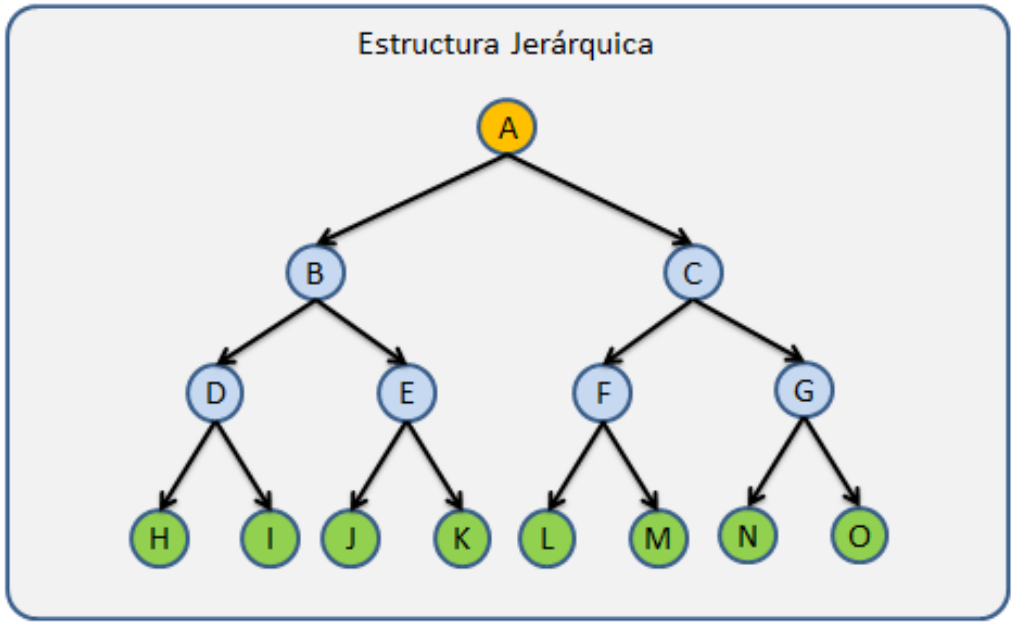
# ESTRUCTURA DE DATOS ÁRBOLES

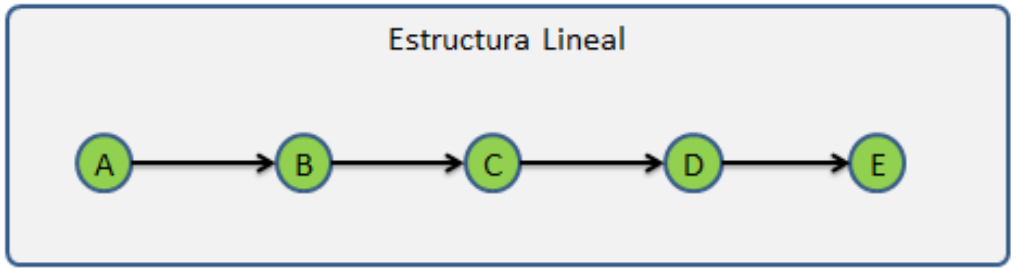
Capítulo 6

# Conceptos básicos

- Un árbol es una estructura de datos no lineal en la que cada nodo puede apuntar a uno o varios nodos.
- Se caracterizan por almacenar sus nodos de forma jerárquica.



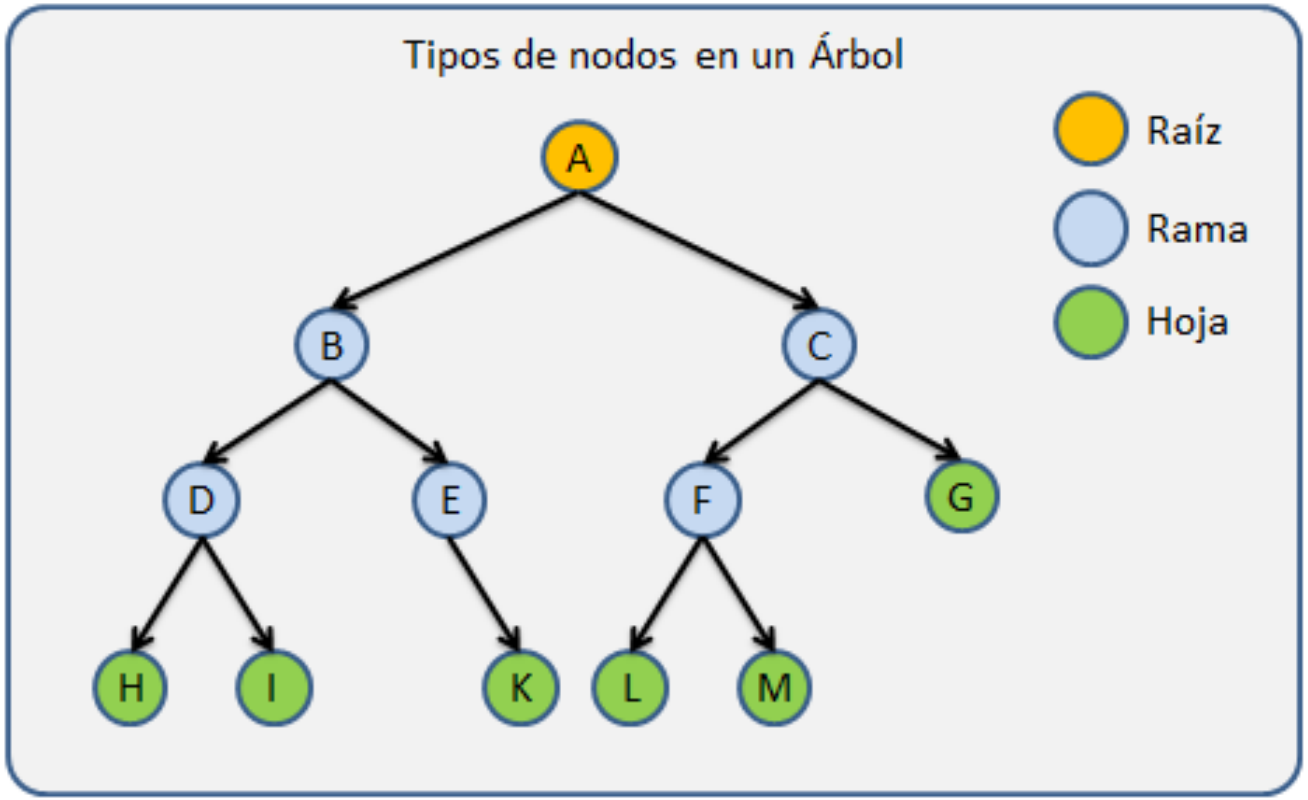

**V.S.**



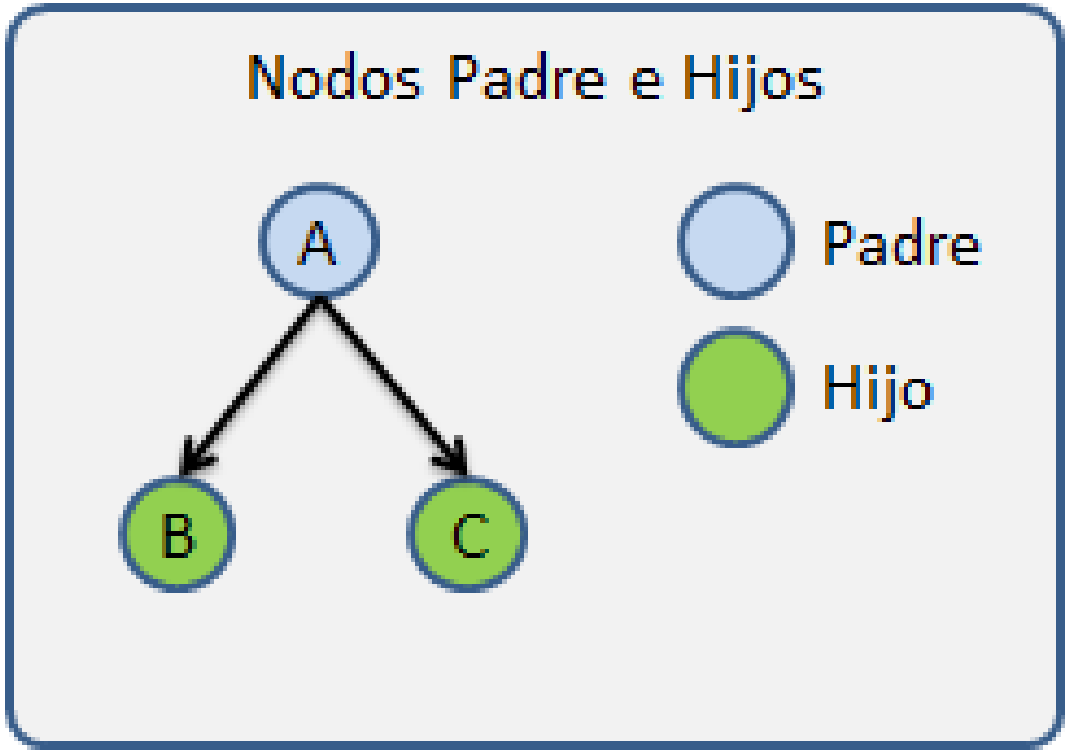
- Los árboles genealógicos y los organigramas son ejemplos comunes.
- Los árboles se emplean para analizar circuitos eléctricos y para representar la estructura de fórmulas matemáticas, así como para organizar la información de bases de datos.
- Para entender mejor la estructura de datos árbol, de que elementos esta compuesto.

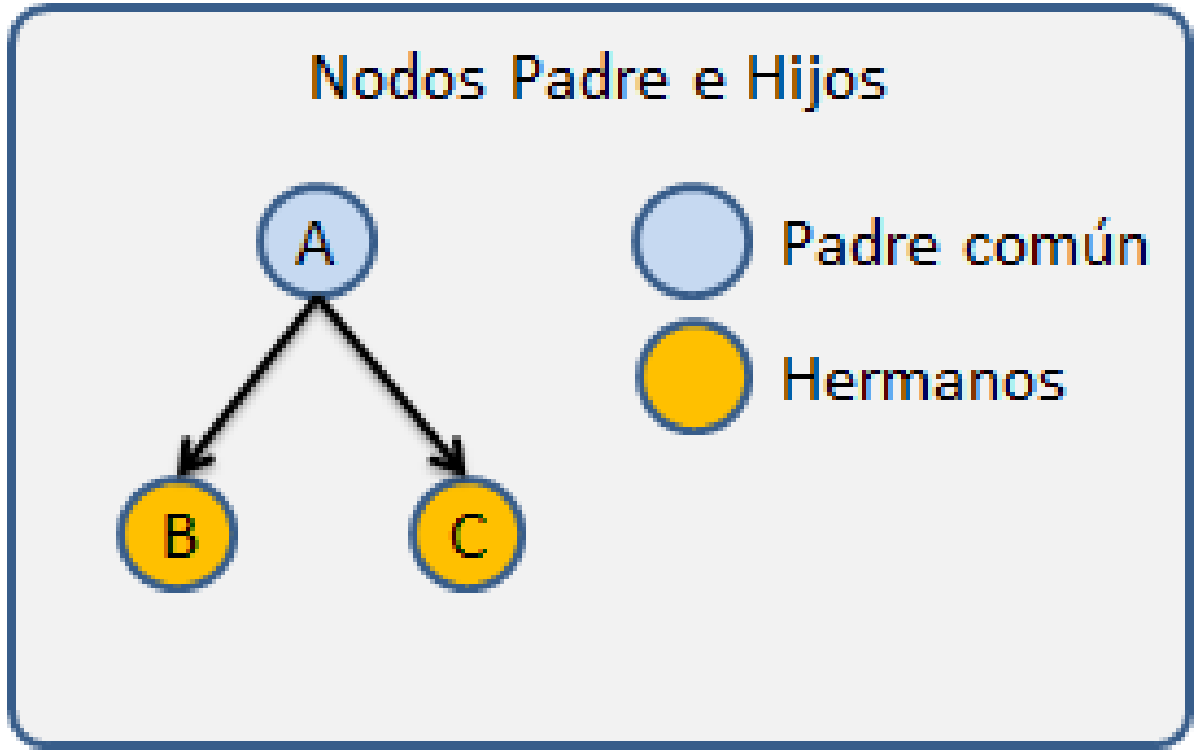
# Conceptos importantes de los árboles

- **Nodos.** Se llama nodo a cada elemento que contiene un árbol.
- **Nodo raíz.** Se refiere al primer nodo de un árbol.
- **Nodo padre.** Se utiliza este término para llamar a todos aquellos nodos que tienen al menos un hijo.
- **Nodo hijo.** Los hijos son todos aquellos nodos que tienen un padre.
- **Nodo hermano.** Los nodos hermanos son aquellos nodos que comparten a un mismo padre en común dentro de la estructura.
- **Nodo hoja.** Son todos aquellos nodos que no tienen hijos.
- ***Nodo rama.*** Estos son todos aquellos que no son la raíz y que además tiene al menos un hijo.



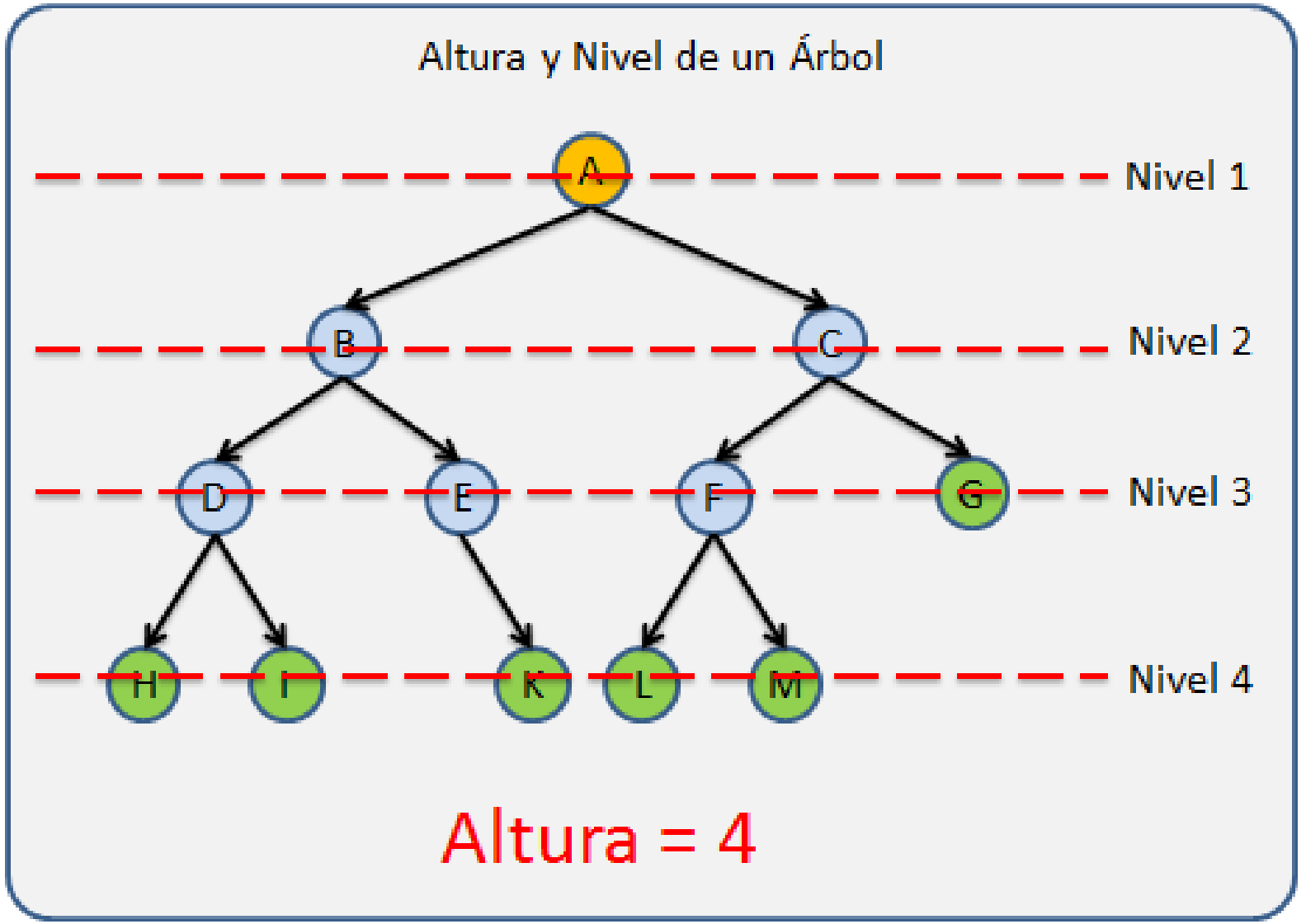

- Raiz: A
- Ramas: B, C, D, E, F
- Hojas: H, I, K, L, M


Nodos Padre e Hijos
A
B
C
Padre
Hijo

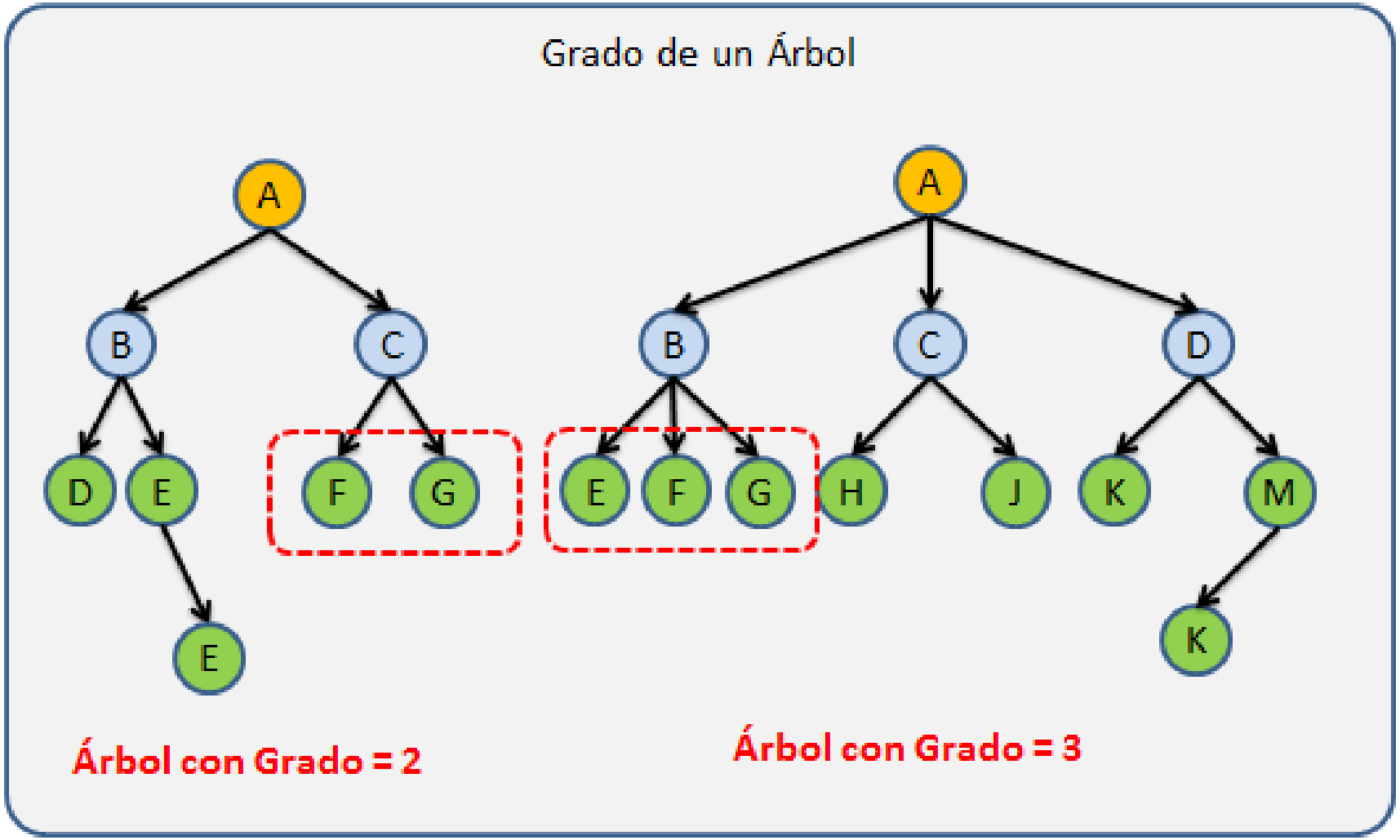

Nodos Padre e Hijos
A
B
C
Padre común
Hermanos

# Otros conceptos importantes

- **Nivel**. El nivel es cada generación del árbol. Cuando a un nodo hoja le agregamos un hijo, el nodo hoja pasa a ser un nodo rama, pero además el árbol crece una generación por lo que el árbol tiene un nivel más.
  - El árbol vacío tiene 0 niveles
  - El nivel de la raíz es 1
  - El nivel de otro nodo es incrementando en 1 el nivel de su padre.
- **Altura.** Es el numero máximo de niveles de un árbol

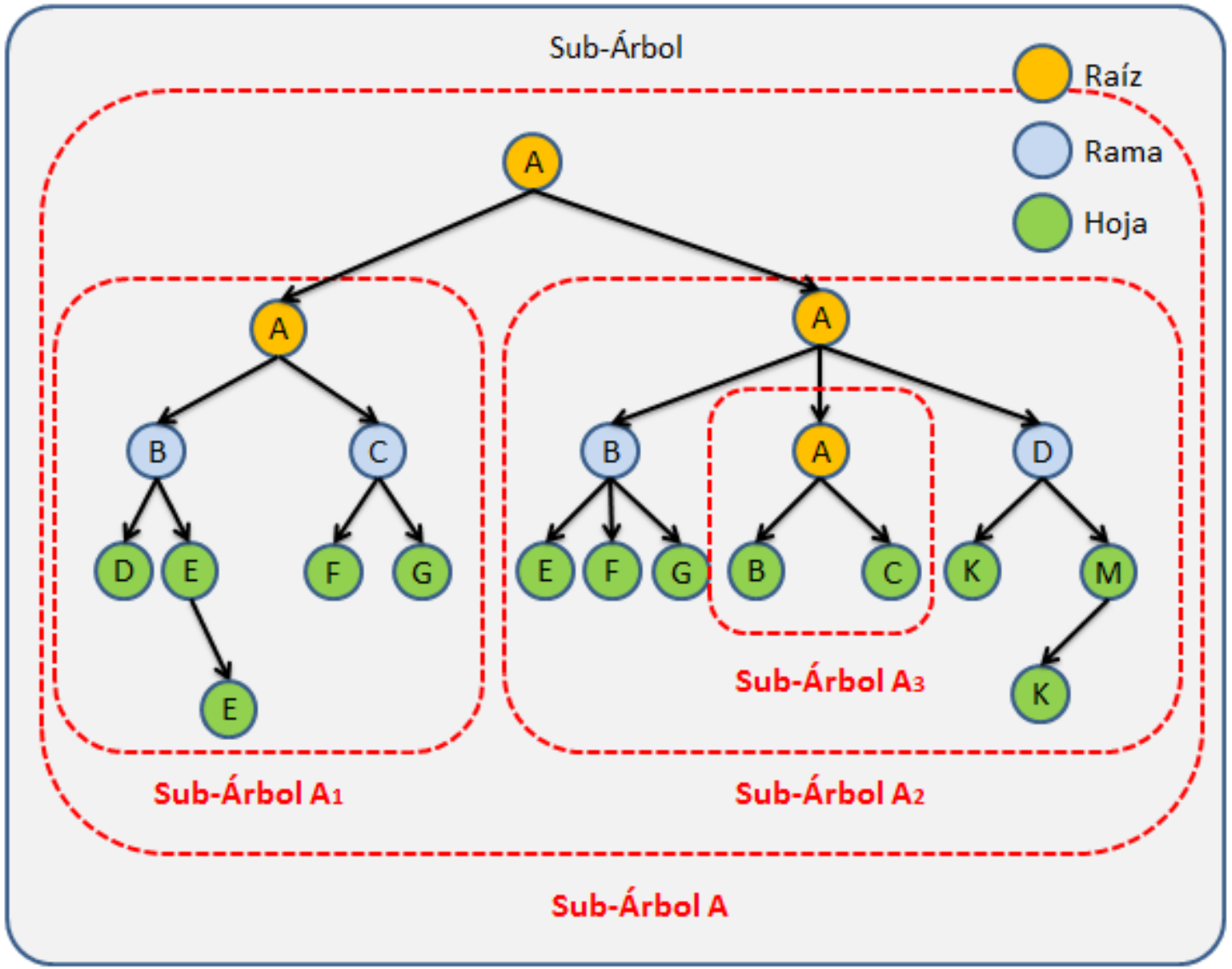

Altura y Nivel de un Árbol
A
B
C
D
E
F
G
H
I
K
L
M
Nivel 1
Nivel 2
Nivel 3
Nivel 4
Altura = 4

- **Grado.** El grado se refiere al número mayor de hijos que tiene alguno de los nodos del árbol.



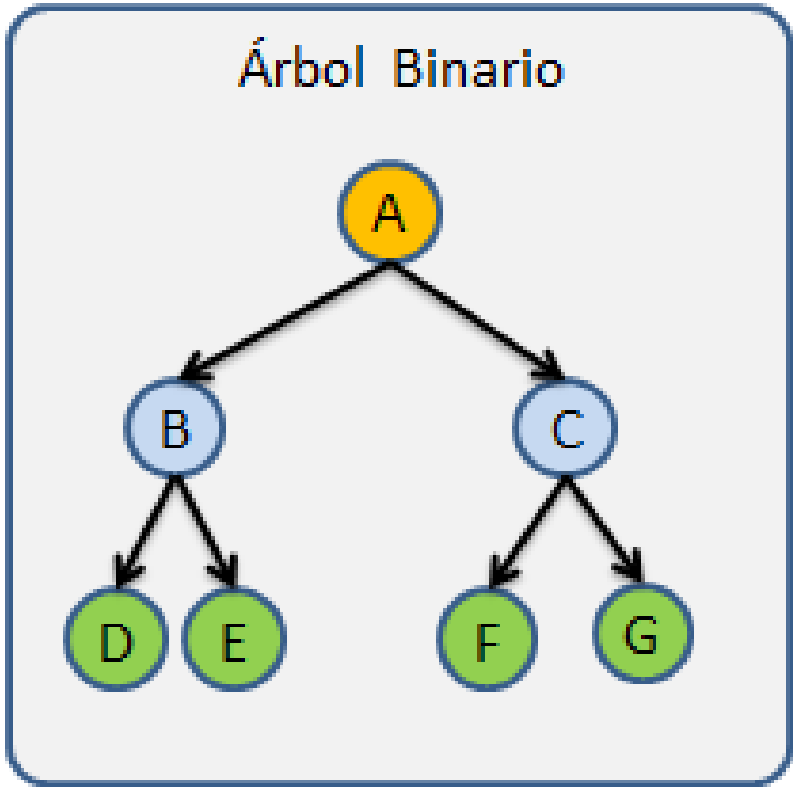
# Sub árbol

- Un sub árbol es todo árbol generado a partir de una sección determinada de árbol, por lo que podemos decir que un árbol es un nodo Raíz con N Sub Árboles.



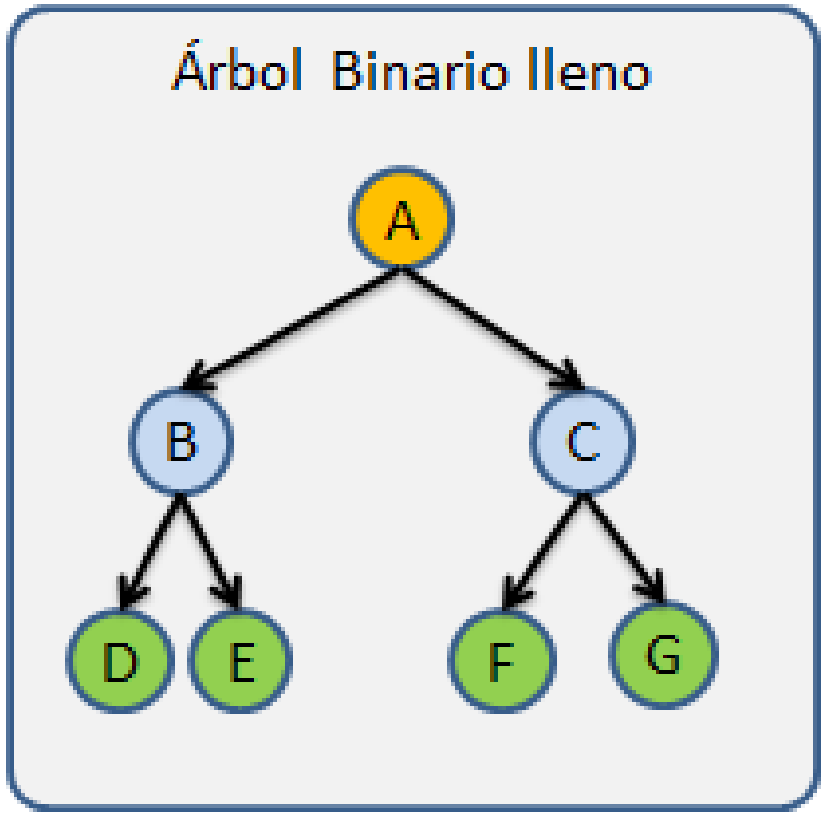
# Árboles binarios

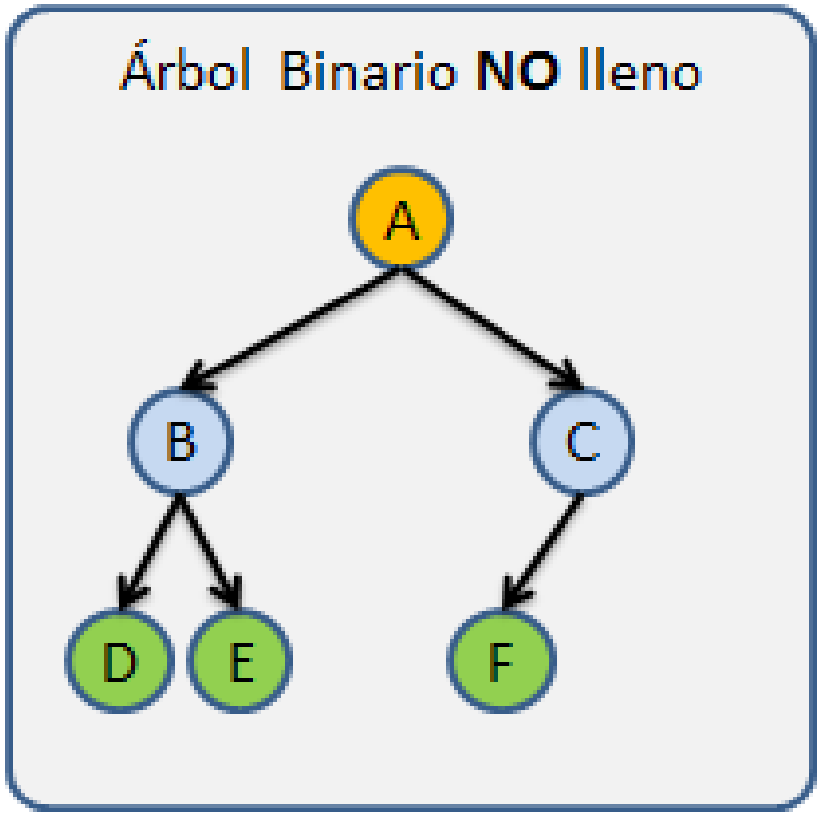
- Un árbol binario está formado por un conjunto de elementos donde cada elemento tiene un sub árbol izquierdo y un sub árbol derecho que a su vez son árboles binarios.
- Esta estructura se caracteriza porque cada nodo solo puede tener máximo 2 hijos, dicho de otra manera es un árbol de grado 2.



# Árboles binarios

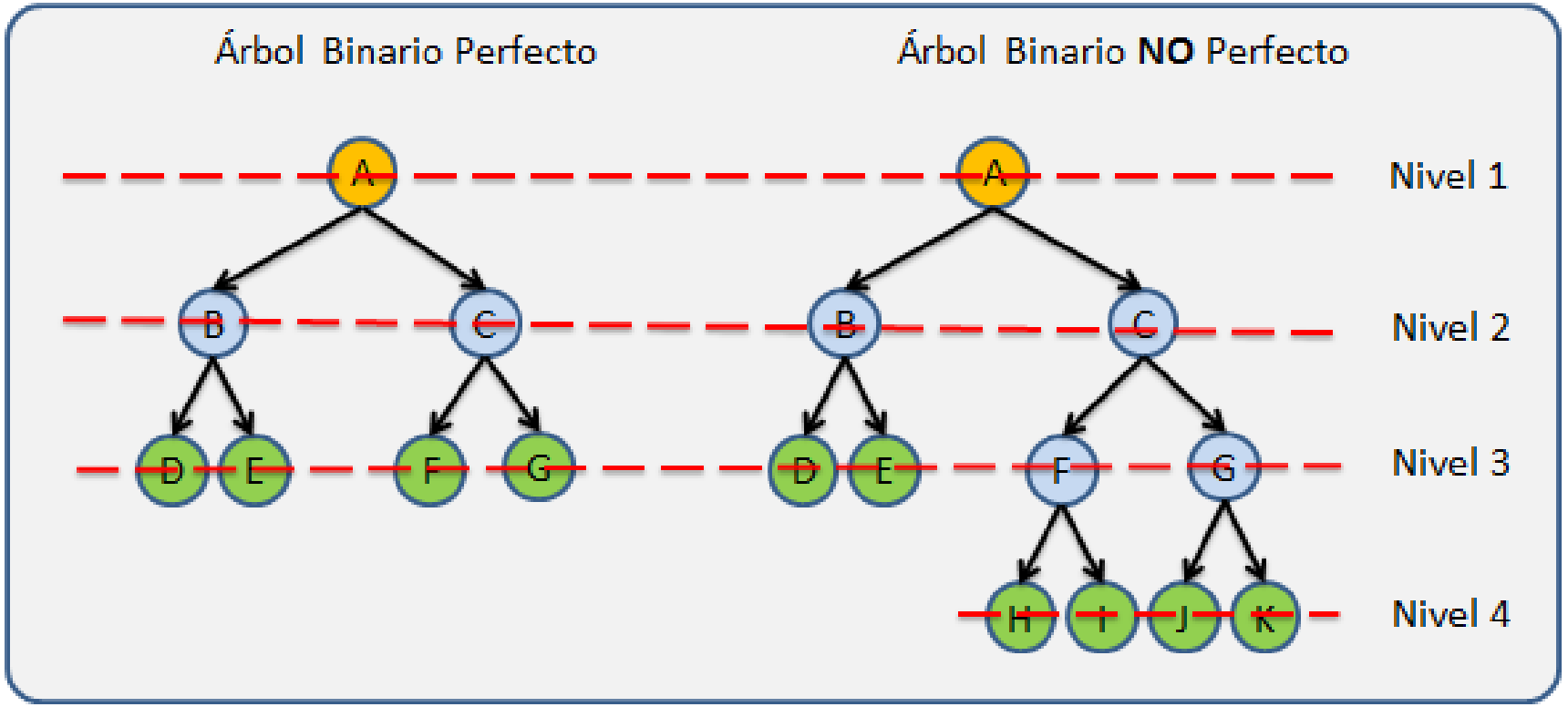
- **Árbol binario lleno:** Es aquel donde todos los nodos tienen cero o 2 hijos con excepción de la Raíz.



# Árboles binarios

- **Árbol binario perfecto.** Es un árbol lleno donde todas las hojas están en el mismo nivel.



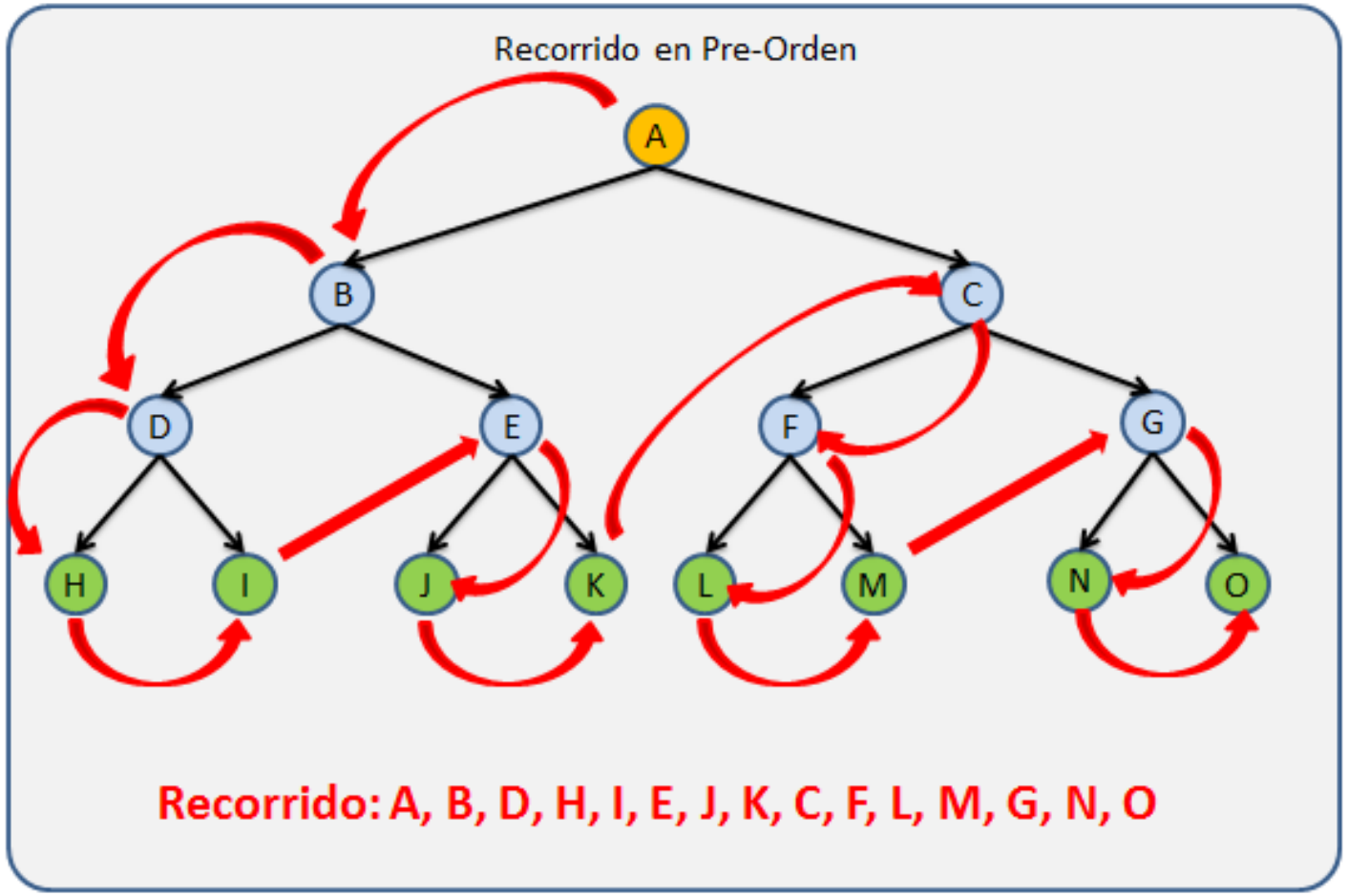
# Recorridos de árboles binarios

- Los recorridos son algoritmos que nos permiten recorrer un árbol en un orden específico. Los recorridos nos pueden ayudar a encontrar un nodo en un árbol, o buscar una posición determinada para insertar o eliminar un nodo.

- Búsqueda no informada. Es el recorrido que se realiza por todo el árbol sin tener un orden específico.
- Búsqueda en profundidad. Recorrido en pre-orden, entre-orden (in-orden), post-orden.

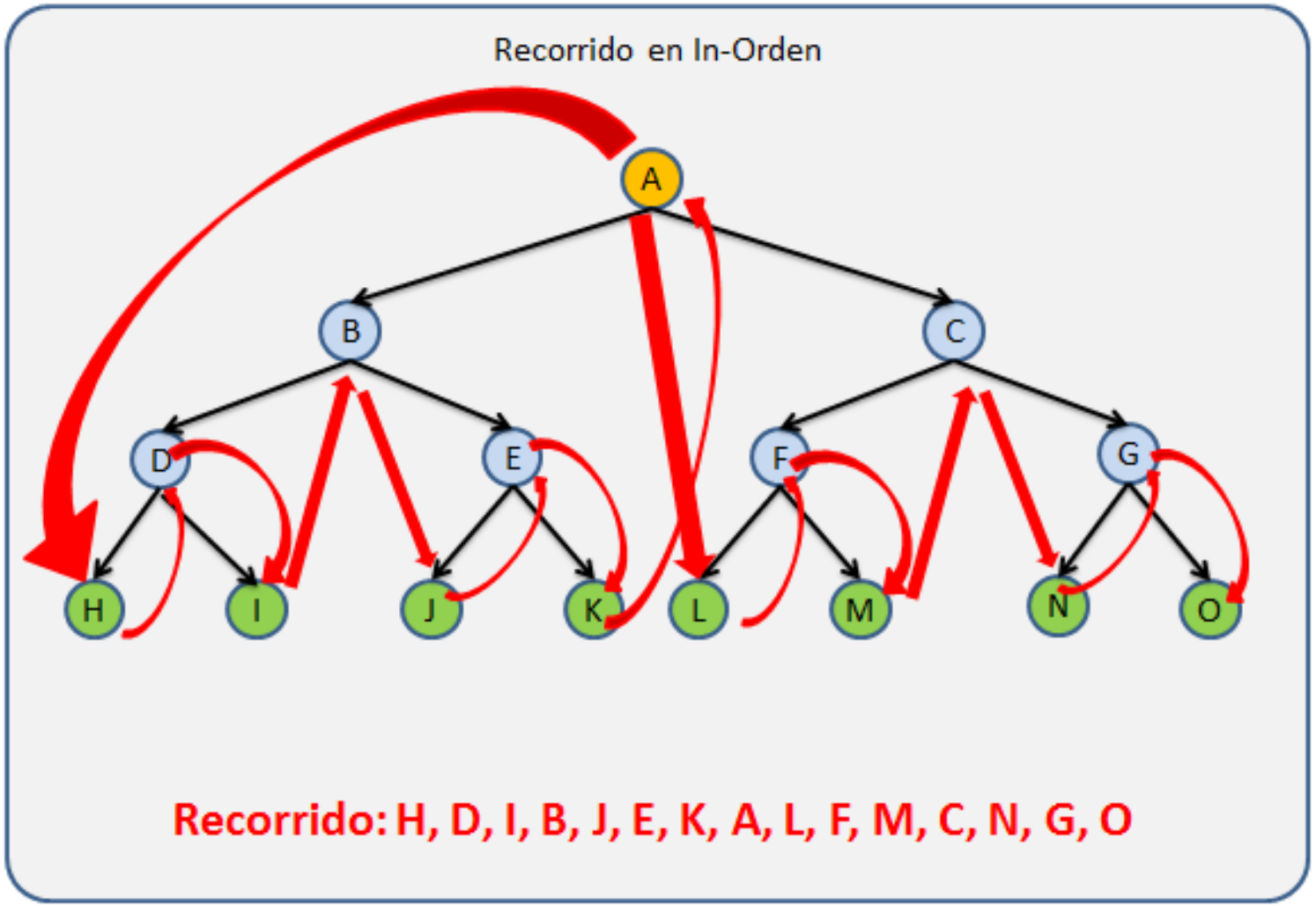
# Recorrido en pre-orden

- El recorrido inicia en la Raíz, luego Sub Árbol Izquierdo y Sub Árbol Derecho.



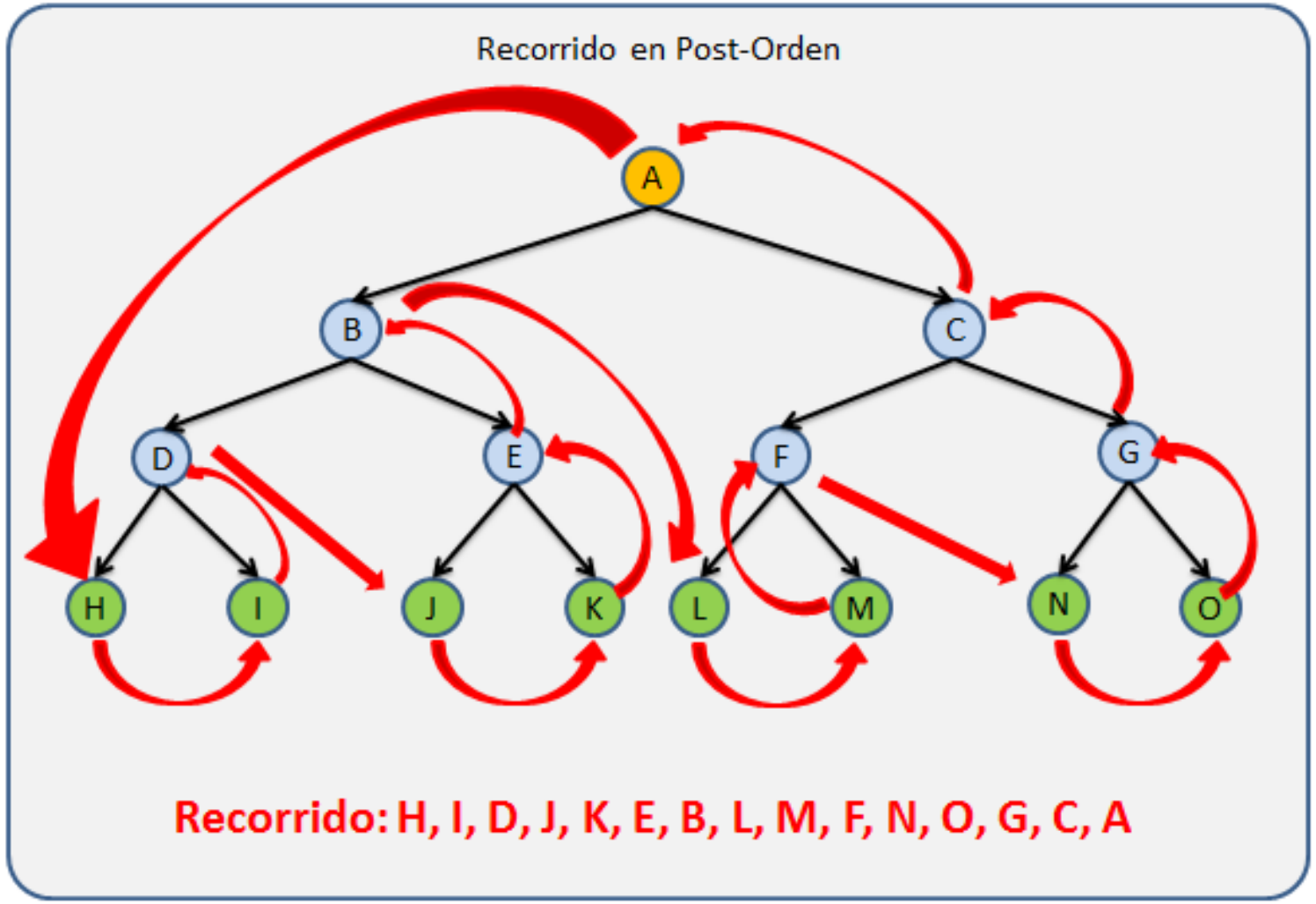
# Recorrido en entre-orden (in-orden)

- El recorrido inicia en el Sub Árbol Izquierdo, luego Raíz y Sub Árbol Derecho.

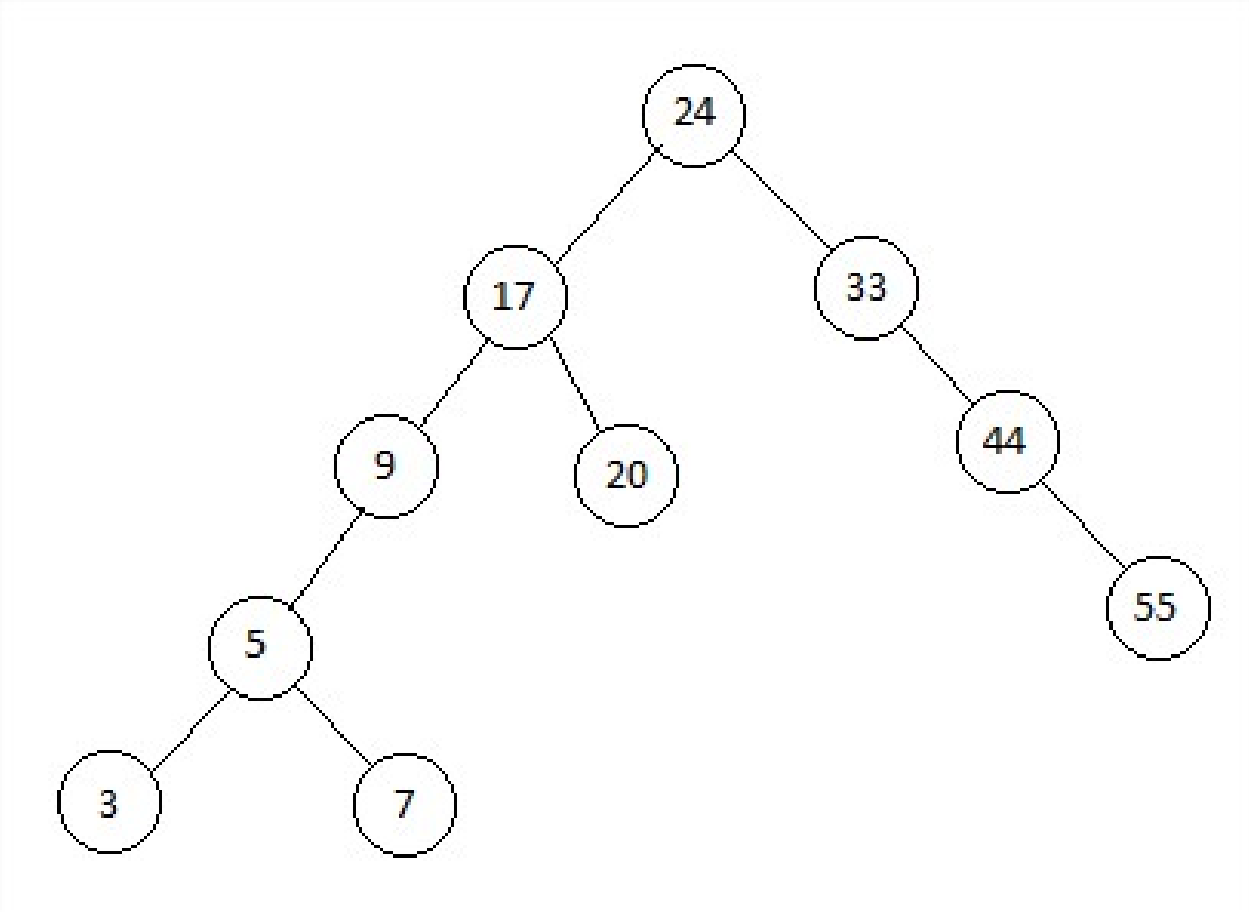


# Recorrido en Post-orden

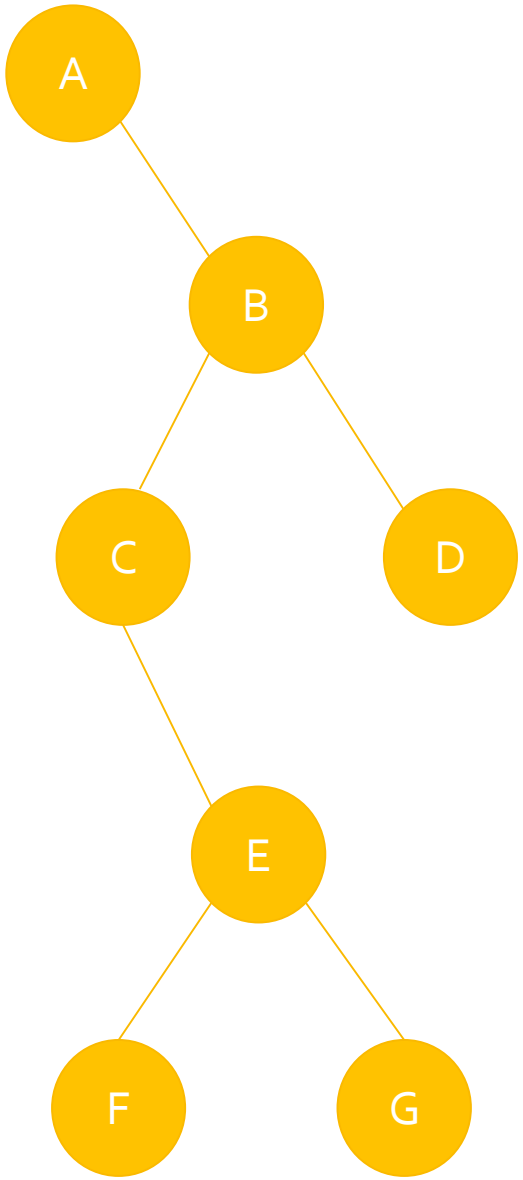
- El recorrido inicia en el Sub Árbol Izquierdo, luego Sub Árbol Derecho y Raíz.



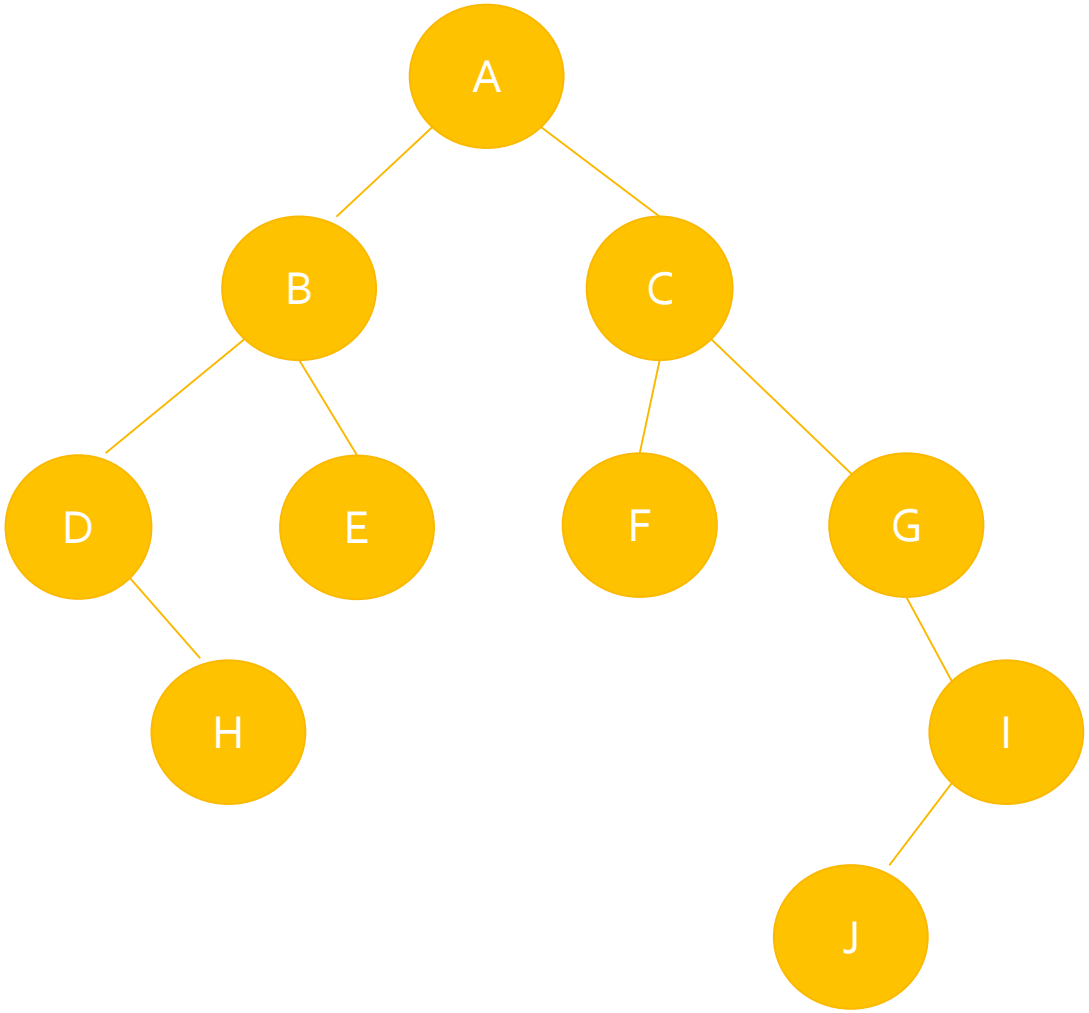
# Obtener los recorridos en pre-orden, entre-orden y post-orden



# Obtener los recorridos en pre-orden, entre-orden y post-orden

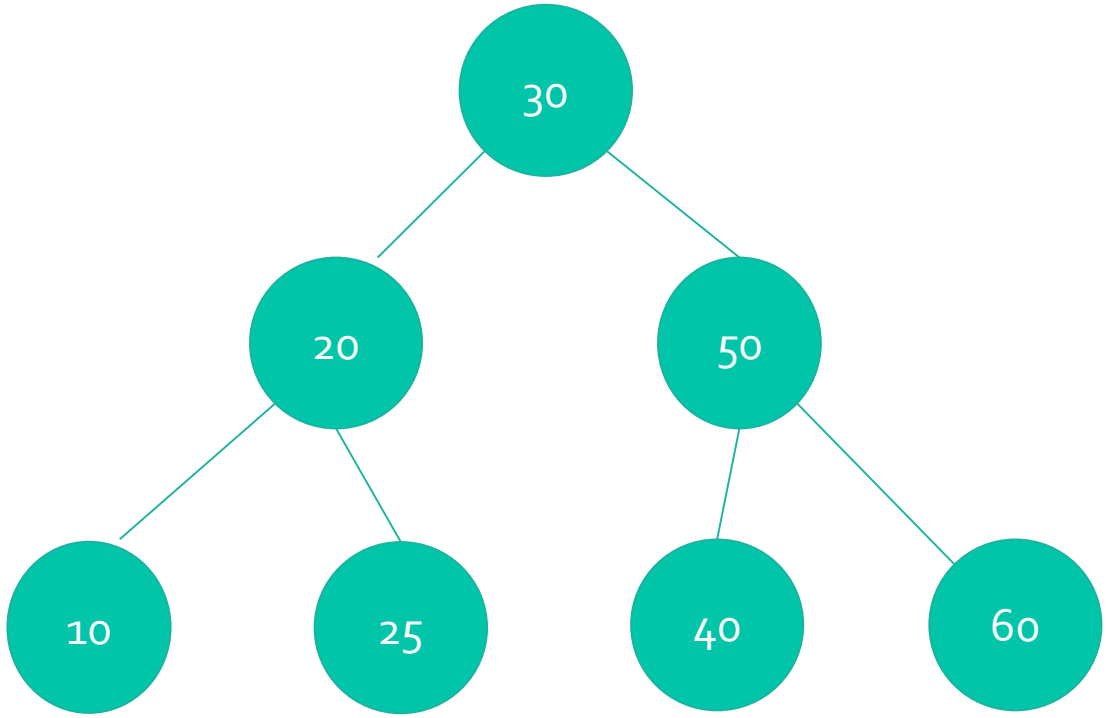


# Obtener los recorridos en pre-orden, entre-orden y post-orden



# Árbol lexicográfico

- Es un árbol en el cual todos los elementos a la derecha son mayores y los de la izquierda menores a la Raíz.



# Ejemplos de árbol lexicográficos

- Crear el árbol lexicográfico para las siguientes listas de elementos:
  - A) 50, 30, 80, 10, 60, 100, 20, 15
  - B) 50, 40, 80, 30, 60, 100, 10, 55, 20

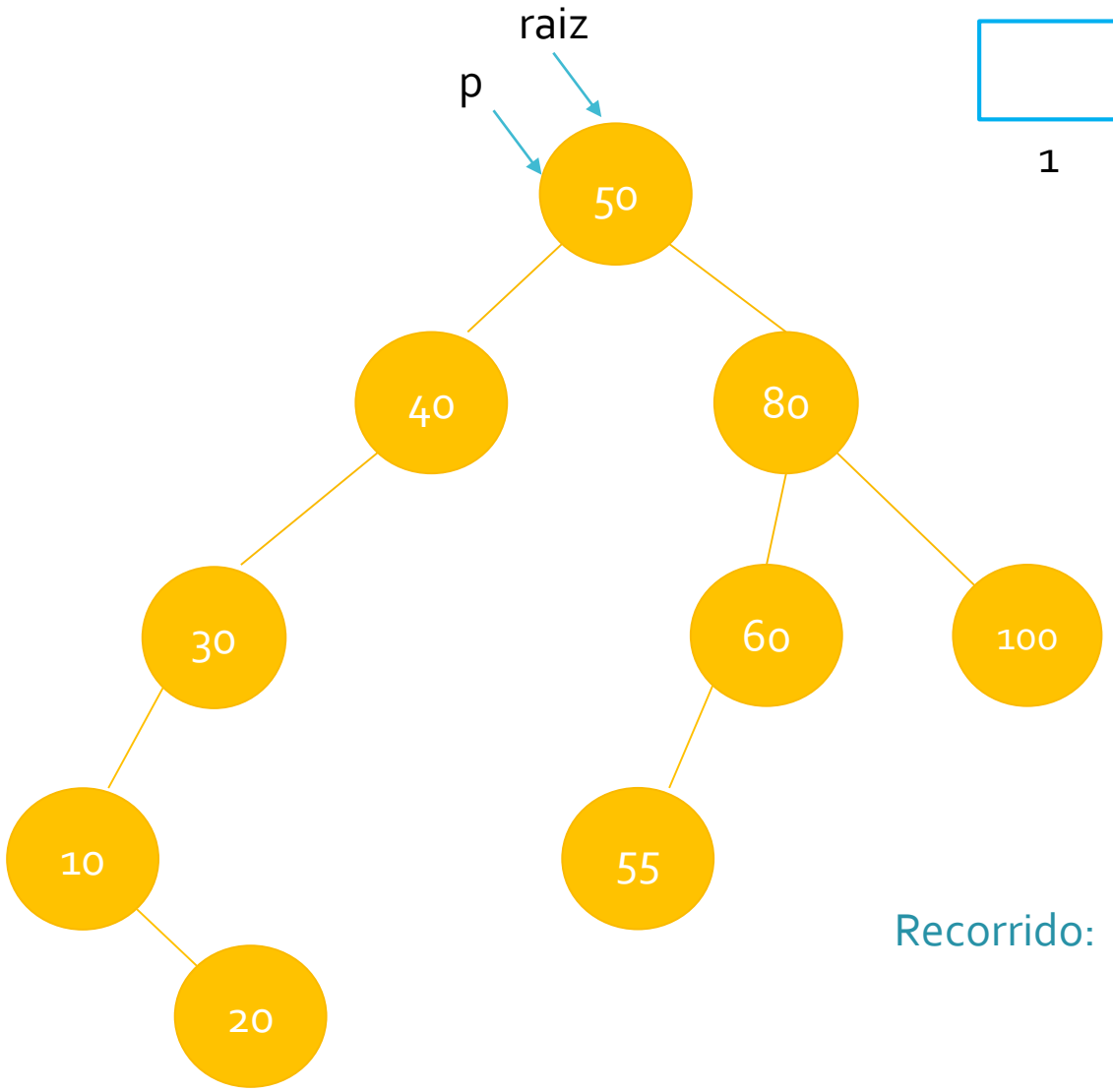
# Implementemos árboles binarios en C#

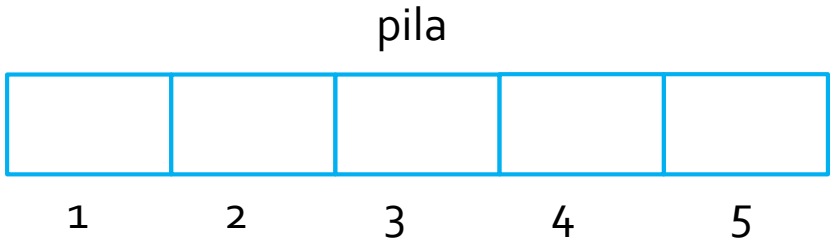
- **Crear árbol lexicográfico**
- Verificamos si el árbol se encuentra vacío (raíz = null)
- Sino, recorremos el árbol; si el elemento a insertar es menor a la raíz  recorremos el subárbol izquierdo, sino recorremos el subárbol derecho.
- Luego de recorrer el árbol, si el elemento a insertar es menor a su padre se enlaza a su padre por el enlace izquierdo, si es mayor se enlaza a su padre por el enlace derecho.

# Implementemos árboles binarios en C#

- **Pre-orden recursivo**
- Si el árbol no está vacío
- Mostramos la raíz
- Ingresamos como parámetro al método recursivo el subárbol izquierdo
- Ingresamos como parámetro al método recursivo el subárbol derecho

**Recorrido en pre-orden no recursivo.** Usaremos una pila para insertar la raíz de los subárbol izquierdo.




**TOPE**

0

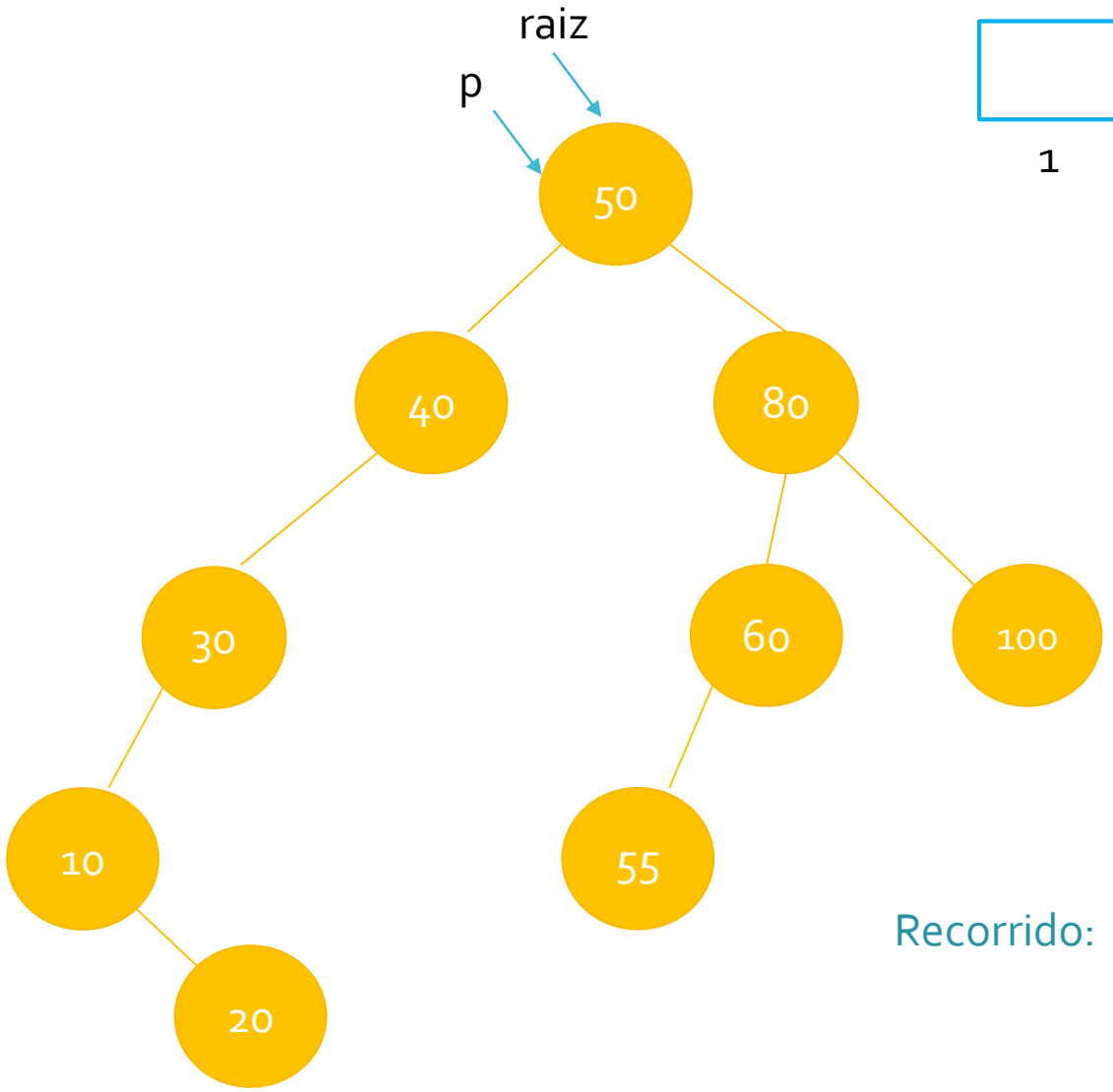
Recorrido:

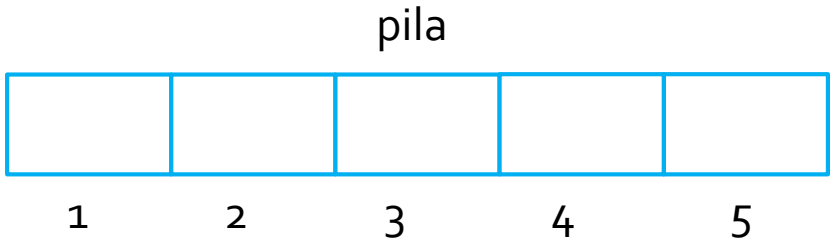
1. Hacemos que p "apunte" a la raíz.
2. Mientras p <> null o tope > 0
3. Mostramos p.info
4. Incrementamos tope en 1
5. Insertamos p en la pila
6. Recorremos el subárbol izquierdo
7. Cuando p sea null; p = tope de pila en su enlace derecho
7. Tope disminuye en 1.
8. Cuando p = null o tope <= 0 hemos recorrido todo el árbol

# Implementemos árboles binarios en C#

- **Entre-orden recursivo**
- Si el árbol no está vacío
- Ingresamos como parámetro al método recursivo el subárbol izquierdo.
- Mostramos la raíz.
- Ingresamos como parámetro al método recursivo el subárbol derecho.

**Recorrido en entre-orden no recursivo.** Usaremos una pila para insertar la raíz de los subárbol izquierdo.




**TOPE**

0

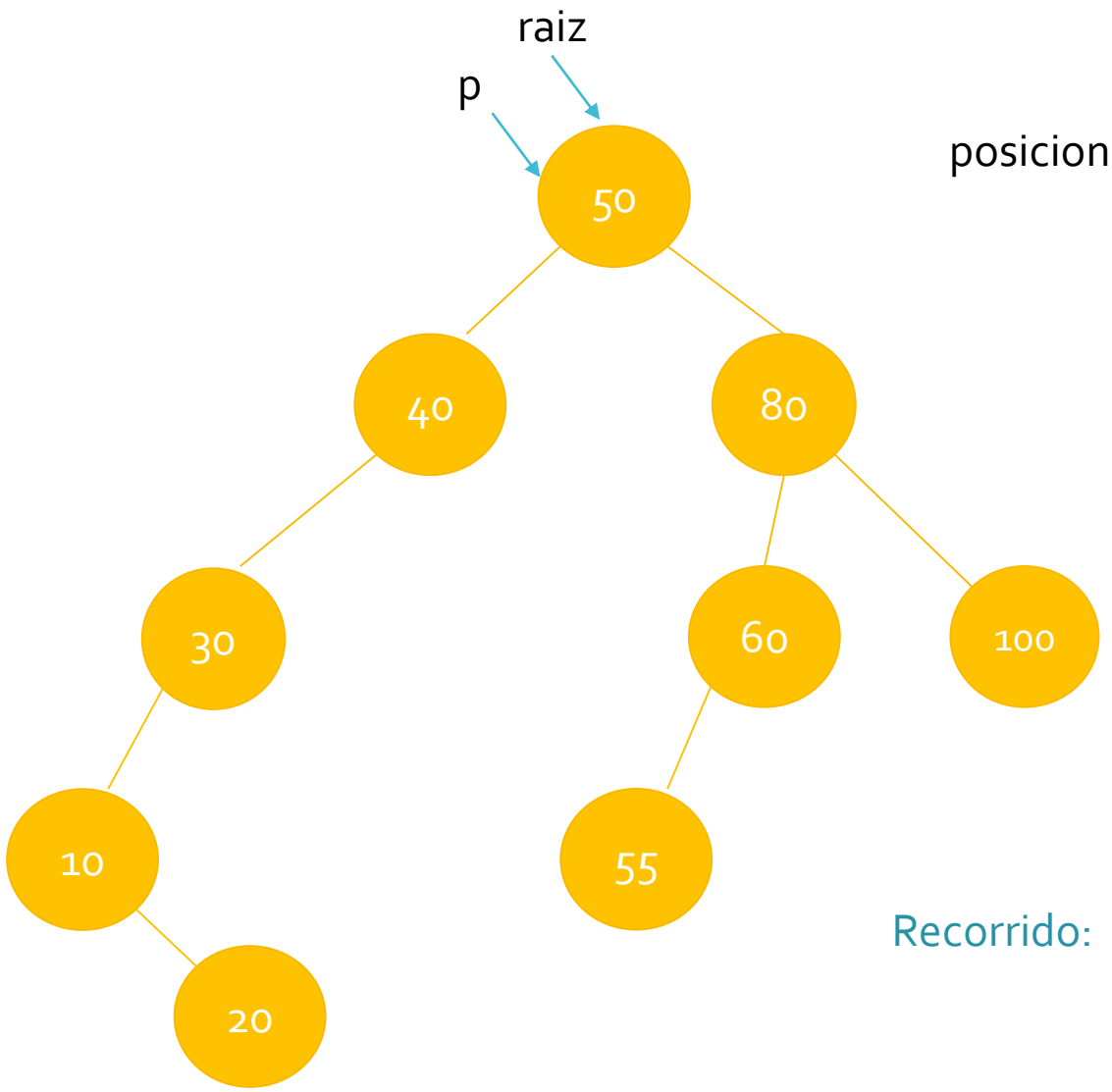
1. Hacemos que p "apunte" a la raíz.
2. Mientras p <> null o tope != 0
3. Incrementamos tope en 1
4. Insertamos p en la pila
5. Recorremos el subárbol izquierdo
6. Cuando p sea null; Mostramos pila en su posición tope
7. p = tope de pila en su enlace derecho
8. Tope disminuye en 1.

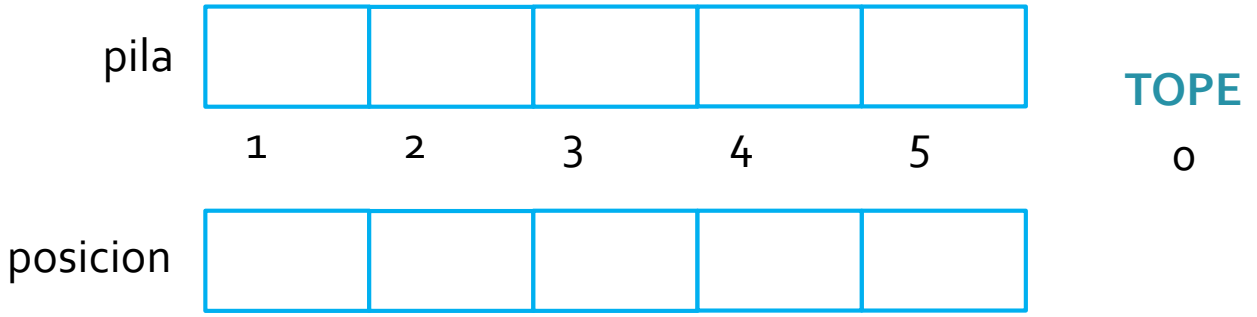
Recorrido:

# Implementemos árboles binarios en C#

- **Post-orden recursivo**
- Si el árbol no está vacío
- Ingresamos como parámetro al método recursivo el subárbol izquierdo.
- Ingresamos como parámetro al método recursivo el subárbol derecho.
- Mostramos la raíz.

**Recorrido en Post-orden no recursivo.** Usaremos una pila para insertar la raíz de los subárbol izquierdo.




**TOPE**

0

Recorrido:

1. Utilizamos un array para guardar la posición de los nodos
2. Hacemos que p "apunte" a la raíz.
3. Mientras p <> null o tope != 0
4. Incrementamos tope en 1
5. Insertamos p en la pila
6. Insertamos **0** en el vector posición en tope
7. Recorremos el subárbol izquierdo
8. Cuando p sea null; si posición en tope = 0, cambiamos el valor de posición tope en 1 y recorremos el árbol derecho.
9. Cuando p sea null; si posición en tope = 1; mostramos nodo
10. Tope disminuye en 1.

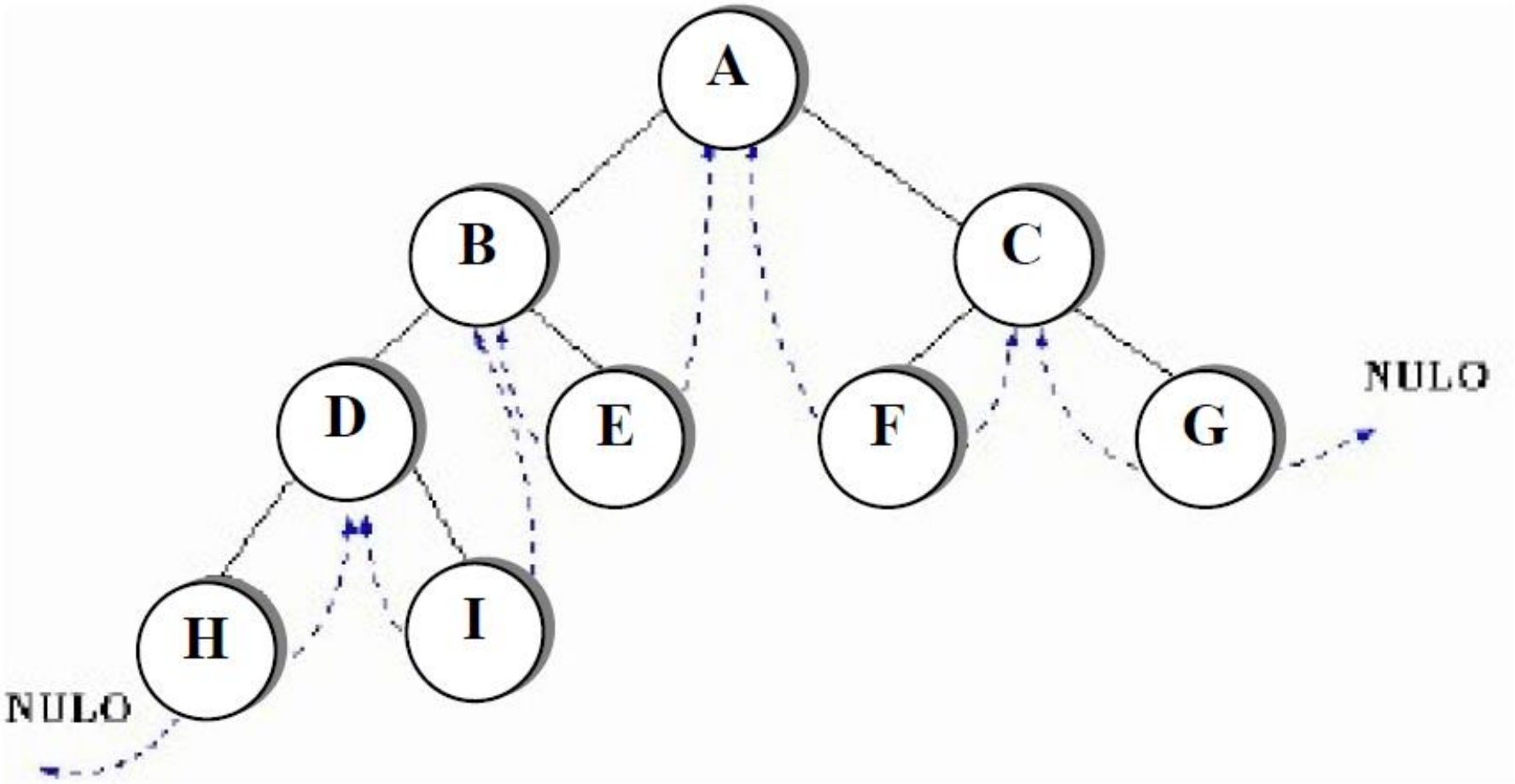
# Árboles binarios hilvanados

- Al observar la representación de un árbol binario se puede observar que existen muchos enlaces nulos.
- Existen mas enlaces nulos que punteros con valores reales.
- Como el espacio de memoria ocupado por los enlaces nulos es el mismo que el ocupado por los no nulos, resulta conveniente utilizar estos enlaces nulos para almacenar información de interés para la manipulación del árbol binario.
- Una forma de utilizar los enlaces nulos es sustituirlos por enlaces a otros nodos del árbol.

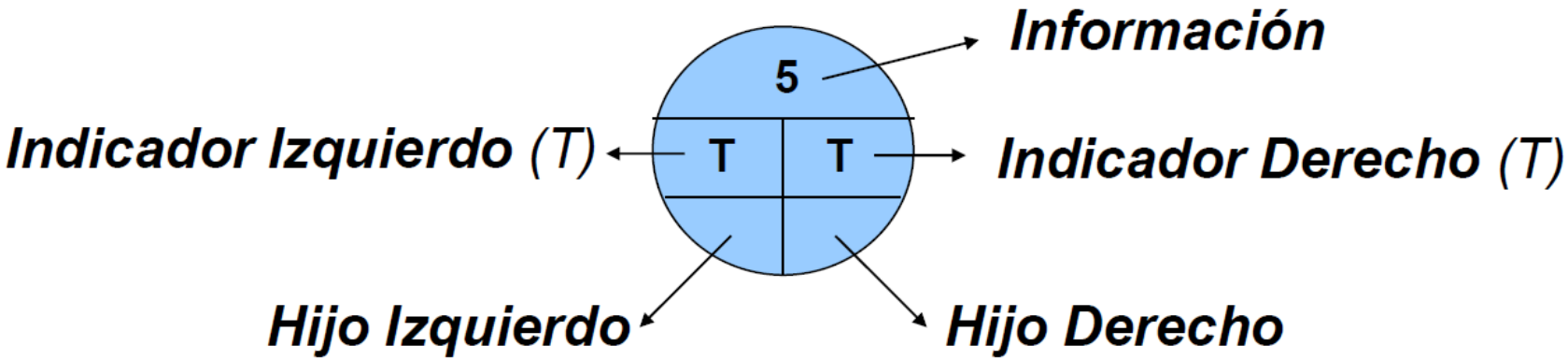
# Árboles binarios hilvanados

- Un árbol hilvanado es un árbol binario en el que cada hijo izquierdo de valor nulo es sustituido por un enlace o hilván al nodo que le antecede en orden simétrico (excepto el primer nodo en orden simétrico) y cada hijo derecho de valor nulo es sustituido por un enlace o hilván al nodo que le sigue en el recorrido en orden simétrico (excepto el último nodo en orden simétrico).
- Estos árboles se generan a partir de un recorrido en entre-orden donde se tendrán definidos otros dos enlaces tanto izquierda como derecha llamados hilvanes.
- Para hilvanar se necesita trabajar con enlaces originales y a través de estos definir o establecer los hilvanes.

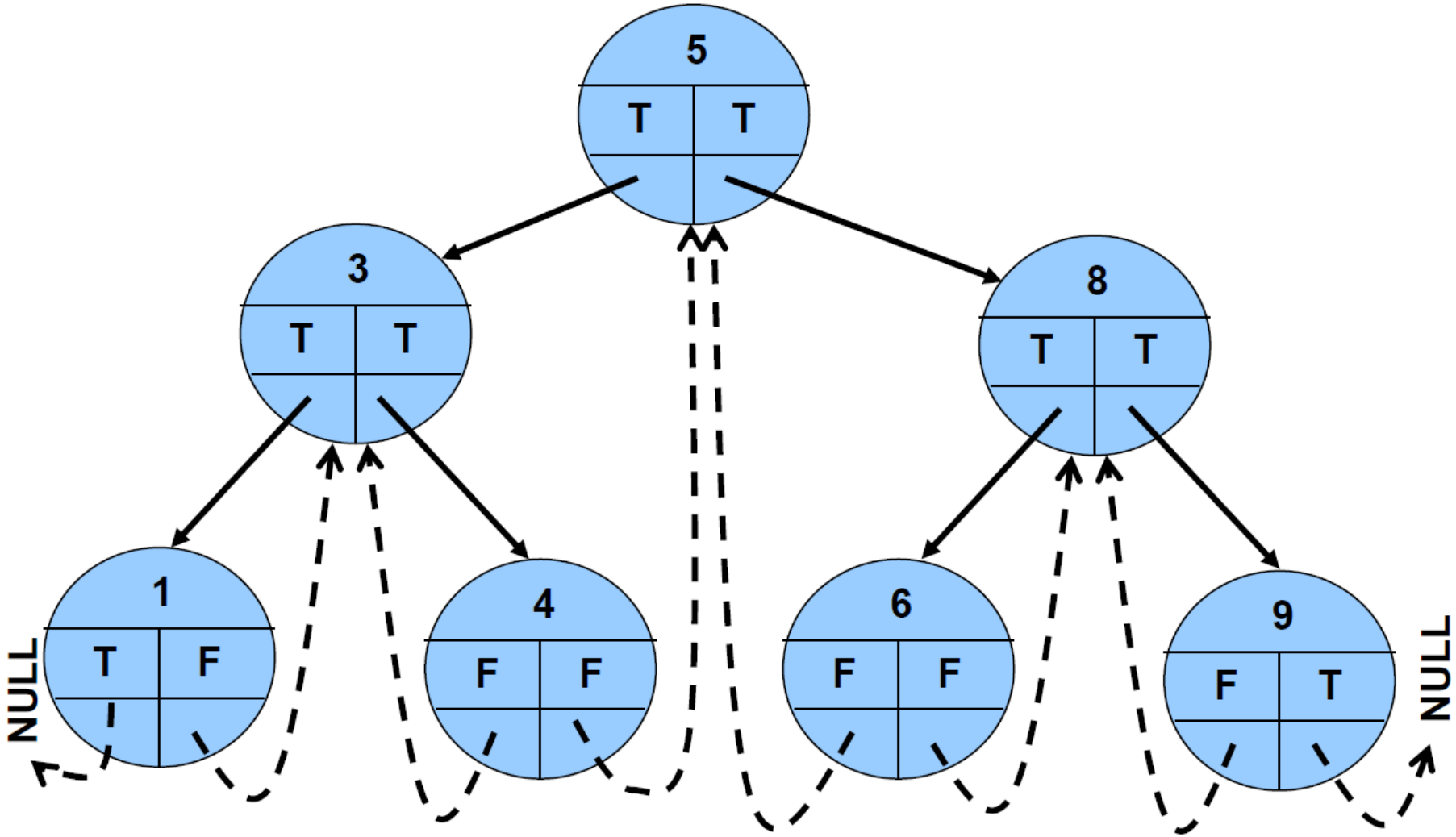
# Árboles binarios hilvanados



# Árboles binarios hilvanados



# Árboles binarios hilvanados

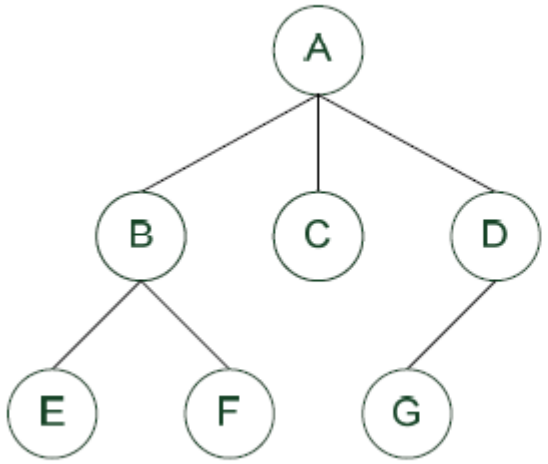


# Árboles generales

- También son llamados n-arios o multicamino, son árboles cuyo grado es mayor que dos.
- Es la estructura utilizada para representar organizaciones jerárquicas donde cada elemento tiene un número variable de hijos.
- Ejemplos:
- Representación de un sistema de ficheros, en el que cada directorio está enlazado con sus descendientes, ficheros o subdirectorios.
- Representación de un árbol genealógico en el que cada persona se enlaza con sus descendientes.

# Implementaciones de árboles generales

- Cada nodo debería tener un número indeterminado de enlaces que apunten a cada uno de sus hijos.
- ¿Cuántos enlaces reservamos?
- Se pueden implementar como un array con indicaciones al padre de cada nodo.

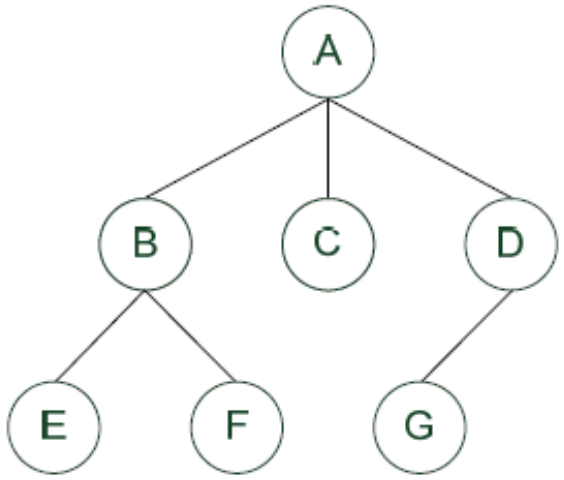


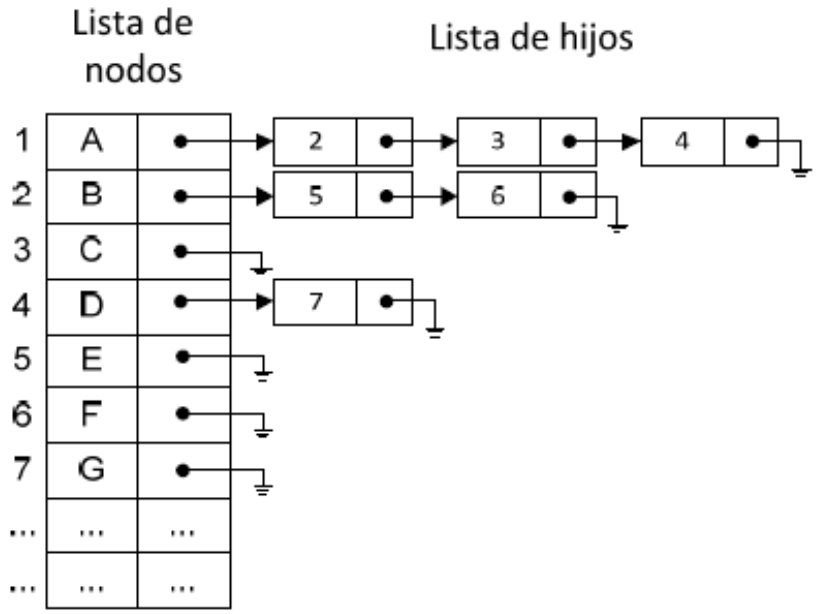

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|
| Nodos | A | B | C | D | E | F | G | ... |
| Padres | 0 | 1 | 1 | 1 | 2 | 2 | 4 | ... |

# Implementaciones de árboles generales

## Implementación mediante una lista de hijos.

- En un índice aparecen todos los nodos.
- Para cada nodo existe una lista de hijos.

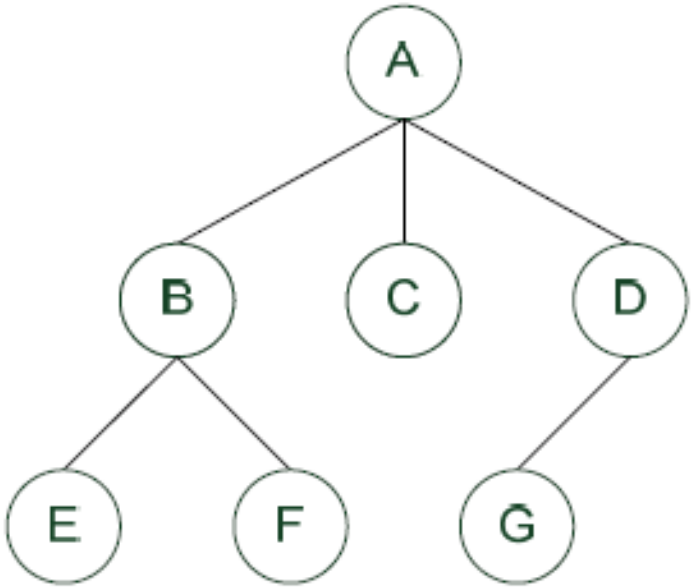


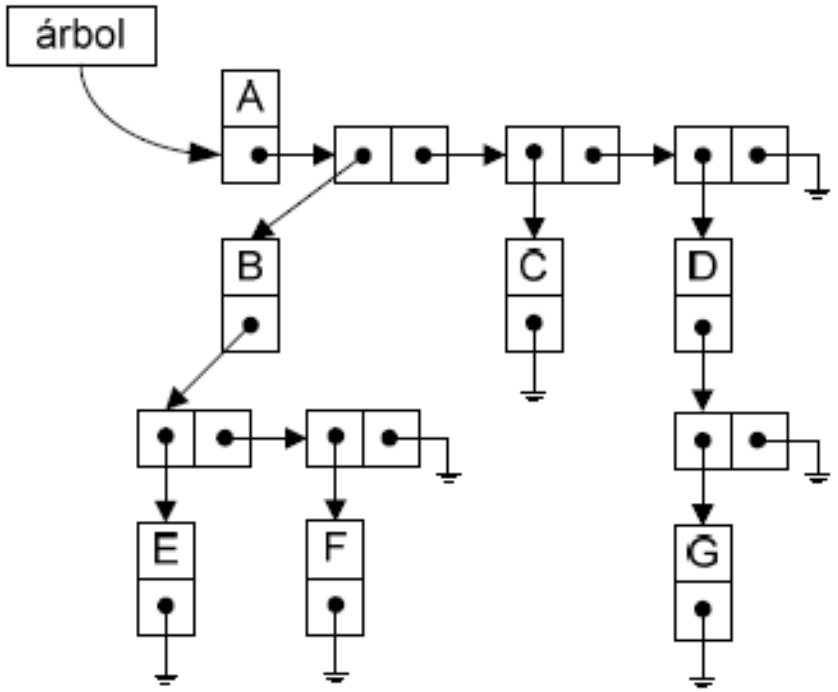



# Implementaciones de árboles generales

## Implementación mediante una de lista de listas

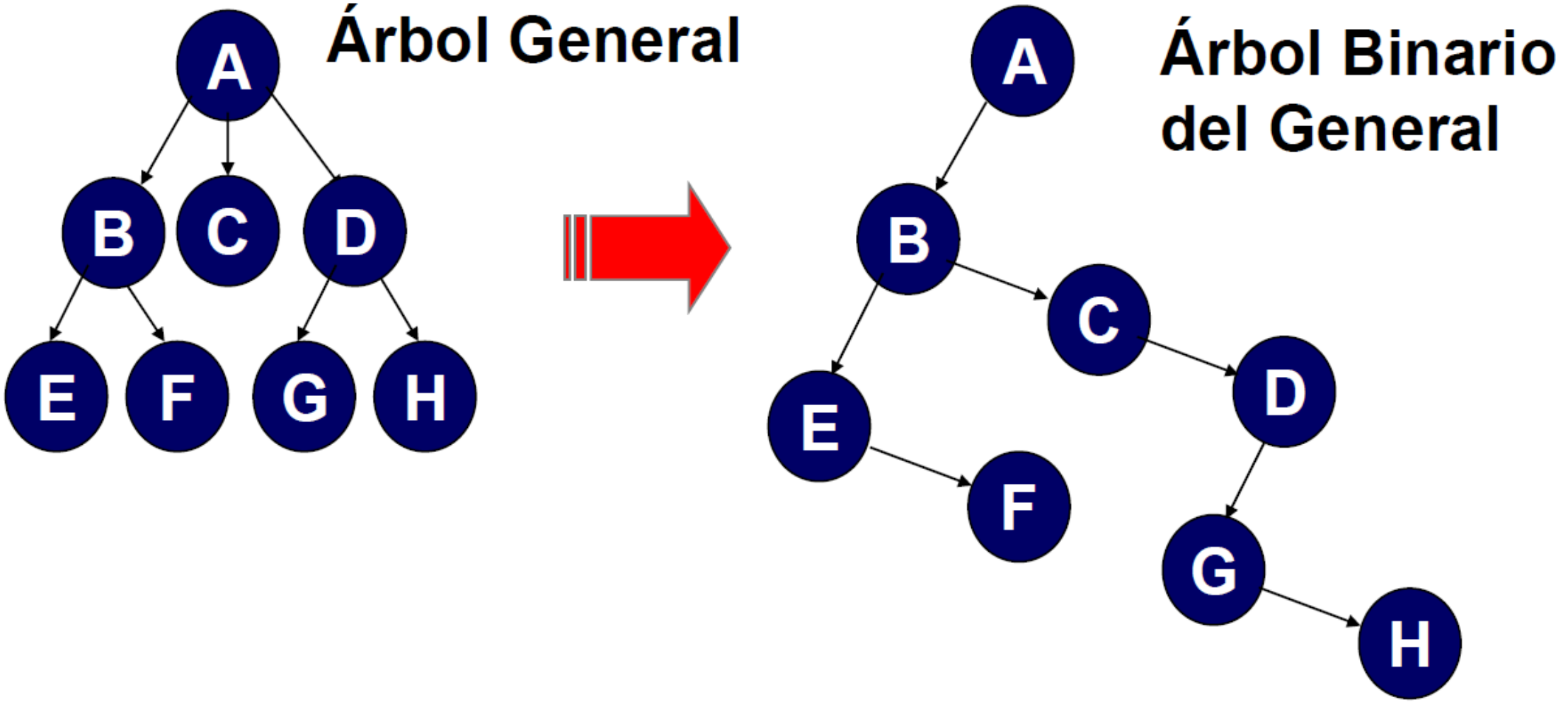
- Cada nodo apunta a una lista enlazada.
- En esa lista, cada nodo apunta a un nodo hijo.



# Transformar el árbol general en binario

- Los árboles generales son más difíciles de implementar que los árboles binarios.
- Un árbol general se puede convertir en binario.
- La raíz del árbol binario será la raíz del árbol general.
- Se enlaza el hijo situado más a la izquierda del nodo raíz del árbol general como hijo izquierdo del nodo raíz en el árbol binario.
- Se enlazan todos los hermanos como hijos derechos en el árbol binario.
- El proceso se repite con cada nodo.




- El que no tiene hijo izquierdo es hoja en el general.
- El que no tiene hijo derecho es el último hermano en el general.

# Colocación secuencial de árboles

- Cuando la representación del árbol general se realiza mediante arreglos y se realiza la transformación de un árbol general a binario, puede colocarse los elementos del árbol en forma ordenada en el arreglo de tal manera que se pueda recorrer el árbol en múltiples ocasiones.
- Los métodos más conocidos son:
  - Almacenamiento en ***Preorden secuencial***
  - Almacenamiento en ***Orden familiar***

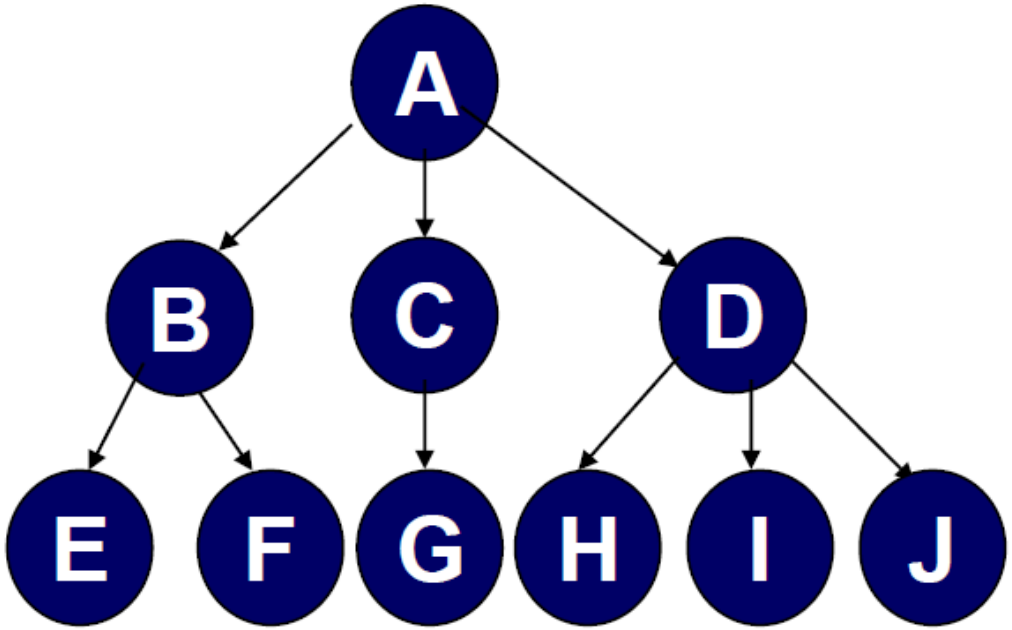
# Colocación en Pre-orden Secuencial

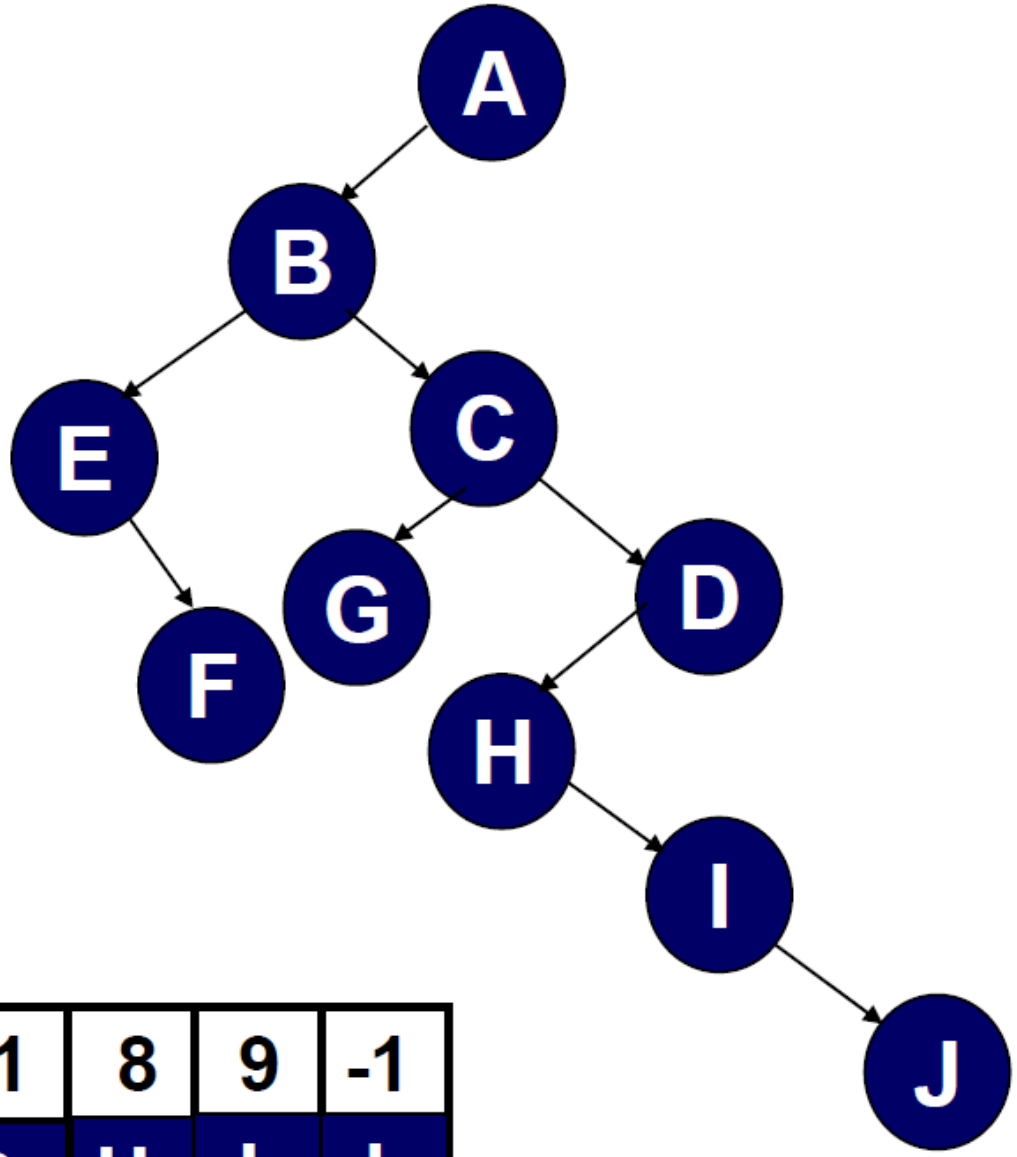
1. Se transforma el árbol general a binario.
2. Los nodos deben colocarse secuencialmente recorriendo el árbol en Pre-orden.
3. Por cada nodo se registran tres campos:

INFO: Árbol binario recorrido en Pre-orden.

ENLDER: Siguiente hermano en el árbol general, hijo derecho en el binario. (-1 si no existe)

TERM: Indica si el nodo es terminal (es una hoja en el general y no tiene hijo izquierdo en el binario).




| ENLDER | -1 | 4 | 3 | -1 | 6 | -1 | -1 | 8 | 9 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|
| INFO | A | B | E | F | C | G | D | H | I | J |
| TERM | F | F | T | T | F | T | F | T | T | T |

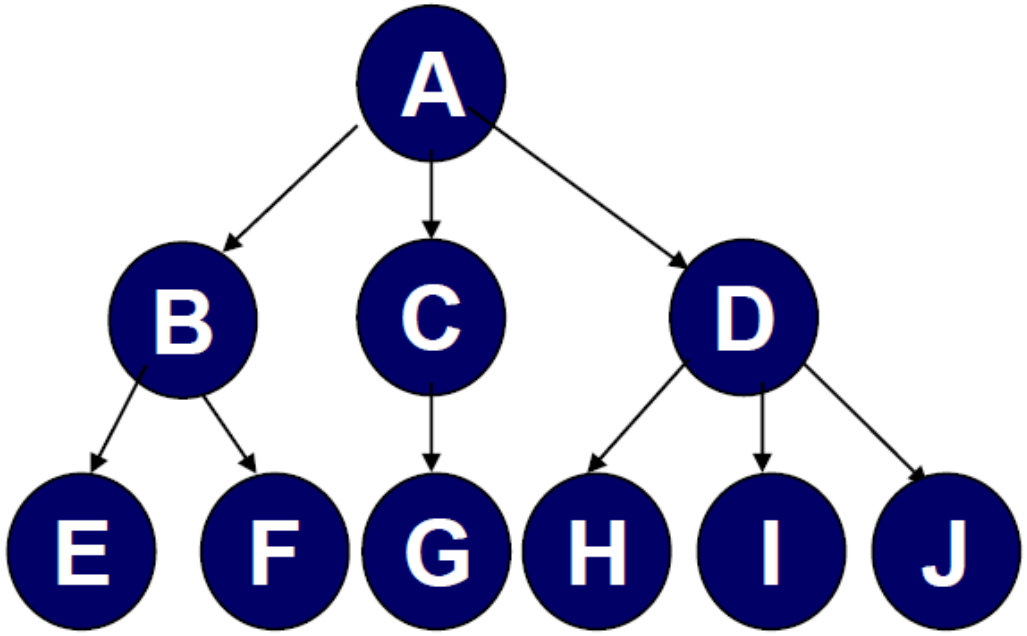
# Colocación en Orden Familiar

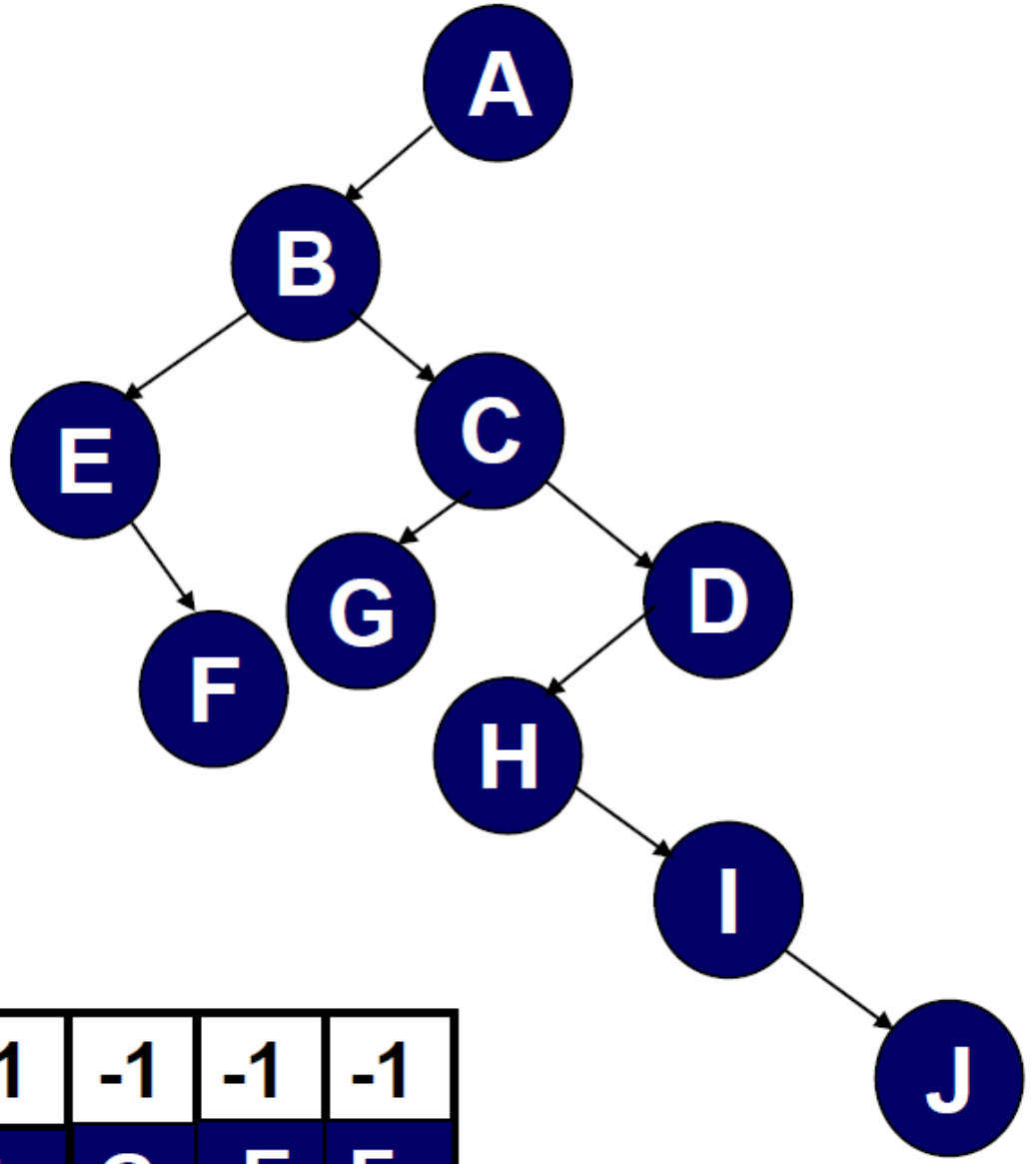
1. Se transforma a binario.
2. Los nodos deben colocarse secuencialmente recorriendo el árbol en post-orden invertido (Raíz, Sub Árbol Derecho, Sub Árbol Izquierdo).
3. Por cada nodo se registra tres campos:

INFO: Árbol binario recorrido en Post-orden invertido.

ENLIZQ: Primer hijo en el árbol general e hijo izquierdo en el binario. (-1 si no existe)

FAM: Indica último hermano en el árbol general y enlace derecho en NULL en el binario.




| ENLIZQ | 1 | 8 | 7 | 4 | -1 | -1 | -1 | -1 | -1 | -1 |
|---|---|---|---|---|---|---|---|---|---|---|
| INFO | A | B | C | D | H | I | J | G | E | F |
| FAM | T | F | F | T | F | F | T | T | F | T |