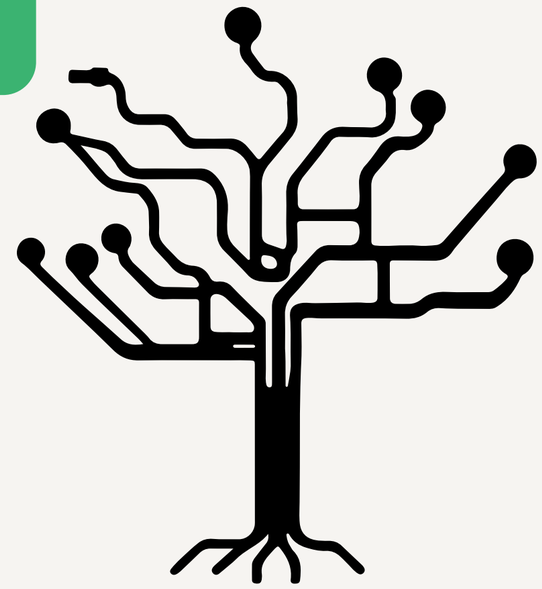
# Trabajo Práctico Integrador II

## Programación I

# IMPLEMENTACIÓN DE ÁRBOLES EN PYTHON



**ALUMNAS**

Alexia Rubin

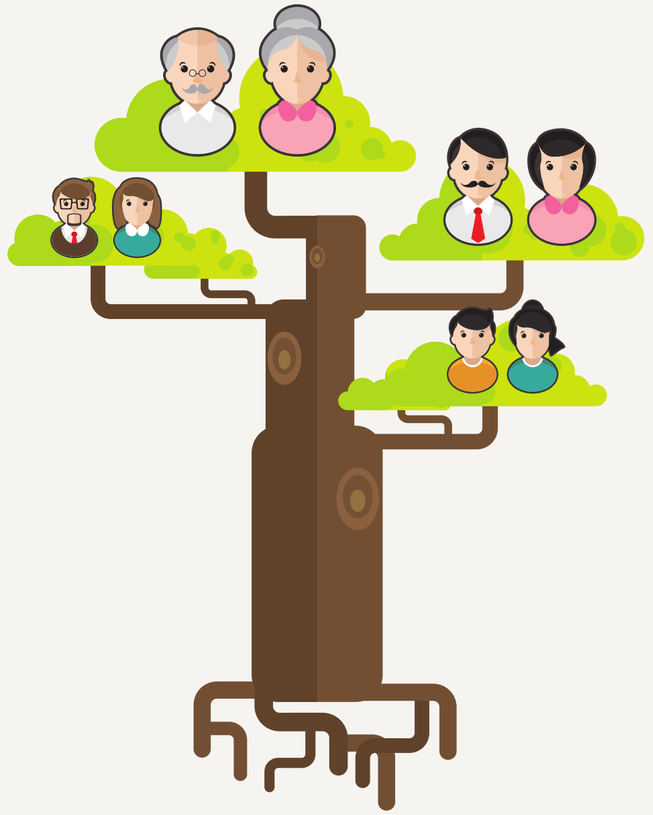
María Victoria Volpe

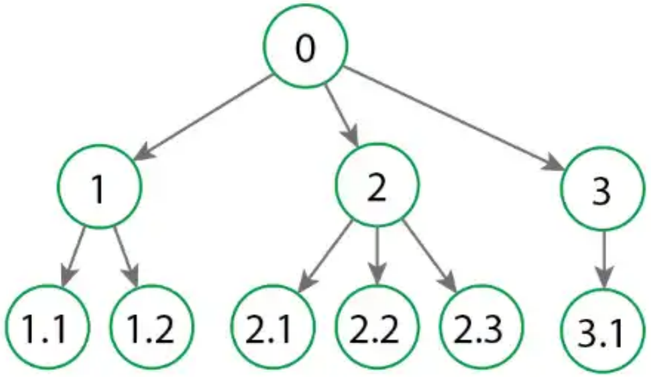
# Objetivos

01 Marco teórico – Árboles en Python

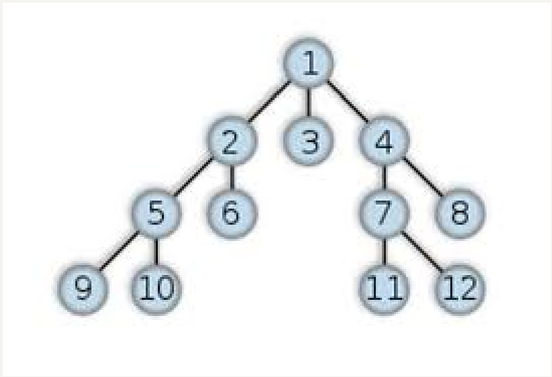
02 Caso Práctico 1

03 Conclusiones






**Árbol: estructura jerárquica no lineal sin ciclos, con un nodo raíz y nodos conectados por ramas.**




**Árbol binario: cada nodo tiene como máximo dos hijos (izquierda y derecha).**

# Aplicaciones de los árboles

🔹 **Sistemas de archivos**
**Organización jerárquica como carpetas y subcarpetas.**
🔹 **Bases de datos**
**Uso de estructuras como B-trees para índices.**
🔹 **Inteligencia artificial**
**Árboles de decisión para tomar decisiones automáticas.**

# Ventajas de los Árboles Binarios

✓ **Permiten búsqueda y organización eficiente (por ejemplo, árboles BST).**
✓ **Fáciles de recorrer de forma sistemática y ordenada.**
✓ **Son simples de implementar de manera recursiva.**

# Características

- **Raíz:**

Nodo principal del árbol (no tiene padre).

- **Padre / Hijo:**

Relación directa entre nodos conectados.

👉 Un padre puede tener varios hijos.

- **Hermanos:**

Nodos que comparten el mismo padre.

- **Hojas:**

Nodos sin hijos (están al final del árbol).

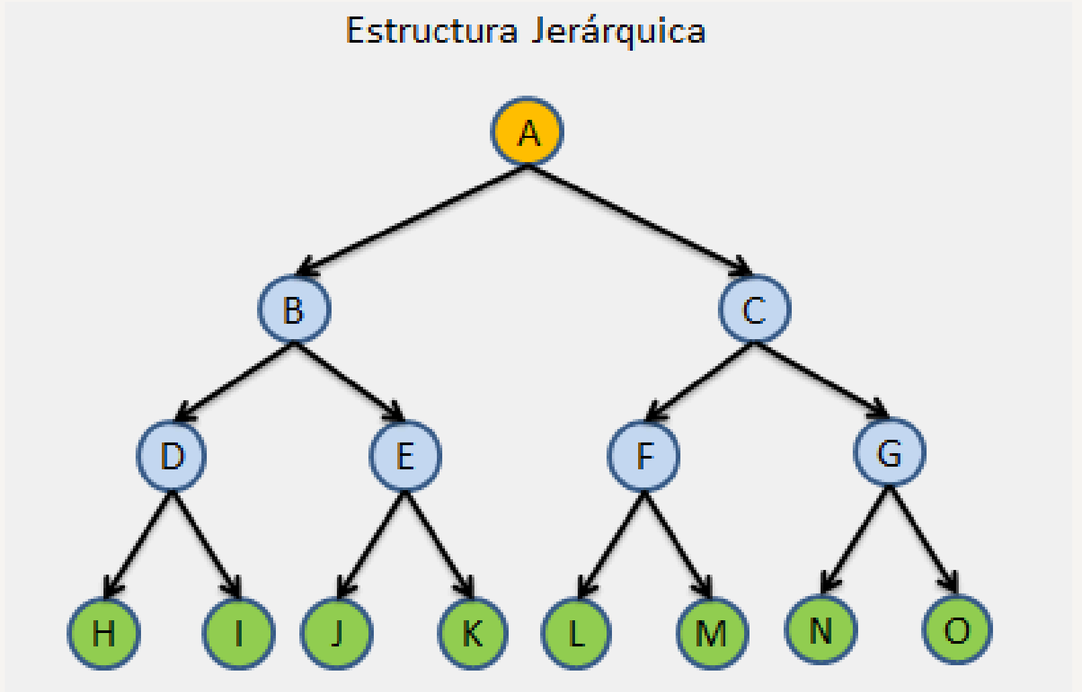
- **Nodos internos:**

Tienen al menos un hijo (no son hojas).

- **Ancestros / Descendientes:**

Relaciones según el recorrido del árbol.

👉 Ejemplo: un abuelo–nieto en estructura de árbol.

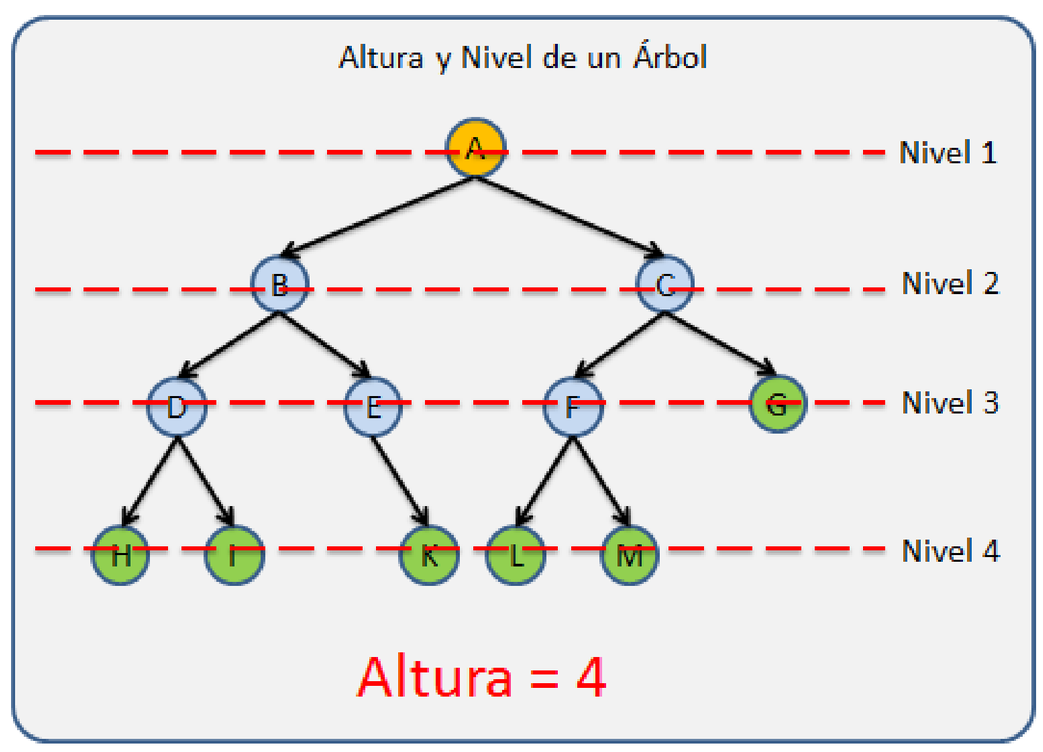


# Propiedades

## Nivel

**cada generación dentro del árbol**

**Un árbol vacío tiene 0 niveles**
**El nivel de la Raíz es 1**
**El nivel de cada nodo se calculado contando cuantos nodos existen sobre el, hasta llegar a la raíz + 1, y de forma inversa también se podría, contar cuantos nodos existes desde la raíz hasta el nodo buscado + 1.**

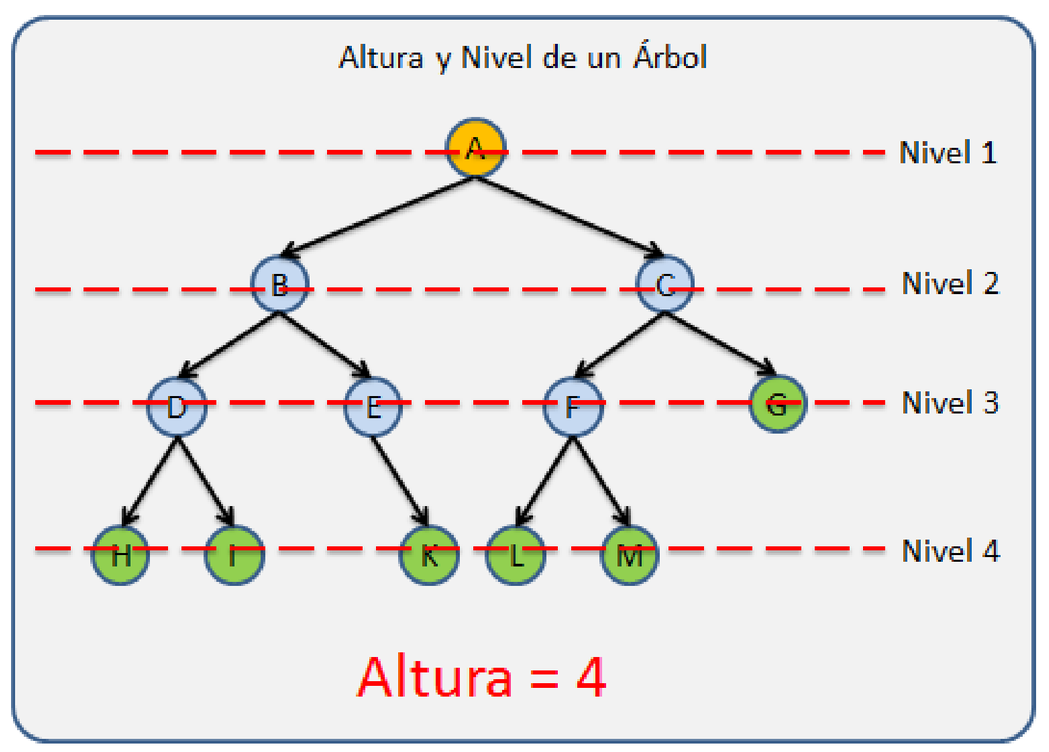



**Fuente de las imágenes sobre árboles: https://www.oscarblancarteblog.com/2014/08/22/estructura-de-datos-arboles/**

# Propiedades

## Altura

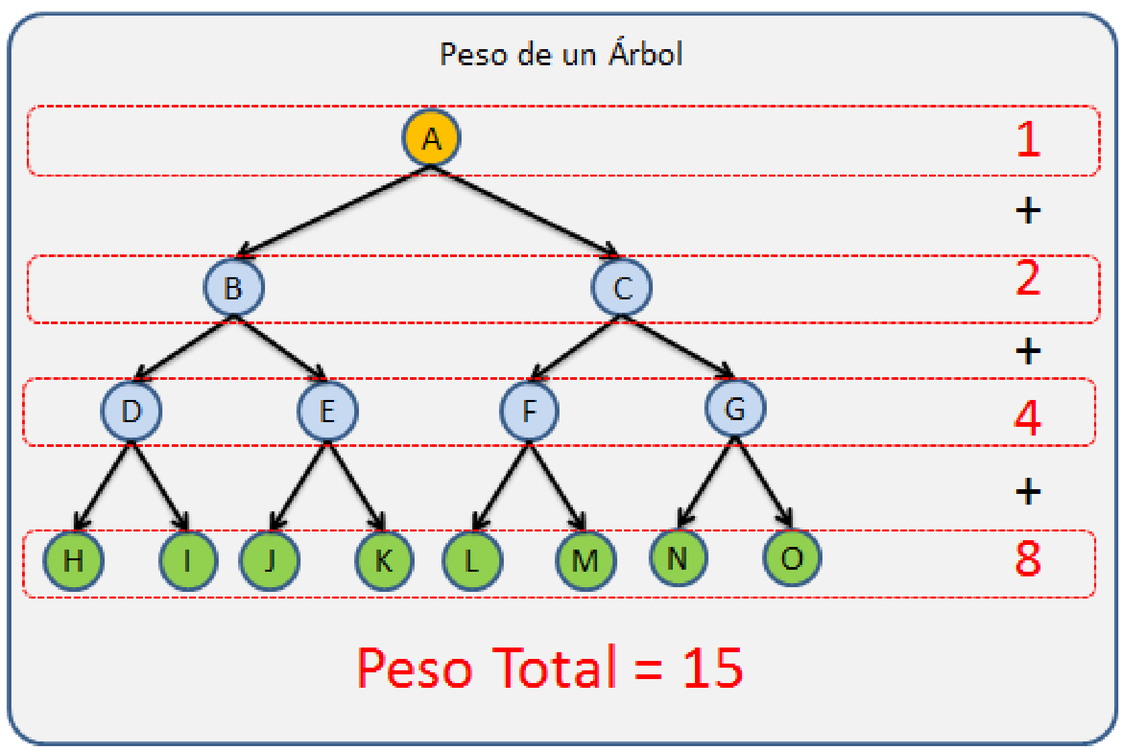
**número máximo de niveles de un Árbol**



# Propiedades

## Peso

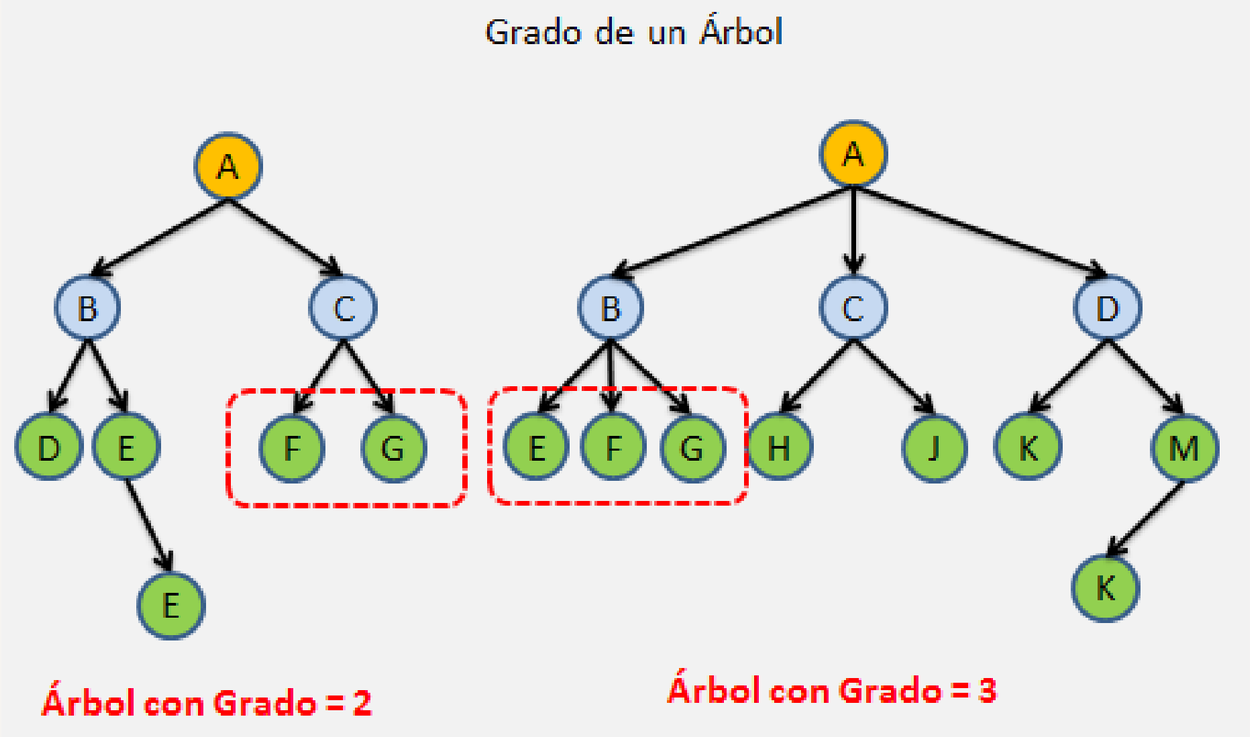
**Conocemos como peso a el número de nodos que tiene un Árbol. Este factor es importante por que nos da una idea del tamaño del árbol y el tamaño en memoria que nos puede ocupar en tiempo de ejecución**



# Propiedades

## Grado

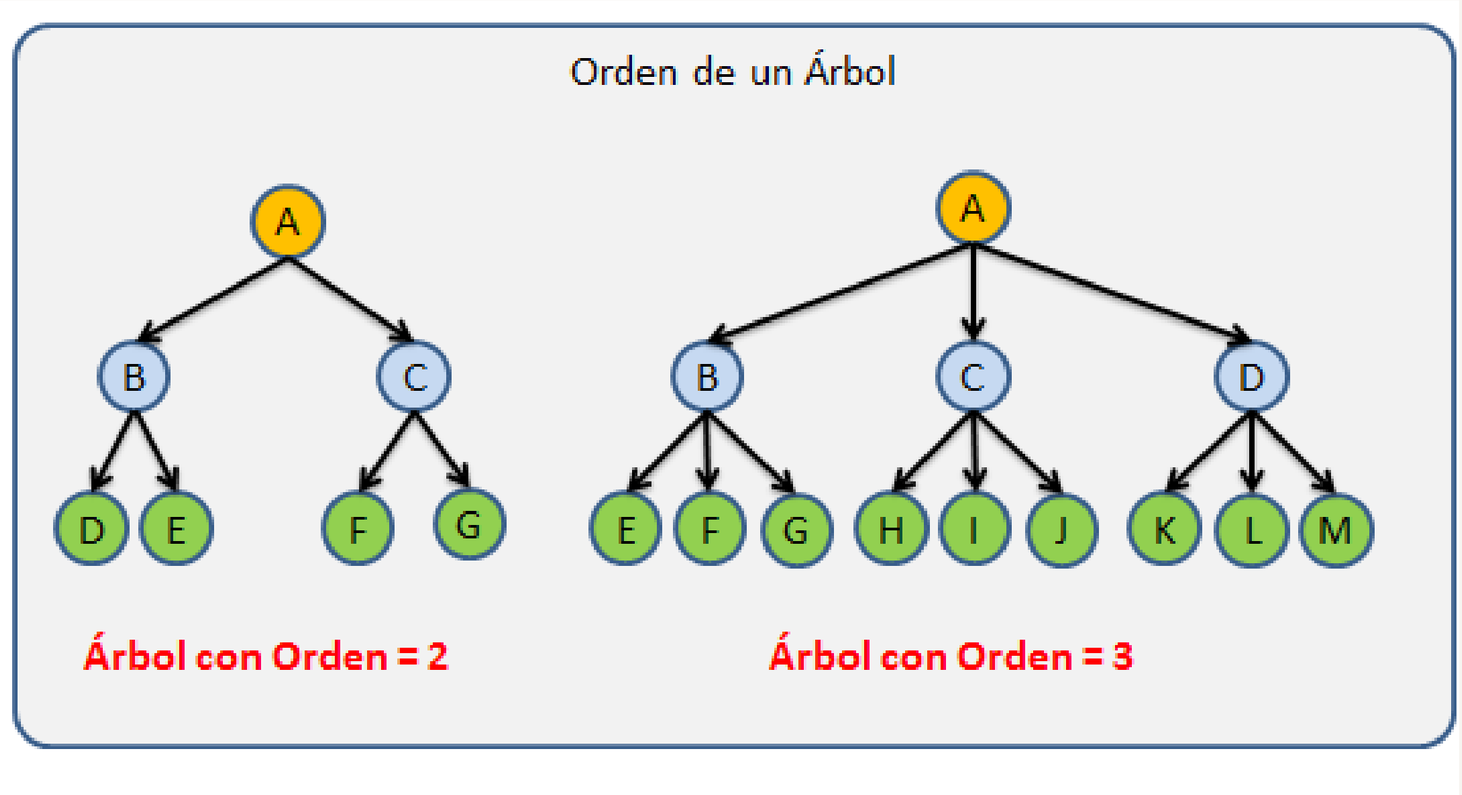
**El grado de un árbol en Python se calcula encontrando el nodo con el mayor número de hijos**



# Propiedades

## Orden

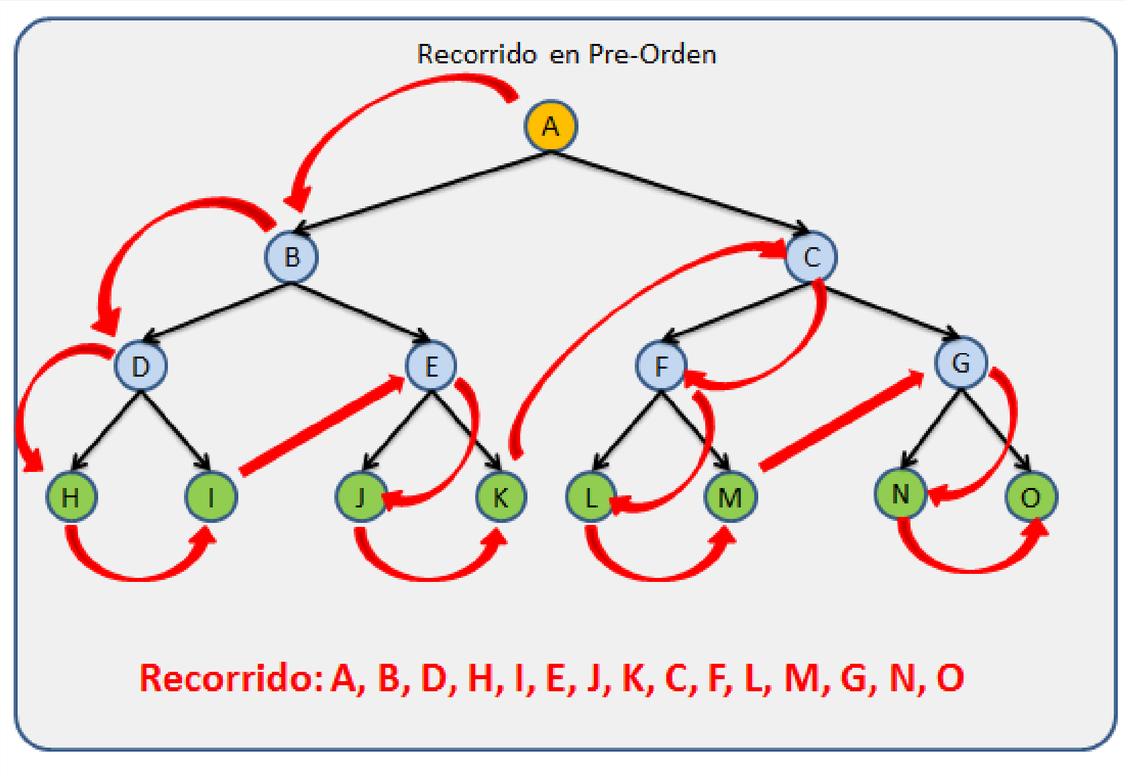
**número máximo de hijos que puede tener un Nodo**



# Recorridos de árbol binario

**Recorrido Pre-orden: El recorrido inicia en la Raíz y luego se recorre en pre-orden cada unos de los sub-árboles de izquierda a derecha.**

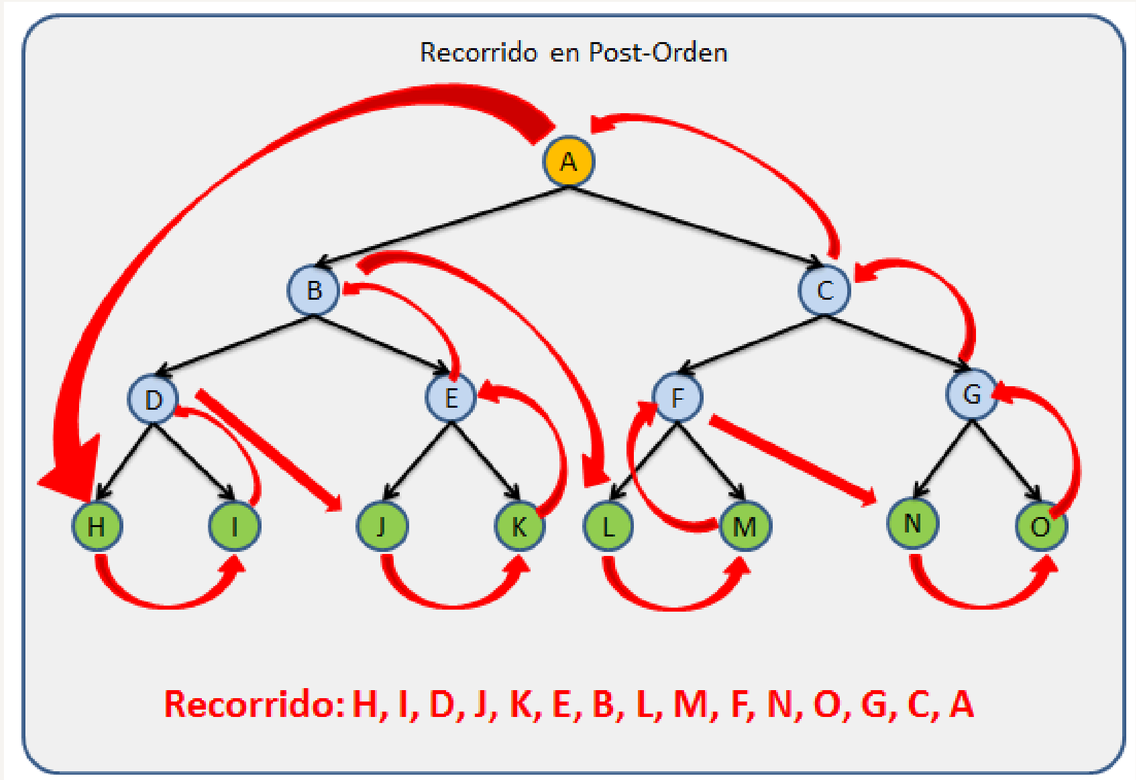
1. **Se comienza por la raíz.**
2. **Se baja hacia el hijo izquierdo de la raíz.**
3. **Se recorre recursivamente el subárbol izquierdo.**
4. **Se sube hasta el hijo derecho de la raíz.**
5. **Se recorre recursivamente el subárbol derecho.**



# Recorridos de árbol binario

## Recorrido Pos-orden: Se recorre el pos-orden cada uno de los sub-árboles y al final se recorre la raíz.

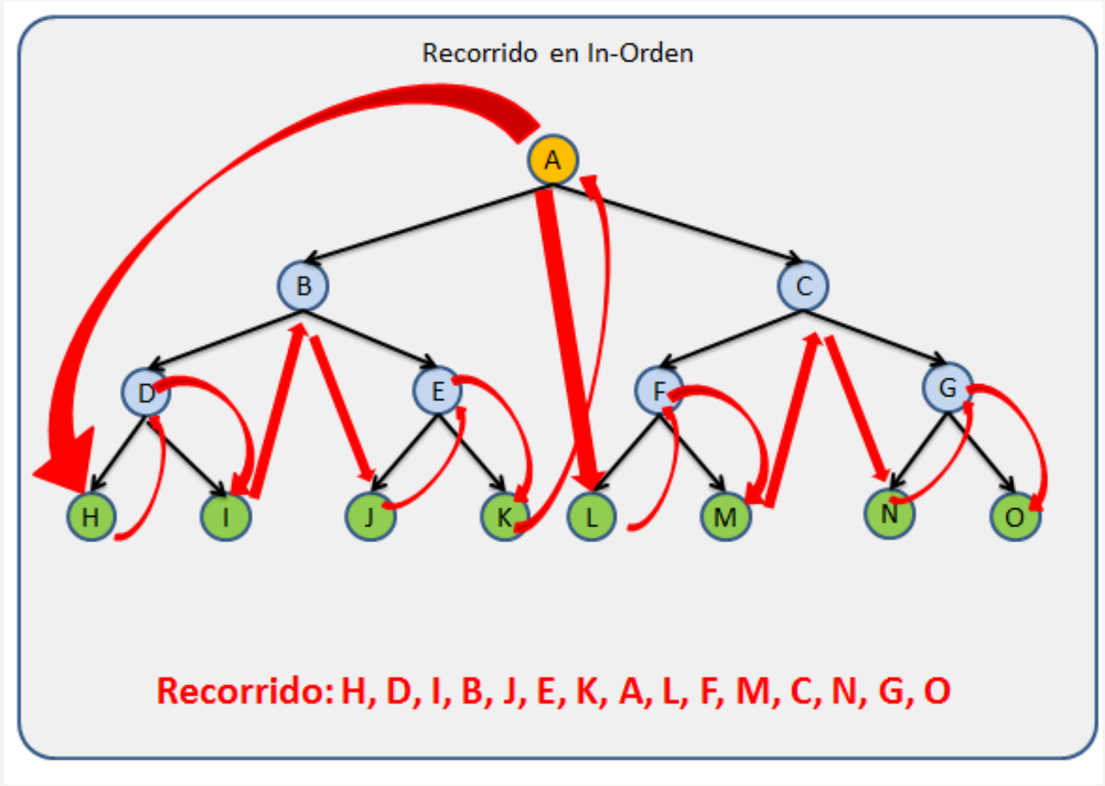
1. **Se comienza por el nodo hoja que se encuentre más a la izquierda de todos.**
2. **Se visita su nodo hermano.**
3. **Se sube hacia el padre de ambos.**
4. **Si tuviera hermanos, se visita su nodo hermano.**
5. **Se repiten los pasos 3 y 4 hasta terminar de recorrer el subárbol izquierdo.**
6. **Se recorre el subárbol derecho, comenzando por el nodo hoja que se encuentre más a la izquierda y siguiendo el mismo procedimiento que con el subárbol izquierdo**
7. **Se visita el nodo raíz.**



# Recorridos de árbol binario

**Recorrido in–orden: Se recorre en in–orden el primer sub–árbol, luego se recorre la raíz y al final se recorre en in–orden los demas sub–árboles**

1. **Se comienza por el nodo hoja que se encuentre más a la izquierda de todos.**
2. **Se sube hacia su nodo padre.**
3. **Se baja hacia el hijo derecho del nodo recorrido en el paso 2.**
4. **Se repiten los pasos 2 y 3 hasta terminar de recorrer el subárbol izquierdo.**
5. **Se visita el nodo raíz.**
6. **Se recorre el subárbol derecho de la misma manera que se recorrió el subárbol izquierdo.**



# Caso Práctico


# Conclusiones

- Comprendimos cómo se estructura y representa un árbol binario en Python.
- Se logró construir y recorrer un árbol binario utilizando listas anidadas en Python.
- Se aplicaron conceptos como: recursividad, recorrido binario, y condiciones múltiples.
- Se implementaron funciones para calcular peso y altura, facilitando el análisis estructural del árbol.
- Los árboles de decisión demostraron ser una forma efectiva de modelar elecciones binarias.