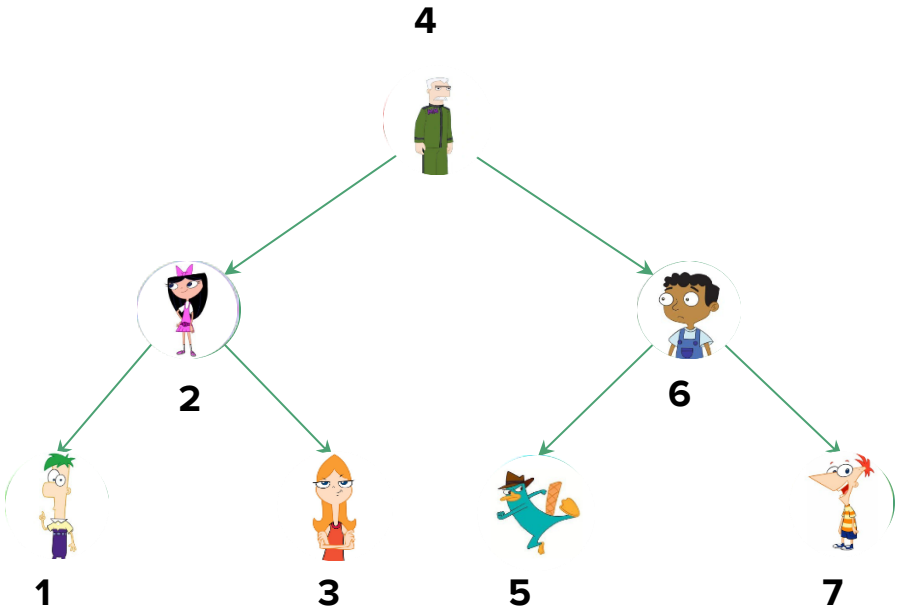
- Cuando está perfectamente balanceado.
- Dado que la diferencia de altura entre el subárbol izquierdo y el subárbol derecho de cada nodo es como máximo 1. con n el número de nodos.

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);
}
```

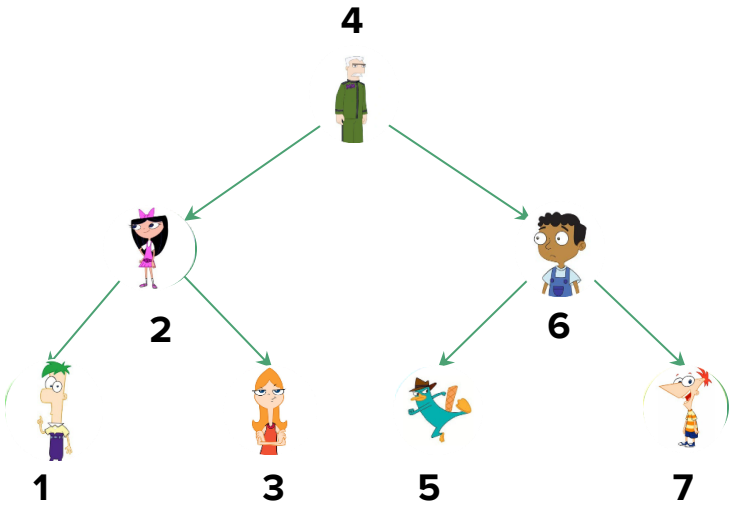
```
void printfuntion (Node* node)
{
        if (node == NULL)

                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?



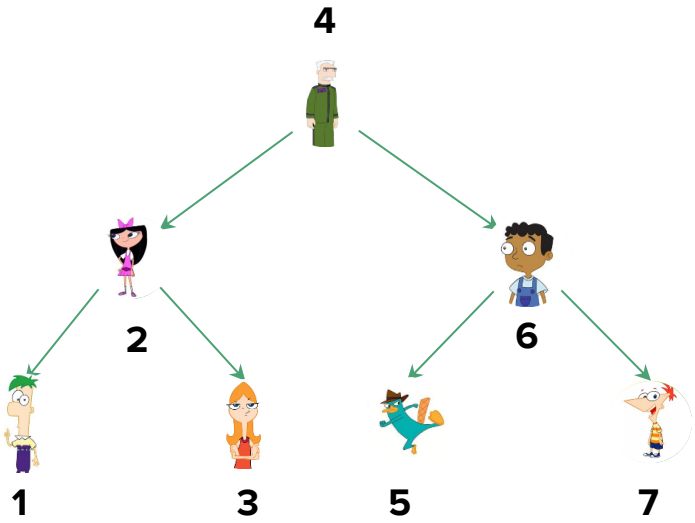
InOrder

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

1234567

Cómo recorrer árboles?

PreOrder

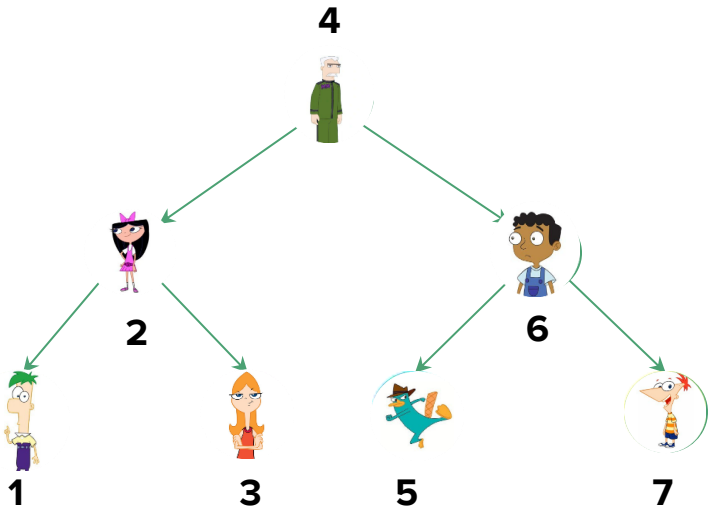



```
void printfuntion (Node* node)
{
        if (node == NULL)
                        return;
        printf('%d', node->value);
                printfuntion(node->left);
                printfuntion(node->right);

}
```

4213657

Cómo recorrer árboles?



PostOrder

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

}
```

1325764

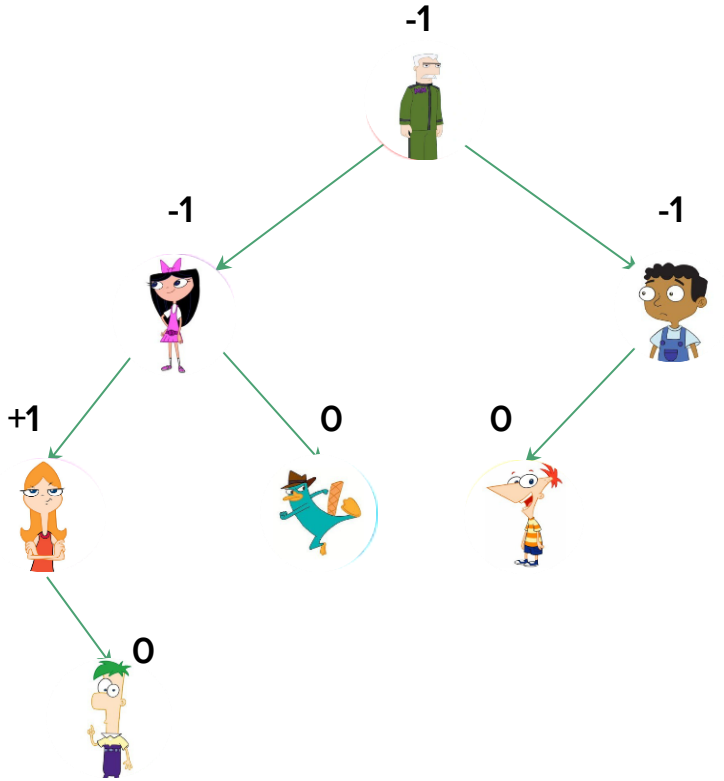
Qué es balance

- 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))

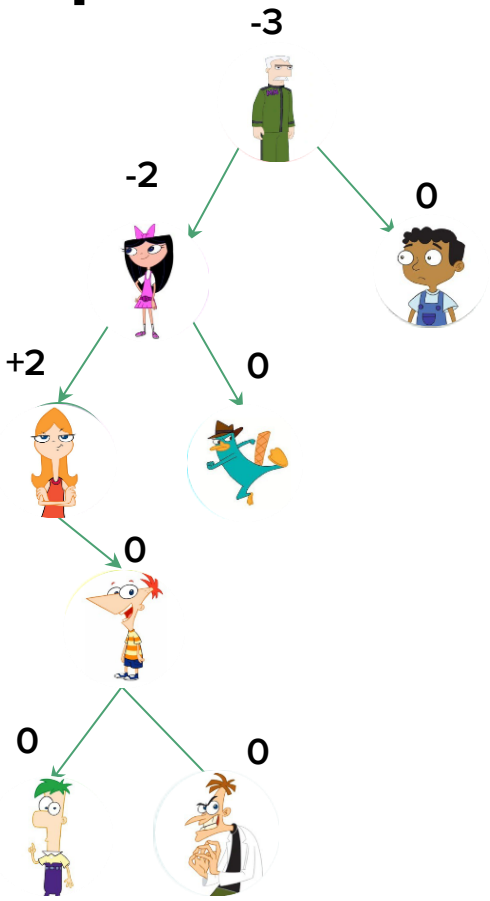
AVL

- 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?

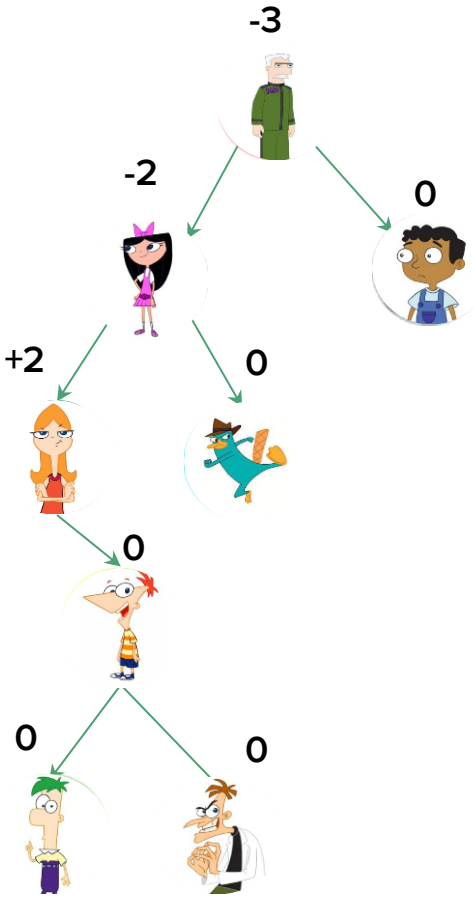


¿Que significa que difiera a lo más de 1?

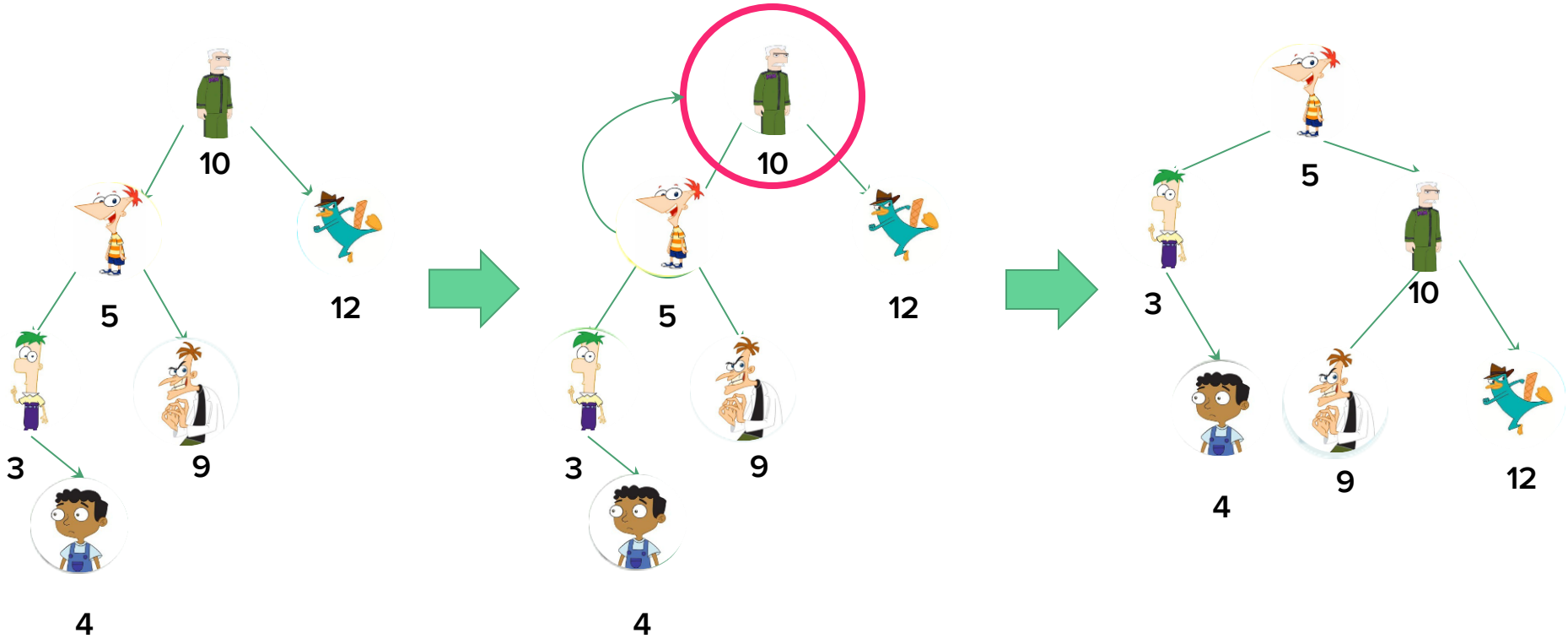


¿Cómo lo arreglamos?

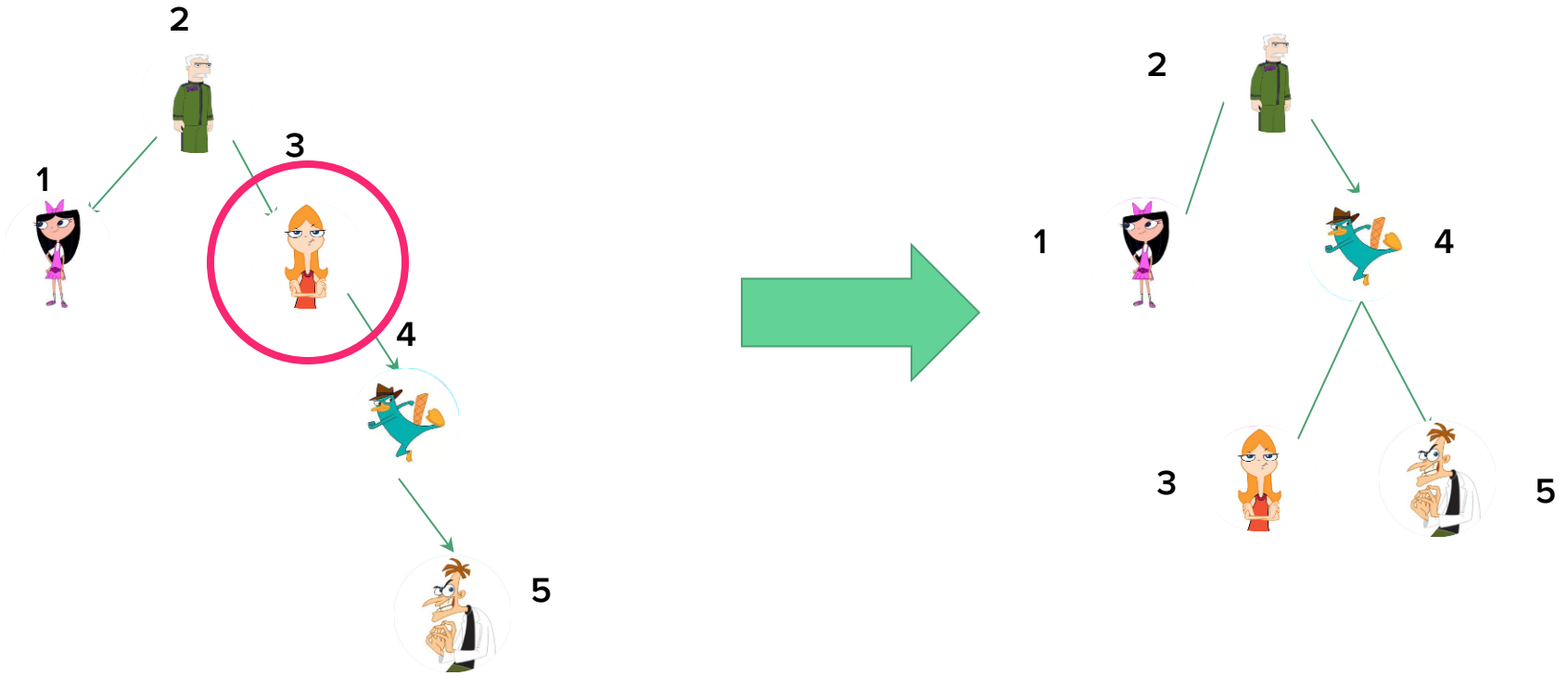
1. Identificar primer nodo desbalanceado de forma ascendente
2. Identificar rotación
3. Rotar
4. Evaluar balance



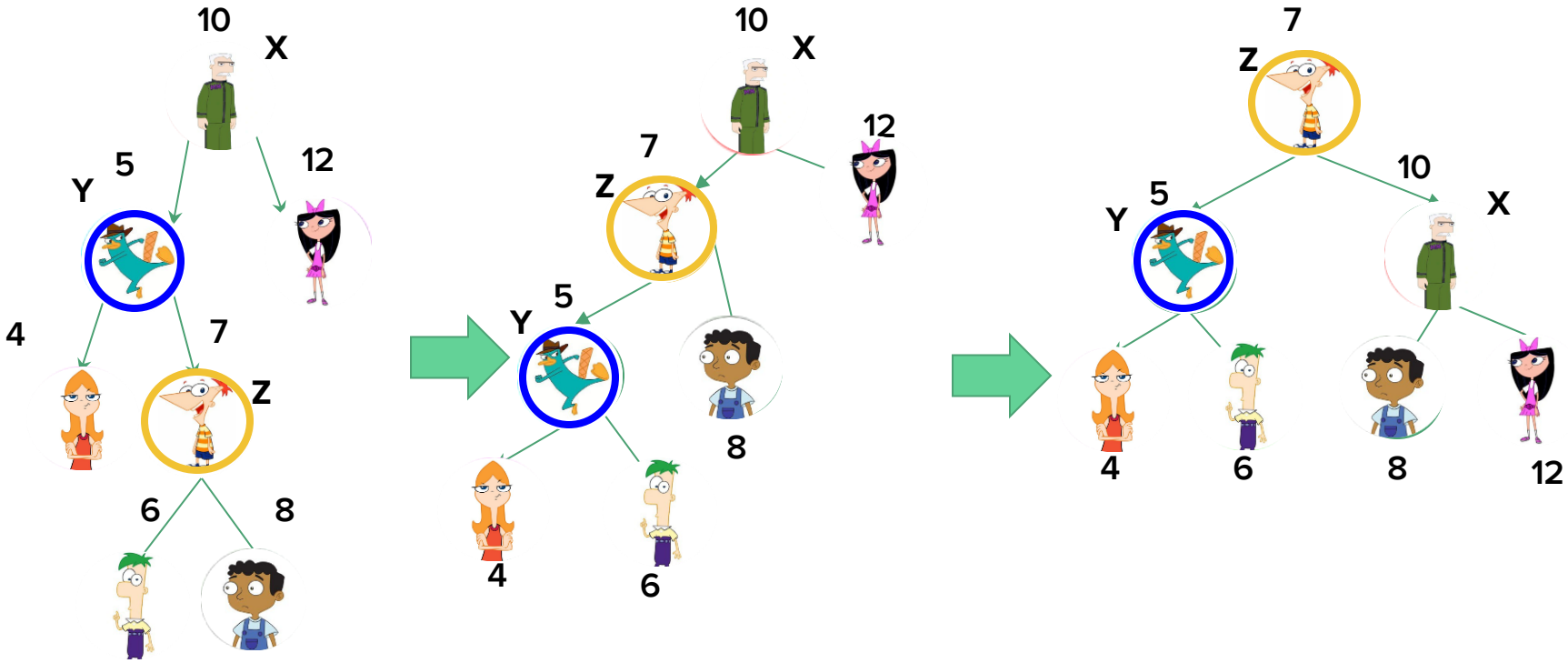
Rotación Simple



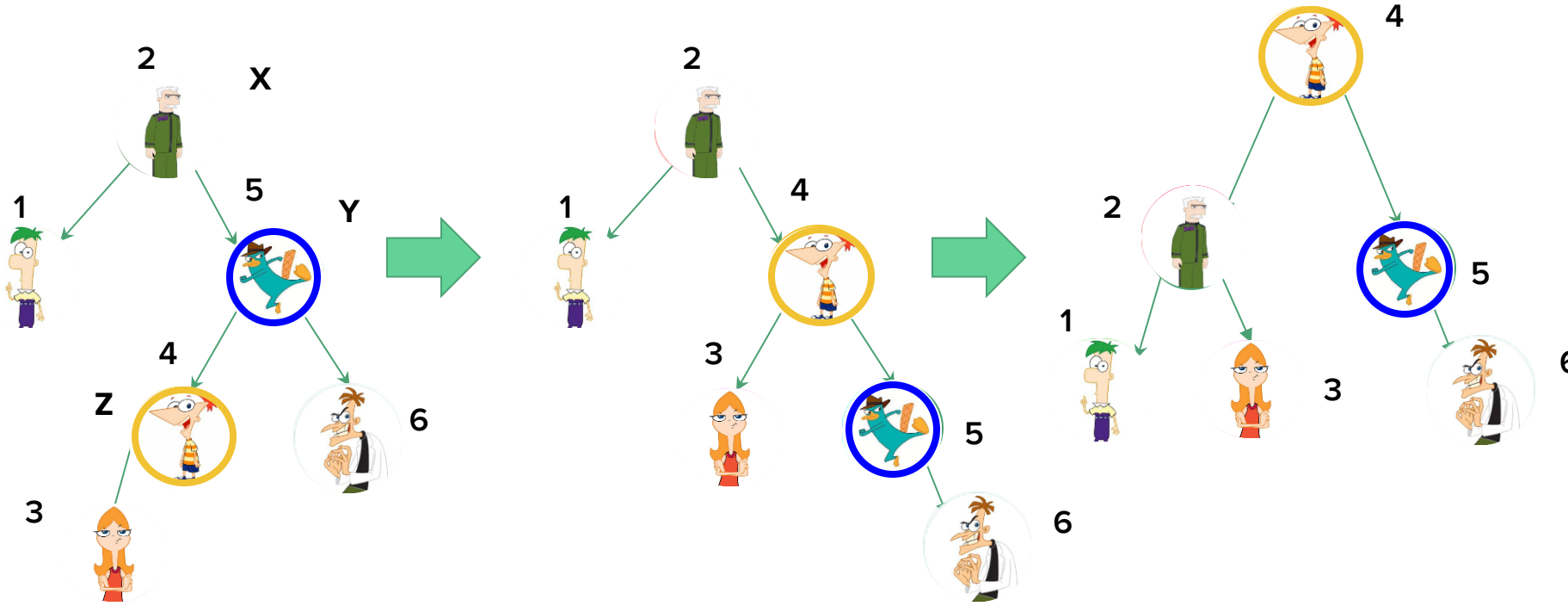
Rotación Simple



Rotación: Doble



Rotación Doble



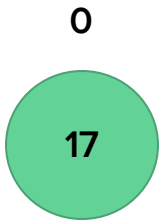
Ejercicio rotación AVL

Considera que tienes un AVL vacío y una lista desordenada:

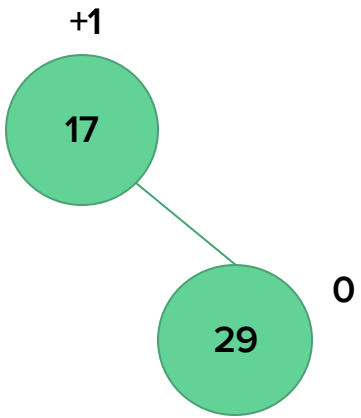
17, 29, 53, 61, 73, 37

Inserta los nodos realizando las rotaciones diferentes

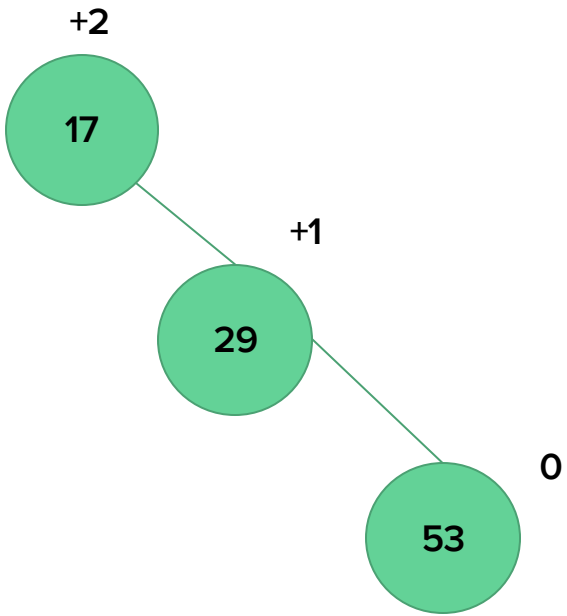
Insertar 17



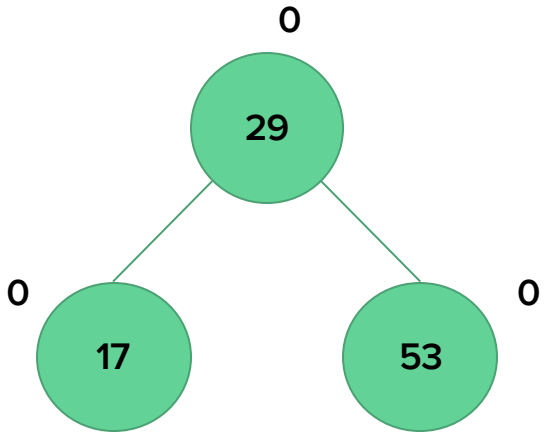
Insertar 29



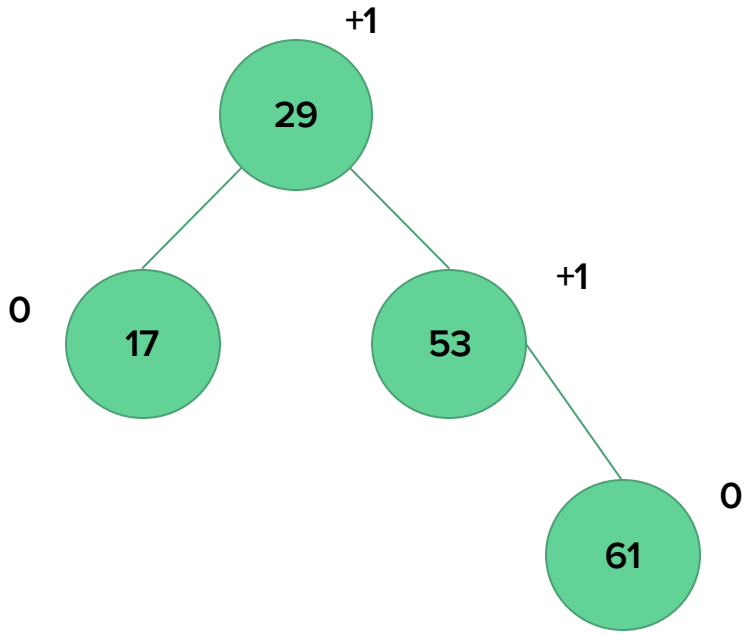
Insertar 53



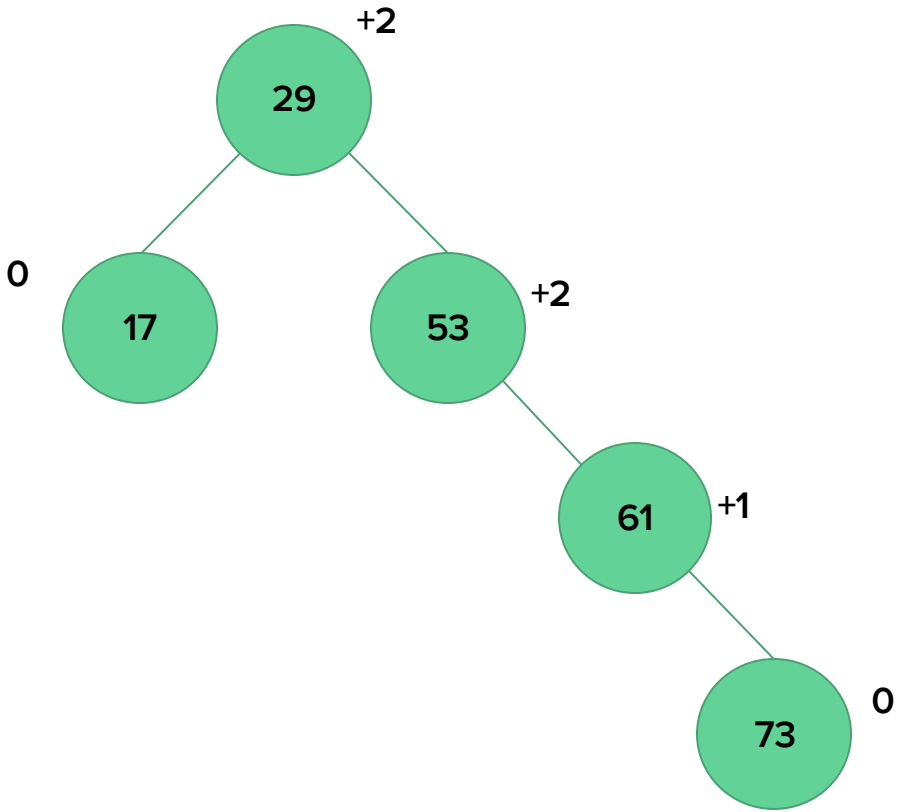
Rotación Simple



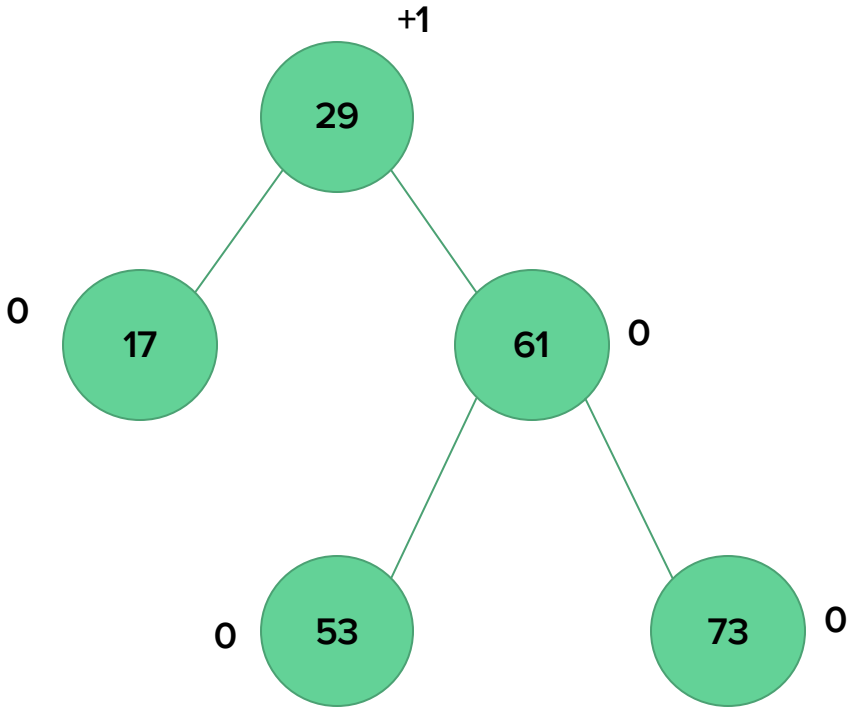
Insertar 61



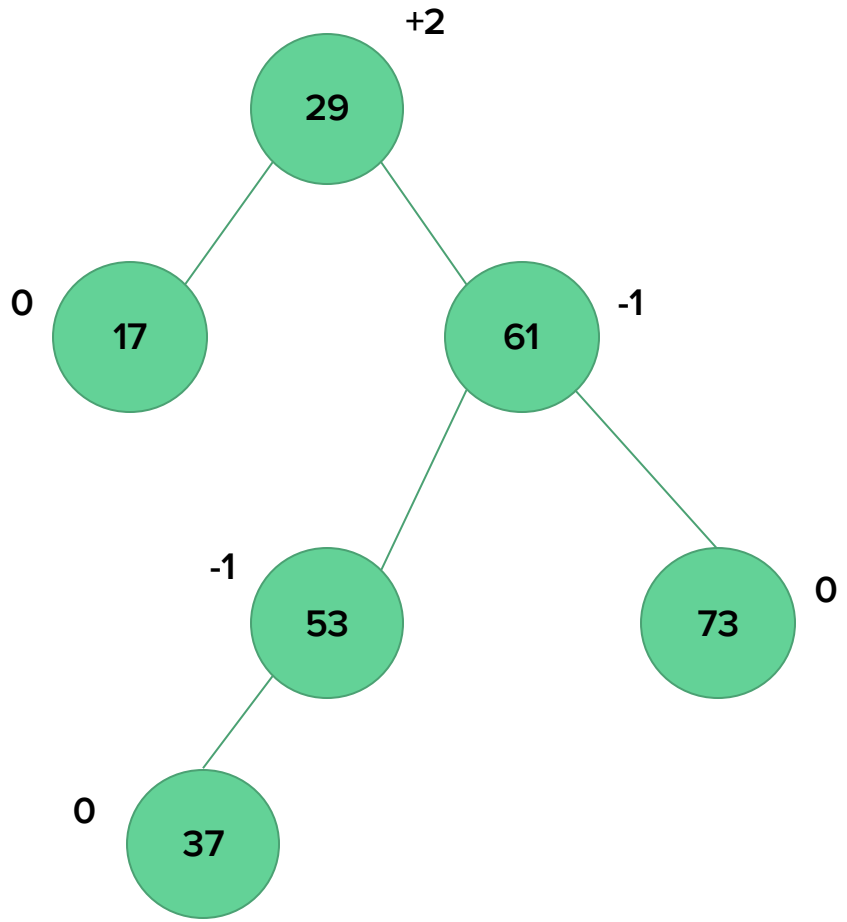
Insertar 73



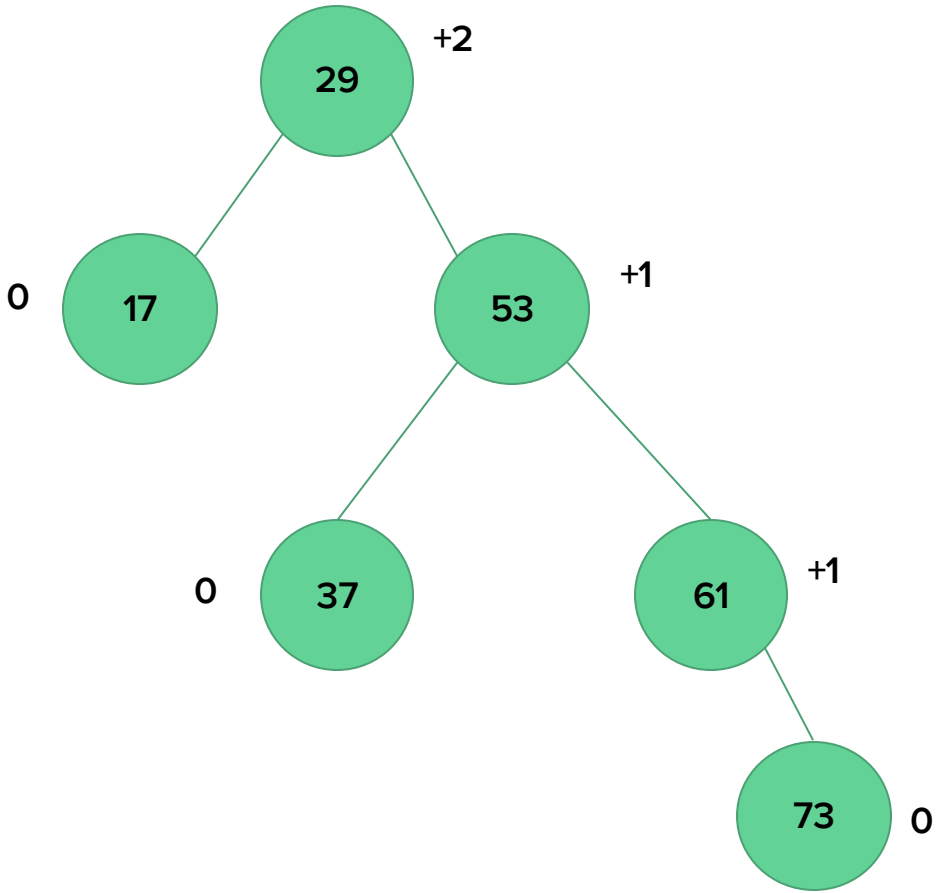
Rotación Simple



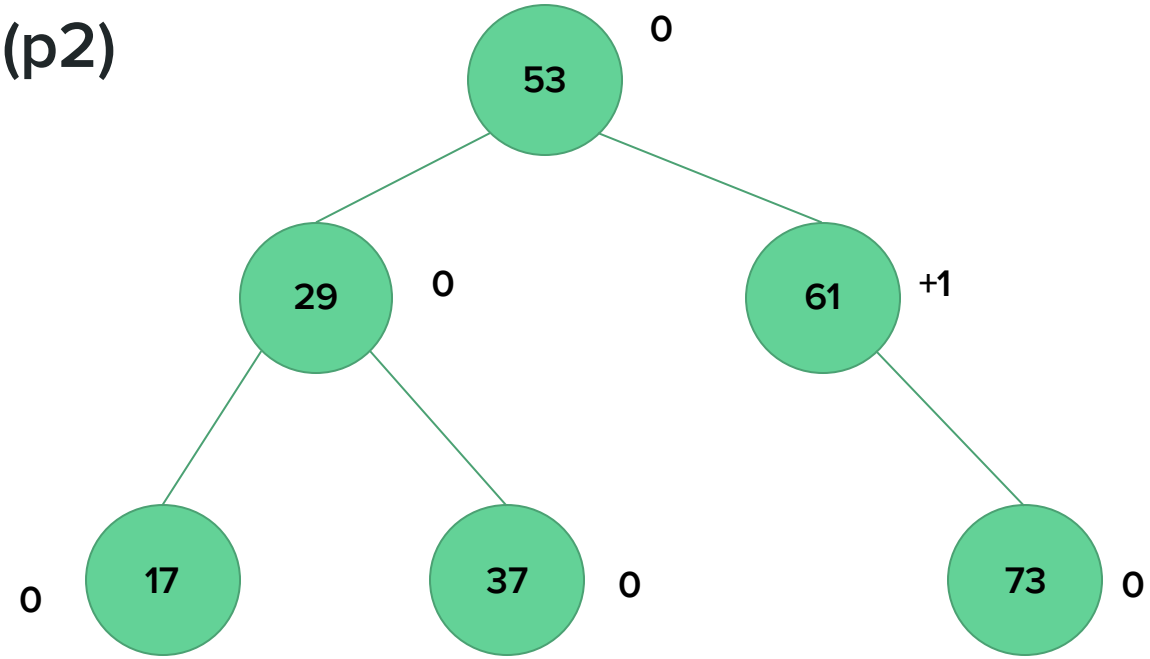
Insertar 37



Rotación doble (p1)



Rotación doble (p2)



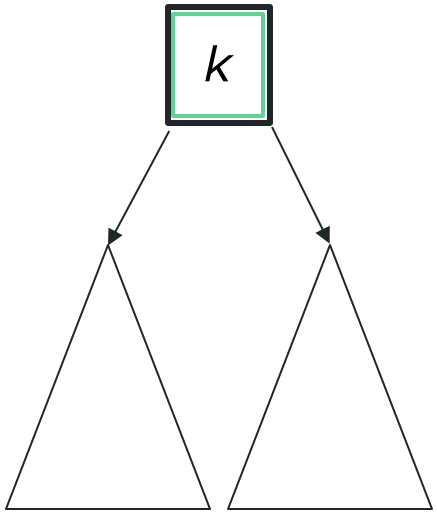
Árboles 2-3

Árboles 2-3

- Queremos tener las hojas del árbol a la misma profundidad → garantizamos profundidad O(log(n)) para operaciones de búsqueda
- Tendremos 2 tipos de **nodos**:
  - Cada **nodo** está compuesto por 1 o 2 **valores** ordenados.
  - Cada nodo puede tener exacta- y respectivamente 2 o 3 hijos (o 0 en caso de ser un nodo hoja). Si el nodo tiene 2 hijos, es un 2-nodo. Si tiene 3 hijos, es un 3-nodo
  - En un 2-nodo, que tiene un valor k:
    - el hijo izquierdo cumplirá con ser menor que k
    - el hijo derecho cumplirá con ser mayor que k
  - En un 3-nodo, que tiene dos valores ordenados k_1 y k_2:
    - el hijo izquierdo cumplirá con ser menor que k_1
    - el hijo central estará entre k_1 y k_2
    - el hijo derecho cumplirá con ser mayor que k_2

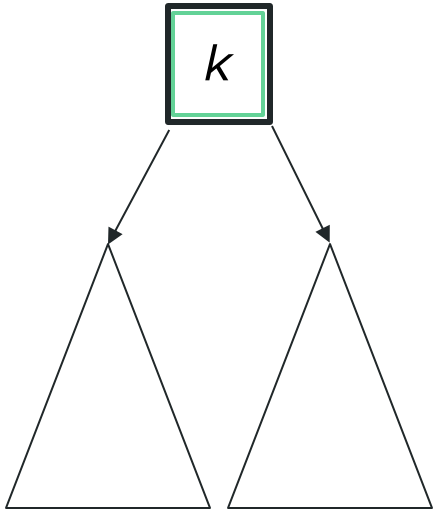
Árboles 2-3: Ejemplos

2-Nodo

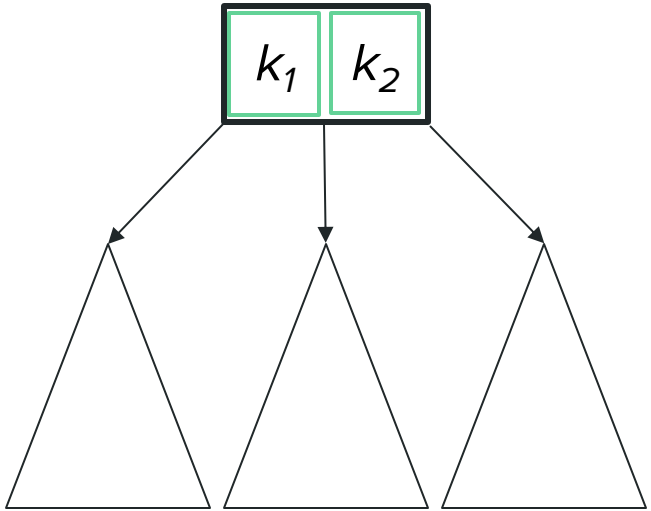


Árboles 2-3: Ejemplos

2-Nodo

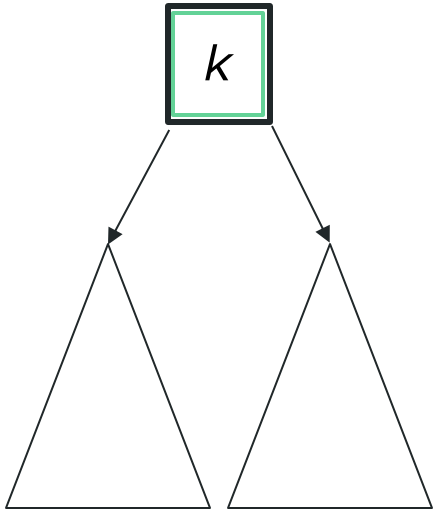


3-Nodo

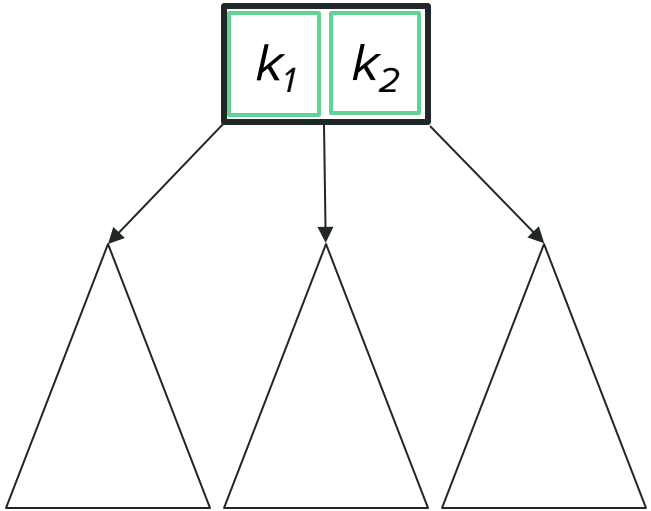


Árboles 2-3: Ejemplos

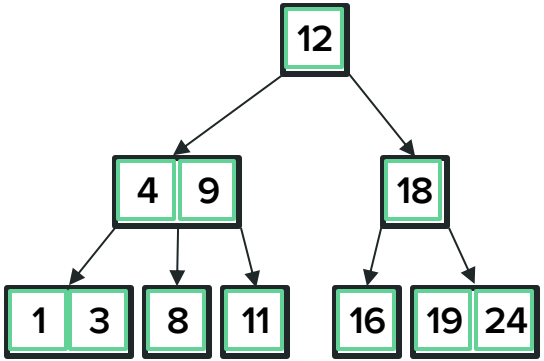
2-Nodo



3-Nodo



Árbol 2-3



Construyendo el árbol 2-3

- La construcción se hace en base a inserciones
- La idea es construir el árbol de “abajo hacia arriba”
- La secuencia de pasos es:
  - Insertar el valor en una hoja existente
  - Si el nodo hoja queda como 3-nodo, terminamos
  - Si el nodo hoja queda como 4-nodo (con 3 valores), tenemos que subir el valor del medio al siguiente nivel y crear dos 2-nodos con los valores que quedan en el nivel. Cada uno de estos será apuntado por el lado izquierdo y derecho del valor que subió según corresponda (ver ejemplo)
  - Se repite la modificación de forma recursiva hacia la raíz
- Para saber en qué hoja insertar, buscamos como si fuese un ABB (vamos descendiendo por el árbol)

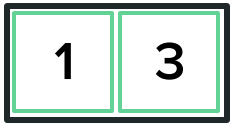
Construyendo el árbol 2-3

Crearemos un árbol 2-3 con los siguientes valores:

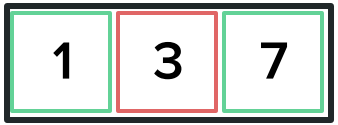
[3, 1, 7, 8, 9, 2, 10, 0, 4, 5, 6, -1]

Valores restantes: [3, 1, 7, 8, 9, 2, 10, 0, 4, 5, 6, -1]

Valores restantes: [1, 7, 8, 9, 2, 10, 0, 4, 5, 6, -1]

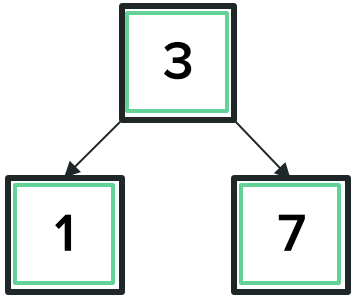


Valores restantes: [7, 8, 9, 2, 10, 0, 4, 5, 6, -1]

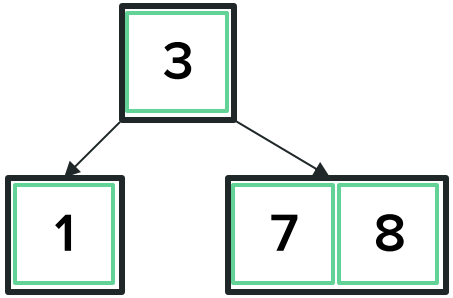


Toca subir el valor del centro

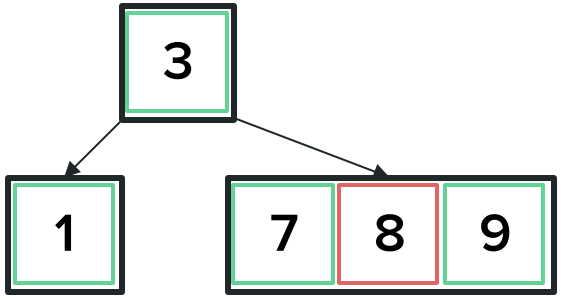
Valores restantes: [7, 8, 9, 2, 10, 0, 4, 5, 6, -1]



Valores restantes: [8, 9, 2, 10, 0, 4, 5, 6, -1]

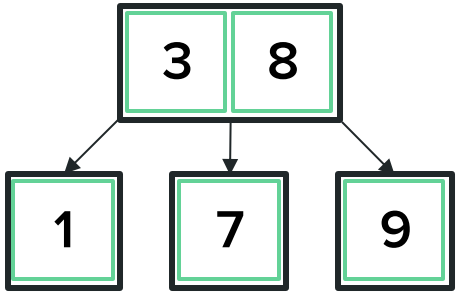


Valores restantes: [9, 2, 10, 0, 4, 5, 6, -1]

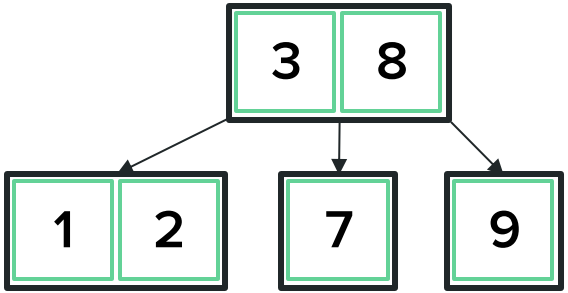


Toca subir el valor del centro

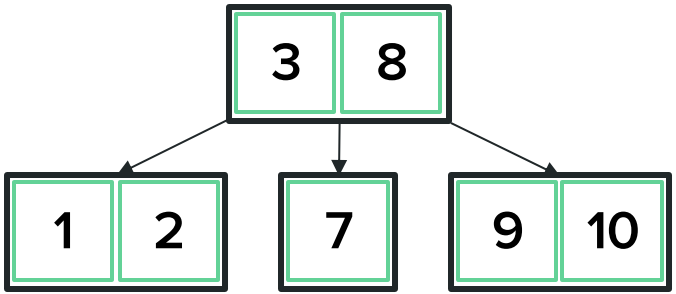
Valores restantes: [9, 2, 10, 0, 4, 5, 6, -1]



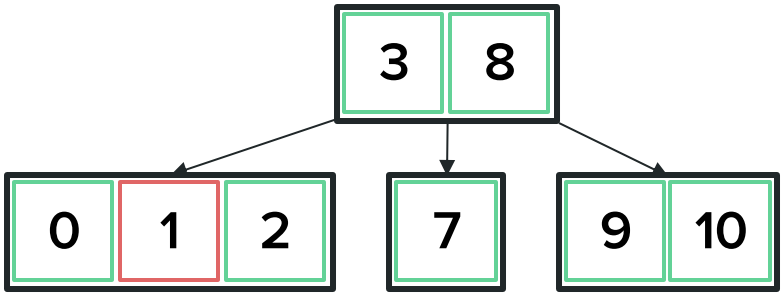
Valores restantes: [2, 10, 0, 4, 5, 6, -1]



Valores restantes: [10, 0, 4, 5, 6, -1]

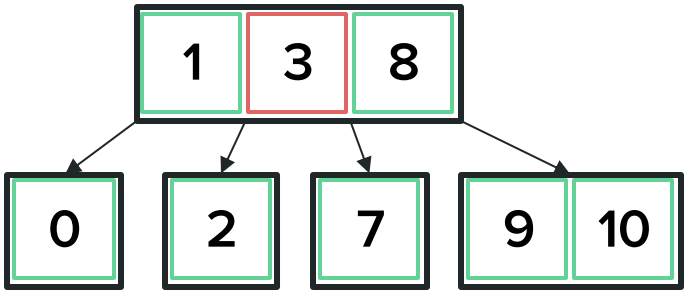


Valores restantes: [0, 4, 5, 6, -1]



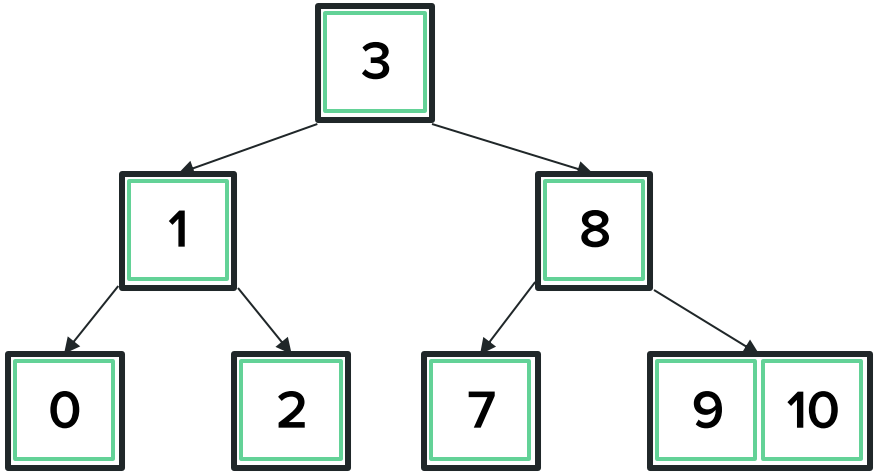
Toca subir el valor del centro

Valores restantes: [0, 4, 5, 6, -1]

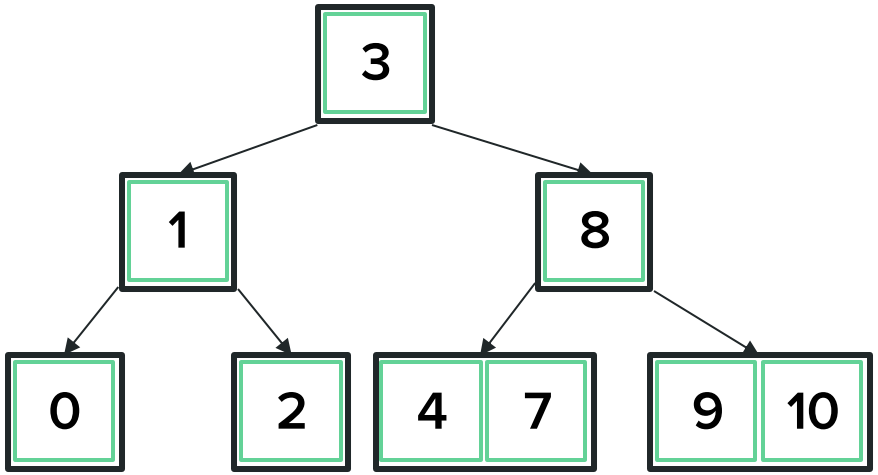


Toca subir el valor del centro

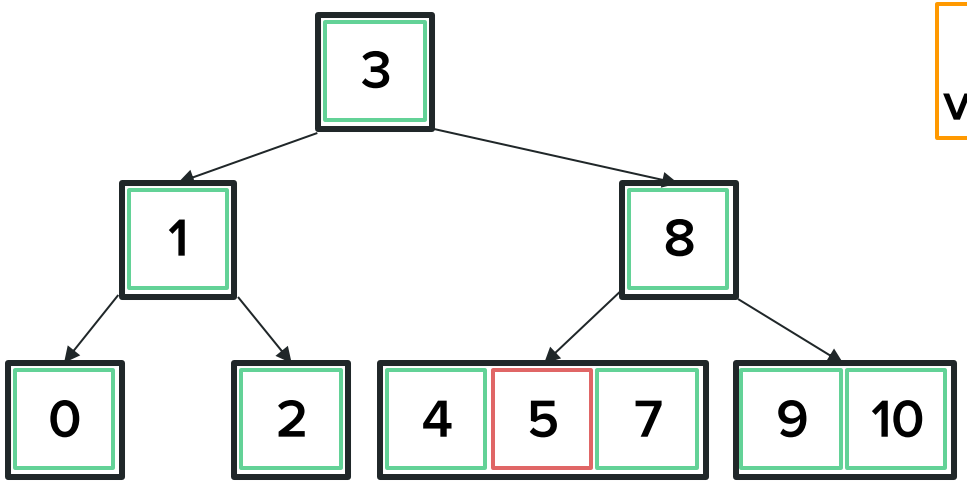
Valores restantes: [0, 4, 5, 6, -1]



Valores restantes: [4, 5, 6, -1]

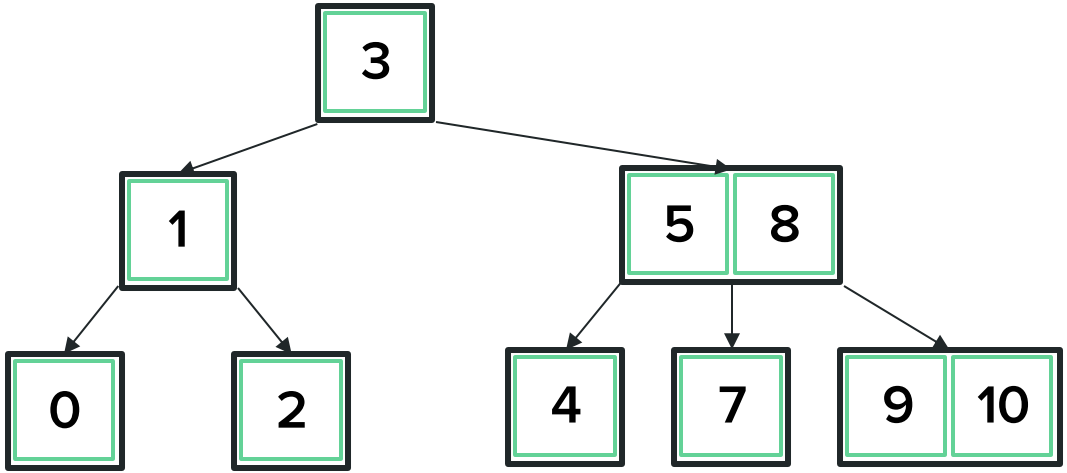


Valores restantes: [5, 6, -1]

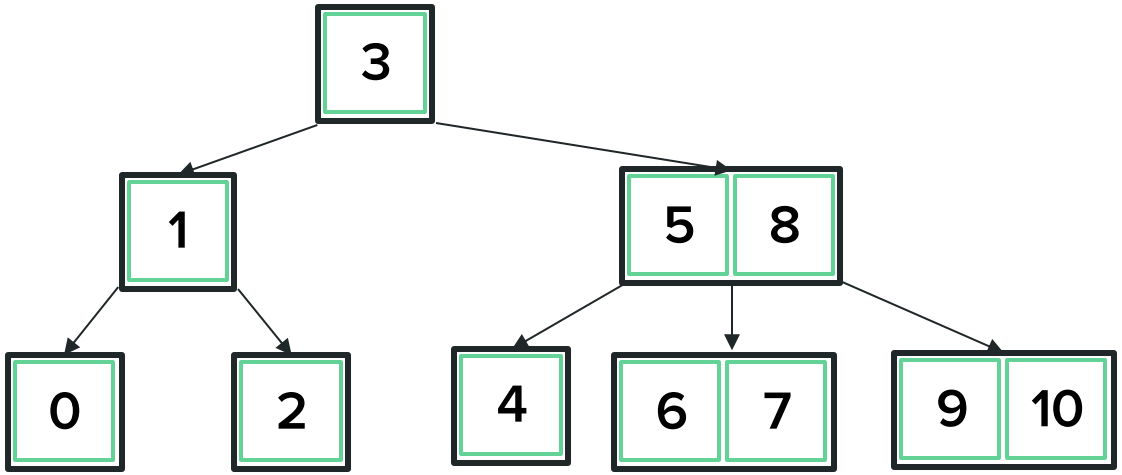


Toca subir el valor del centro

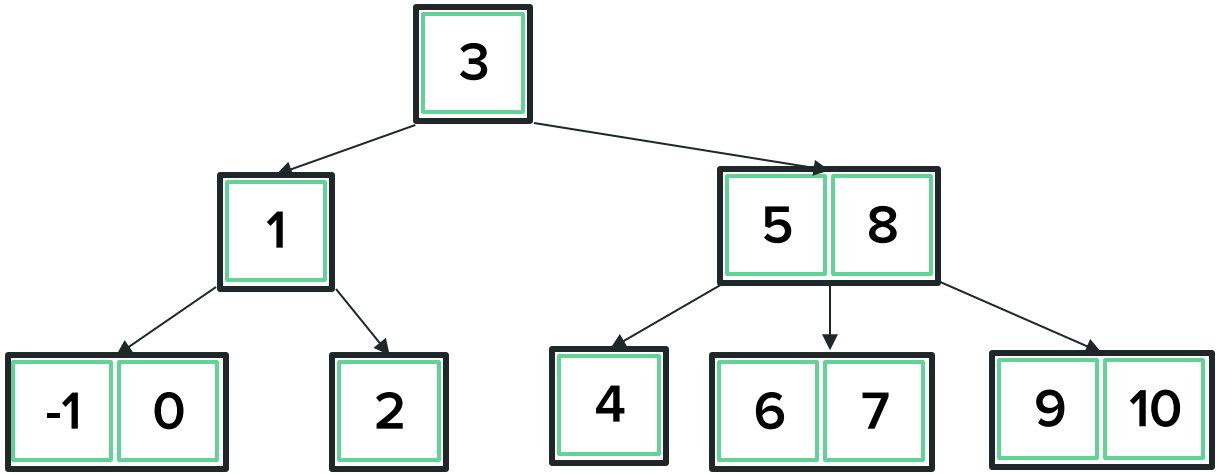
Valores restantes: [5, 6, -1]



Valores restantes: [6, -1]

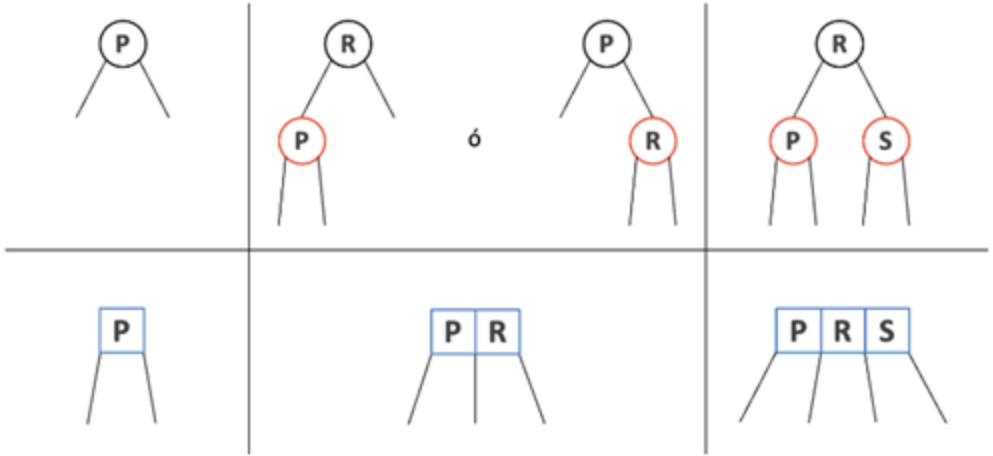


Valores restantes: [-1]

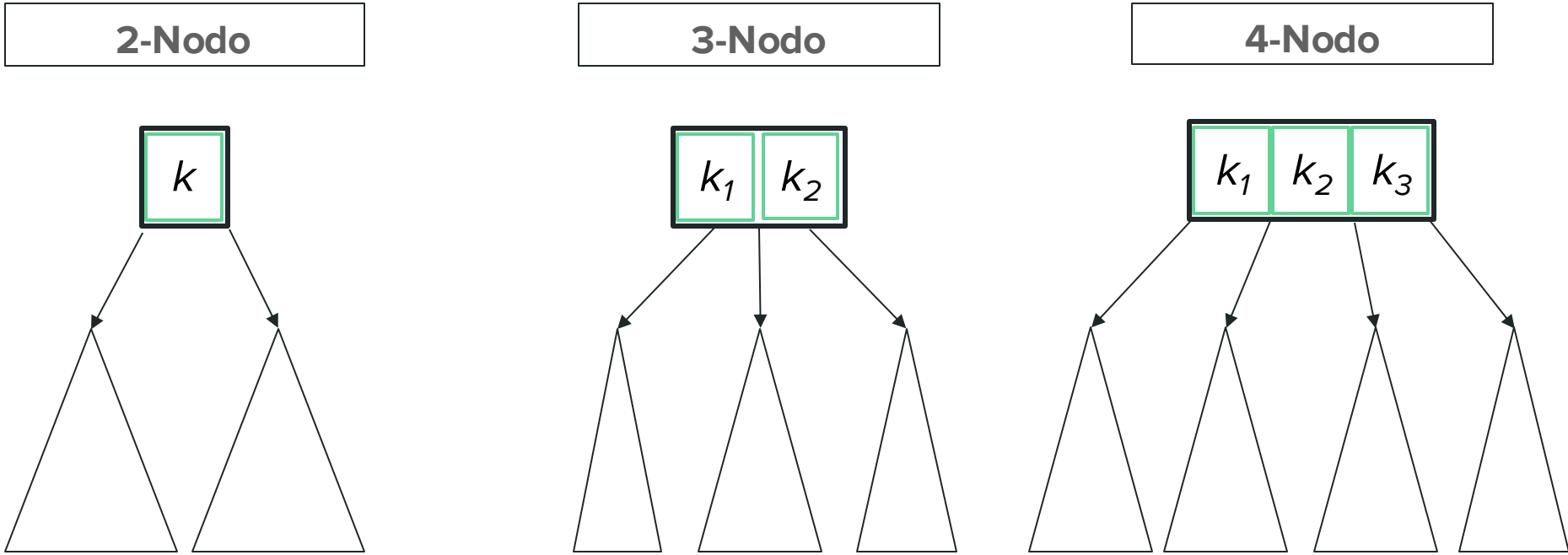


Árboles 2-4

- Igual que un árbol 2-3, pero que ahora permite 4-nodos.
- Tendremos 3 tipos de **nodos**:
  - Los mismos de antes (2-nodos y 3-nodos)
  - 4-nodo: está compuesto por 3 valores ordenados y tiene 4 hijos (salvo que el nodo sea hoja).
- Equivalencia con árboles rojo-negro



Árboles 2-4: Ejemplos



I2 2020-1 Árbol 2-3

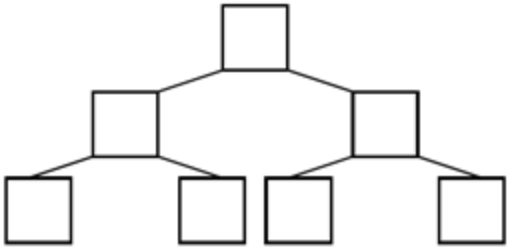
Los árboles 2-3 son árboles de búsqueda en que los nodos tienen ya sea una clave y dos hijos, o bien dos claves y tres hijos; y todas las hojas del árbol (que se exceptúan de la regla anterior porque no tienen hijos) están a la misma profundidad. Teniendo presente estas propiedades, responde:

a) ¿Cuál es la **altura máxima** que puede tener un árbol 2-3 con n elementos? ¿Y la **mínima**? ¿Cómo es la **estructura del árbol** cuando ocurre cada uno de estos casos?

a) **Altura máxima** que puede tener un árbol 2-3 con n elementos

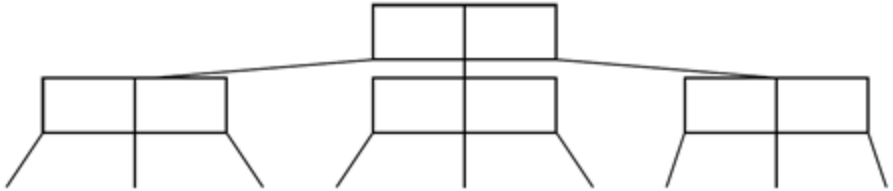
La altura será mayor mientras menos elementos tenga el árbol por nivel, y viceversa.

(a) Sólo nodos 2



Caso altura máxima

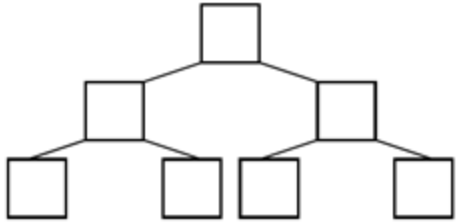
(b) Sólo nodos 3



Caso altura mínima

Caso altura máxima

(a) Sólo nodos 2



- Cantidad elementos para el k-ésimo nivel:

$$2^{k-1}$$

- Cantidad elementos **acumulados** hasta el k-ésimo nivel:

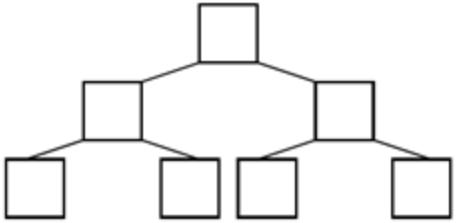
$$2^k - 1$$

Supongamos que nuestro árbol tiene n nodos y una altura h. Cuando el árbol esté completamente lleno tendremos:

$$n = 2^h - 1 \longrightarrow h = \log_2(n+1)$$

Caso altura máxima

(a) Sólo nodos 2



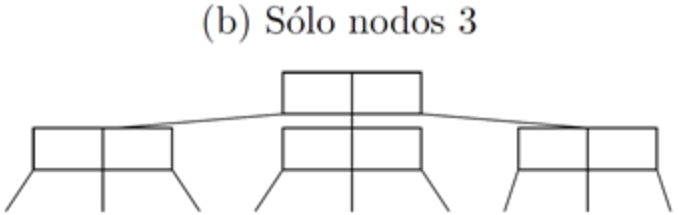
Cuando el árbol esté completamente lleno tendremos:

$$h = \log_2(n+1)$$

Como las hojas deben estar a la misma altura, al añadir un valor extra, el árbol se rebalancea manteniendo su altura, quedando:

$$\lfloor h = \log_2(n+1) \rfloor$$

Caso altura mínima

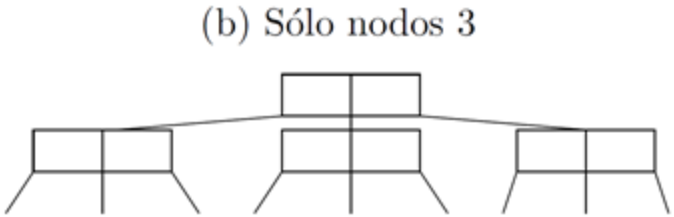


- Cantidad elementos para el k-ésimo nivel: $2 \cdot 3^{k-1}$
- Cantidad elementos acumulados hasta el k-ésimo nivel: $3^k - 1$

Supongamos que nuestro árbol tiene n nodos y una altura h. Cuando el árbol esté completamente lleno tendremos:

$$n = 3^h - 1 \longrightarrow \mathrm{h} = \log_3(n+1)$$

Caso altura mínima



Cuando el árbol esté completamente lleno tendremos:

$$h = \log_3(n+1)$$

Para alcanzar un nivel “k” de altura, debe exceder necesariamente los k - 1 niveles anteriores, y no superar los k niveles. Por lo tanto:

$$\lceil h = \log_3(n+1) \rceil$$

I2 2020-1 Árbol 2-3

Los árboles 2-3 son árboles de búsqueda en que los nodos tienen ya sea una clave y dos hijos, o bien dos claves y tres hijos; y todas las hojas del árbol (que se exceptúan de la regla anterior porque no tienen hijos) están a la misma profundidad. Teniendo presente estas propiedades, responde:

a) Lista ✓
b) Queremos **insertar una clave X** en un **árbol 2-3 T de altura h**, que tiene **n claves**. ¿Qué debe cumplirse para que esta inserción aumente la altura de T? ¿Para qué **valores de n** está garantizado que sí **aumentará la altura?** ¿Para qué valores de n está garantizado que **no aumentará la altura?**

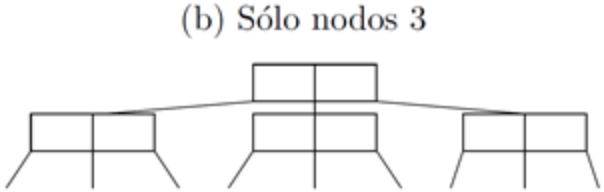
I2 2020-1 Árbol 2-3 b)

Queremos que al insertar la clave X aumente la altura del árbol 2 - 3.

- “**camino**” de un árbol T: nodos que hay que recorrer desde la raíz para llegar a un nodo hoja

De esta forma, para que una **inserción provoque el aumento de altura** de un árbol T se debe cumplir que, en el **camino a la hoja** donde se insertará la nueva clave, **todos los nodos sean nodos 3** (es decir, que tengan 2 claves). Así, se haría un **split desde la hoja donde insertamos hasta la raíz**, aumentando finalmente la altura

Vamos a analizar 2 casos: (altura h con n claves)



1. **Garantía de que aumenta la altura:**

El árbol T debe tener **solo nodos 3**. Notemos que si pasa lo contrario (al menos uno no es un nodo 3), entonces podemos buscar el camino en donde podamos insertar de tal forma que este nodo pase a ser nodo 3, lo que haría que no aumente la altura. Teniendo esto, la cantidad de claves en
función de la altura h es:

$$claves = \sum_{i=1}^{h} 2 \cdot \left(3^{i-1}\right)$$

en un nivel i hay 3^{i-1} nodos, con cada uno 2 claves

$$claves = 3^h - 1$$

2. Garantía de que no aumenta la altura:

El árbol T **no debe tener ningún camino que tenga únicamente nodos 3**. Notemos por la definición del comienzo que la condición para que aumente la altura es que agreguemos una clave en la hoja que tenga un camino con sólo nodos 3, por lo que debemos asegurarnos que no exista tal
camino. Luego, para **garantizar esto debemos contar la cantidad de claves para que no exista un árbol con dicho camino**.

2. Garantía de que no aumenta la altura:

La menor cantidad de claves con la que podría aumentar la altura es tal que todos sean nodos 2 excepto un camino en donde sean nodos 3. Teniendo esto, si tomamos esa cantidad de claves - 1 no podremos tener un camino con nodos 3, y garantizaremos desde esa cantidad hacia abajo que no aumente la altura.

- nivel $1 \rightarrow 2$ claves (nodo 3)
- nivel $2 \rightarrow 4$ claves (nodo $3 + 2$ nodos 2)
- nivel $3 \rightarrow 8$ claves (nodo $3 + 6$ nodos 2)

$\vdots$

- nivel $h \rightarrow 2^h$ claves (nodo $3 + 2^h - 2$ nodos 2)

$$\longrightarrow \quad claves = \sum_{i=1}^{h} 2^i - 1$$

2. **Garantía de que no aumenta la altura:**

$$claves = \sum_{i=1}^{h} 2^i - 1$$

$$claves = 2^{h+1} - 2$$

Luego, para garantizar que la inserción en el árbol T no tendrá efectos en la altura h, la cantidad de claves debe ser menor a $2^{h+1} - 2$

Además, para la factibilidad de un árbol 2-3, se debe cumplir que las claves sean mayores a $2^h - 1$ que es la cantidad mínima de claves que un árbol 2-3 de altura h puede tener.