Árboles

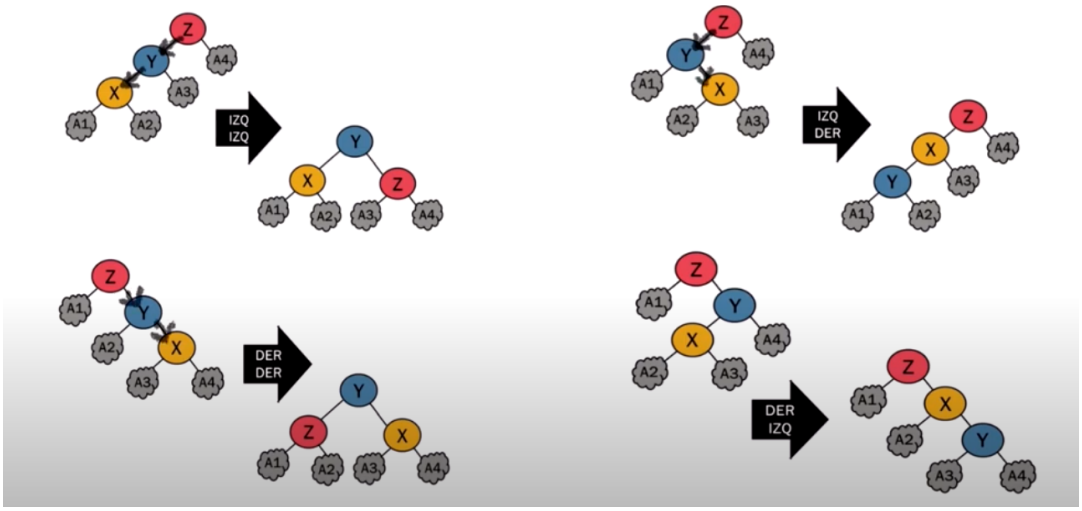
- Todos los arboles son grafos pero no todos los grafos son arboles.
- Estructura de datos en la que un nodo tiene un unico padre y 0 o mas hijos, siempre debe tener una raiz.

```
public class Node {
    public Node[] children;
    public int value;

    public node(int value){
        this.value = value;
    }
}
```

- El arbol se opera pasando el nodo raiz.
- Complejidad: O(log n) *depende de si esta bien balanceado.
- Tipos:
  - Binary Tree:
    - Un nodo tiene un unico padre y maximo 2 hijos.
  - Binary Search Tree:
    - Un nodo tiene un unico padre y maximo 2 hijos.
    - Para cada nodo n, descendientes de la izq ≤ n < descendientes de la der.
  - Binary Complete Tree:

        - Arbol binario en el que todos los niveles deben estar completos a excepcion del ultimo, en caso de que este incompleto debe llenarse de izq a der.
    - Binary AVL Tree:
        - Es un árbol de búsqueda binario autoequilibrado que mantiene su altura mínima.
        - Para cada nodo, la altura de sus subárboles izquierdo y derecho difieren en no más de uno.
        - Estos árboles son útiles en aplicaciones que requieren búsquedas rápidas, como en motores de bases de datos.
        - Rotaciones:



    - Red-Black Tree:
        - Es un tipo de árbol binario de búsqueda auto-equilibrado donde cada nodo tiene un atributo de color: rojo o negro.
        - Se asegura que el camino más largo desde la raíz hasta cualquier hoja no es más del doble que el más corto.
        - Se usa ampliamente en la implementación de bibliotecas de lenguajes de programación.
- Metodos de busqueda:

    - In Order Traversal: Rama izq del nodo, propio nodo, rama der del nodo.
    - Pre Order Traversal: Propio nodo, rama izq del nodo, rama der del nodo.
    - Post Order Traversal: Rama izq del nodo, rama der del nodo, propio nodo.
- Usar cuando: Se necesite hacer busquedas e inserciones en un conjunto de datos.
- No es eficiente para ninguna operacion pero tampoco es ineficiente.

```
class TreeNode {
    int val;
    TreeNode left, right;

    public TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
}

public class BinaryTreeSize {
    // Función para obtener el tamaño del árbol binario
    public int treeSize(TreeNode root) {
        if (root == null) {
            return 0;
        }
        // Recursivamente calcular el tamaño del subárbol izquie
        int leftSize = treeSize(root.left);
        int rightSize = treeSize(root.right);
        // Retornar el tamaño total del árbol, que incluye el no
        return 1 + leftSize + rightSize;
    }
```

```
    // Ejemplo de uso
    public static void main(String[] args) {
        BinaryTreeSize treeSizeCalculator = new BinaryTreeSize(
        // Crear un árbol binario de ejemplo
        TreeNode root = new TreeNode(1);
        root.left = new TreeNode(2);
        root.right = new TreeNode(3);
        root.left.left = new TreeNode(4);
        root.left.right = new TreeNode(5);
        // Calcular el tamaño del árbol
        int size = treeSizeCalculator.treeSize(root);
        System.out.println("El tamaño del árbol binario es: " +
    }
}
```

```
Función calcularAltura(nodo):
    Si nodo es nulo:
        Devolver 0
    Sino:
        alturaIzquierda = calcularAltura(nodo.izquierdo)
        alturaDerecha = calcularAltura(nodo.derecho)
        Devolver máximo(alturaIzquierda, alturaDerecha) + 1
```