

A large, ancient tree trunk with a thick, textured bark dominates the center of the image. Sunlight filters through the dense canopy of green leaves above, creating bright highlights on the trunk and the surrounding ground. The overall atmosphere is natural and serene.

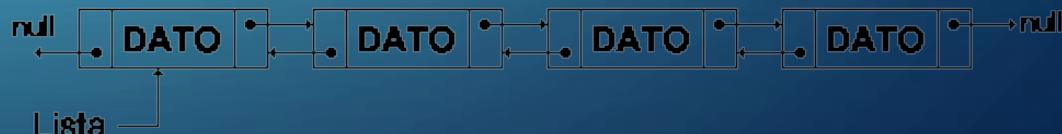
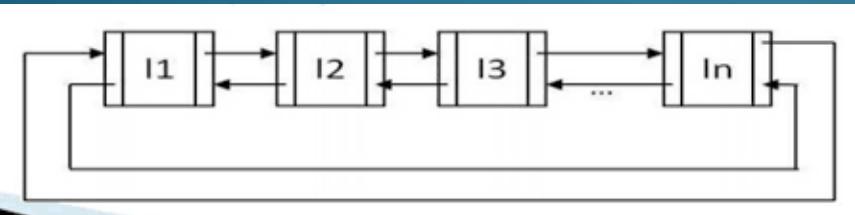
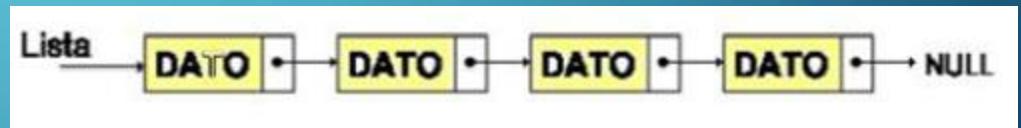
ÁRBOLES

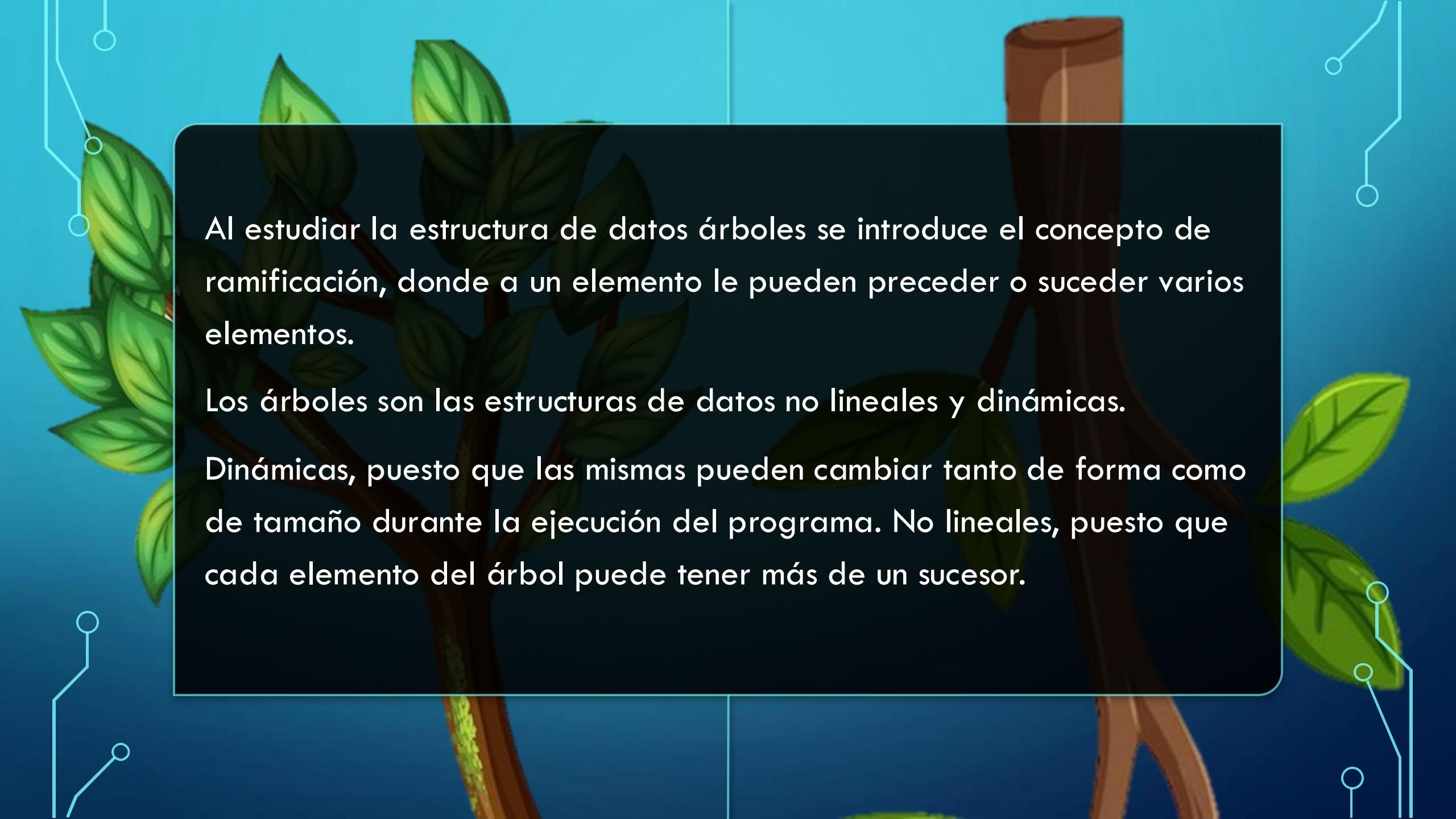
ESTRUCTURA DE DATOS



INTRODUCCIÓN

Hasta el momento sólo se han estudiado estructuras de datos lineales donde a cada elemento siempre le sucede o le precede como máximo otro elemento.





Al estudiar la estructura de datos árboles se introduce el concepto de ramificación, donde a un elemento le pueden preceder o suceder varios elementos.

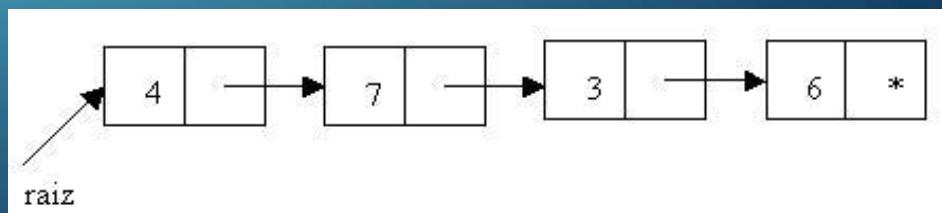
Los árboles son las estructuras de datos no lineales y dinámicas.

Dinámicas, puesto que las mismas pueden cambiar tanto de forma como de tamaño durante la ejecución del programa. No lineales, puesto que cada elemento del árbol puede tener más de un sucesor.

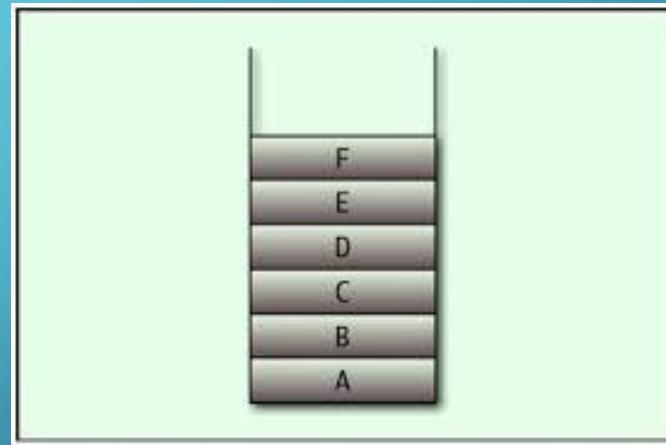
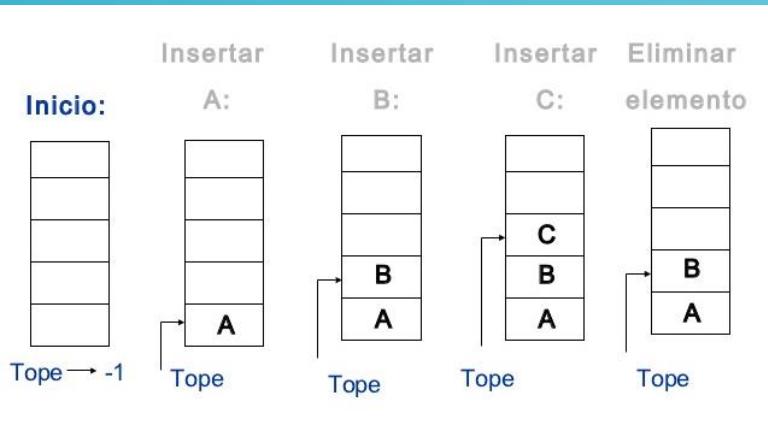
En la siguiente tabla se presentan las principales estructuras de datos clasificadas de acuerdo con su capacidad para cambiar en forma y tamaño, durante la ejecución del programa

Sueldos				
1200	750	820	550	490
Sueldos[1]	Sueldos[2]	Sueldos[3]	Sueldos[4]	Sueldos[5]

Estructuras estáticas	Estructuras dinámicas
Arreglos	Listas
Registros	Árboles
	Gráficas

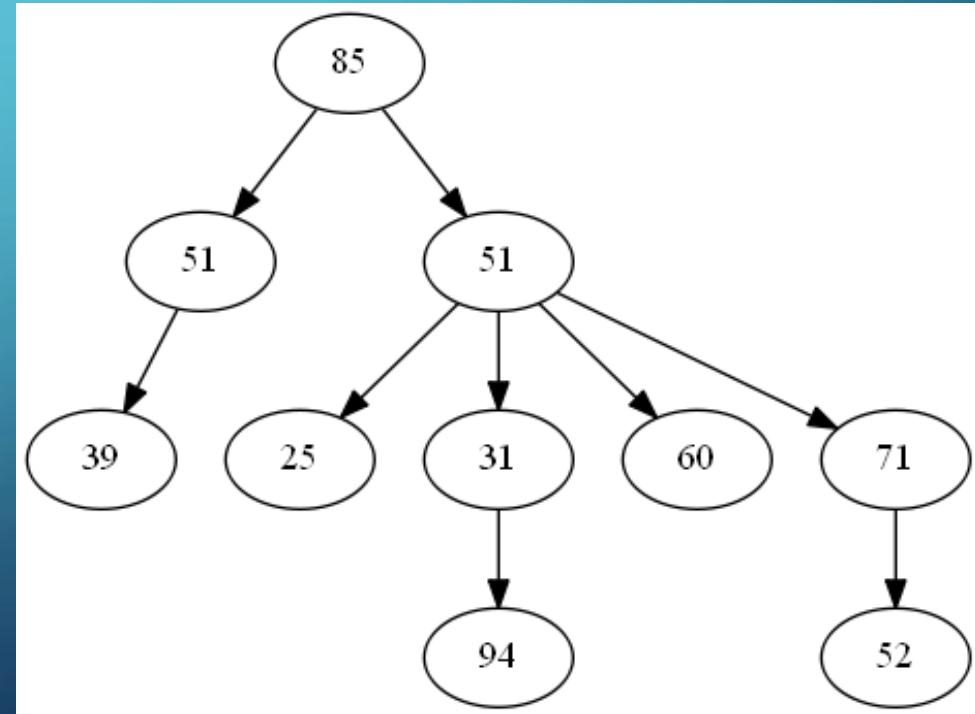
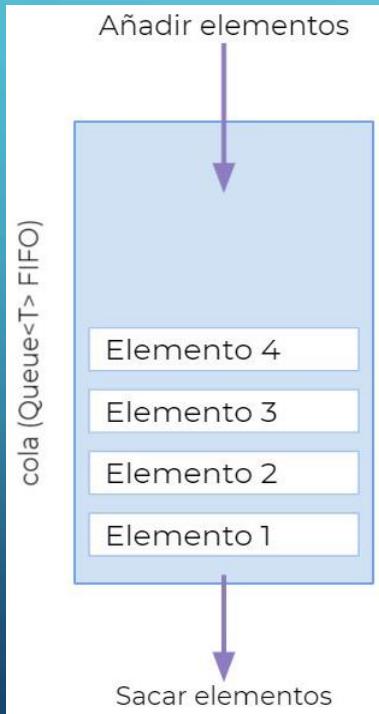


No se consideran a las pilas y colas, puesto que dependen de la estructura que se utilice para implementarlas. Si se usan arreglos, se tratan como estructuras estáticas, más sin embargo se implementan con listas, serán estructuras dinámicas.



En la tabla siguiente se presentan las principales estructuras de datos clasificadas según su distribución de sus elementos.

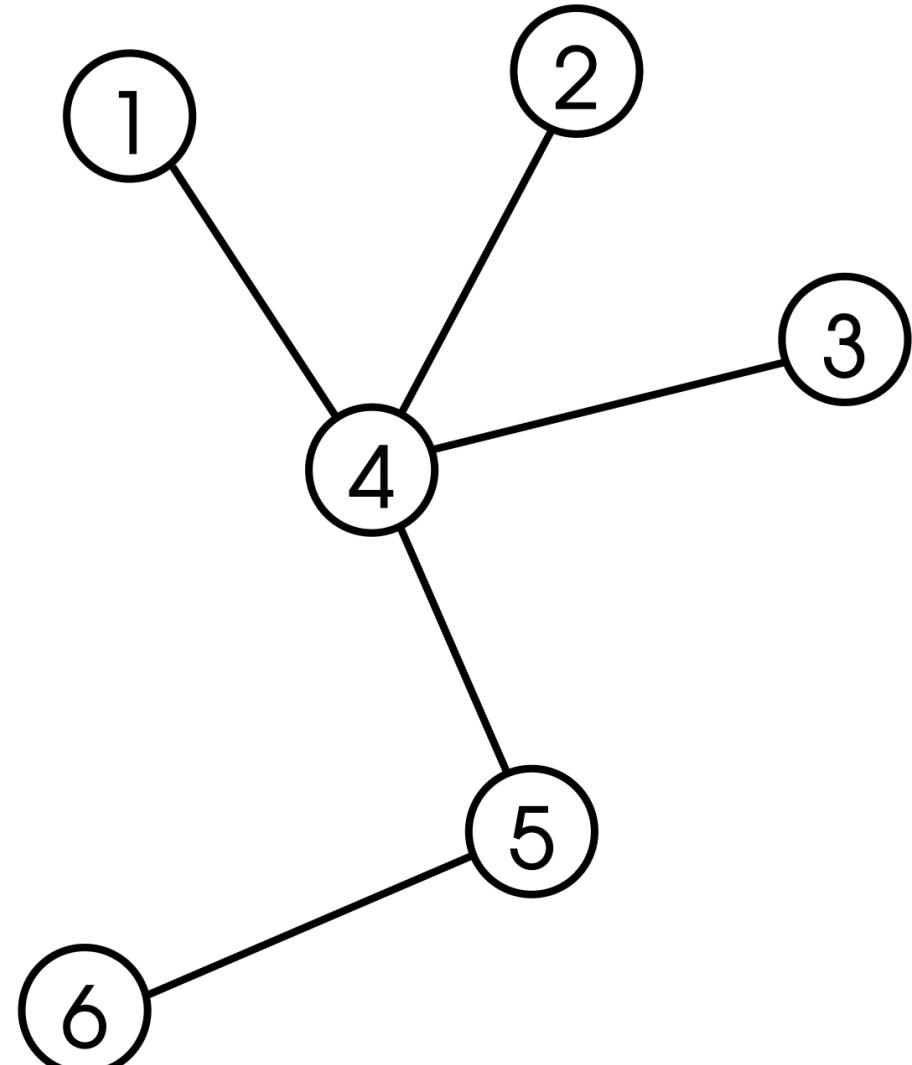
Estructuras Lineales	Estructuras no lineales
Arreglos	Árboles
Registros	Gráficas
Pilas	
Colas	
Listas	



DEFINICIÓN MATEMÁTICA

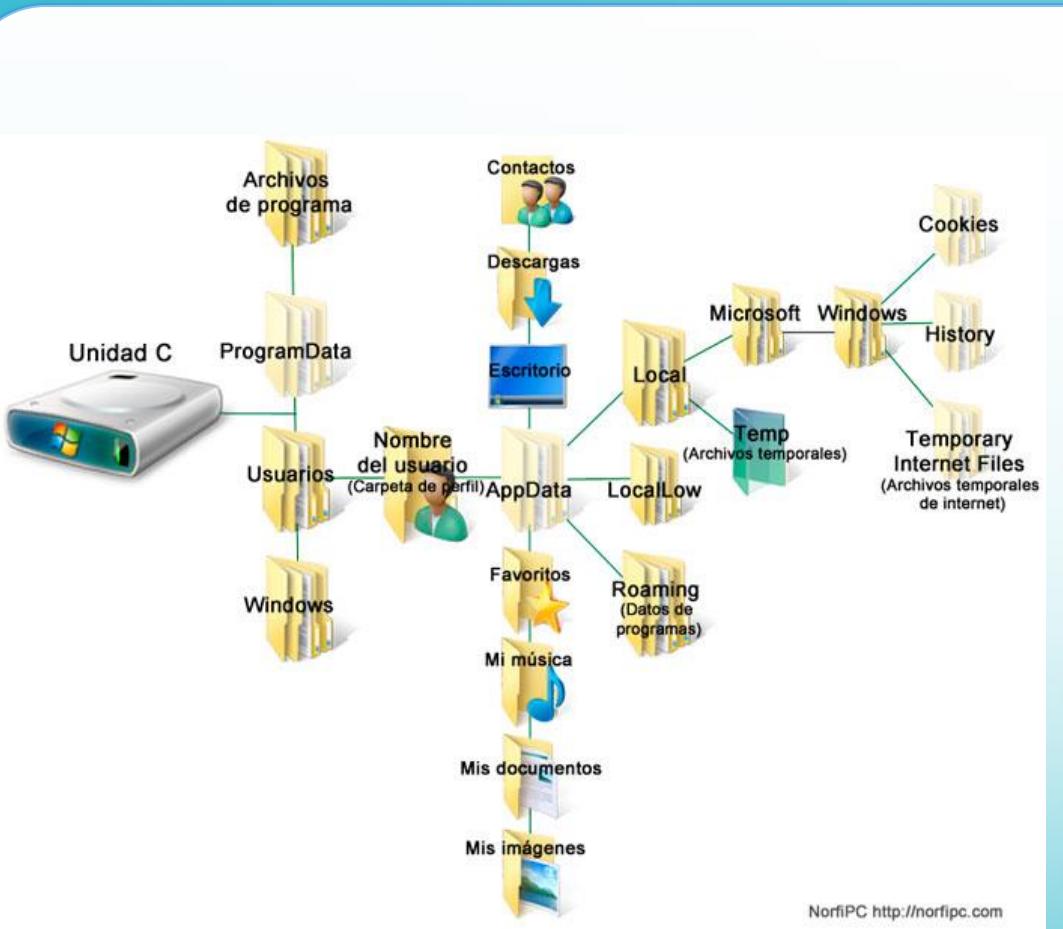
En teoría de grafos, se conoce a un árbol como a aquel grafo no dirigido compuesto por un conjunto de vértices y aristas tal que:

- Es conexo: Existe un camino entre cualquier par de vértices.
- Es acíclico: No tiene ciclos.



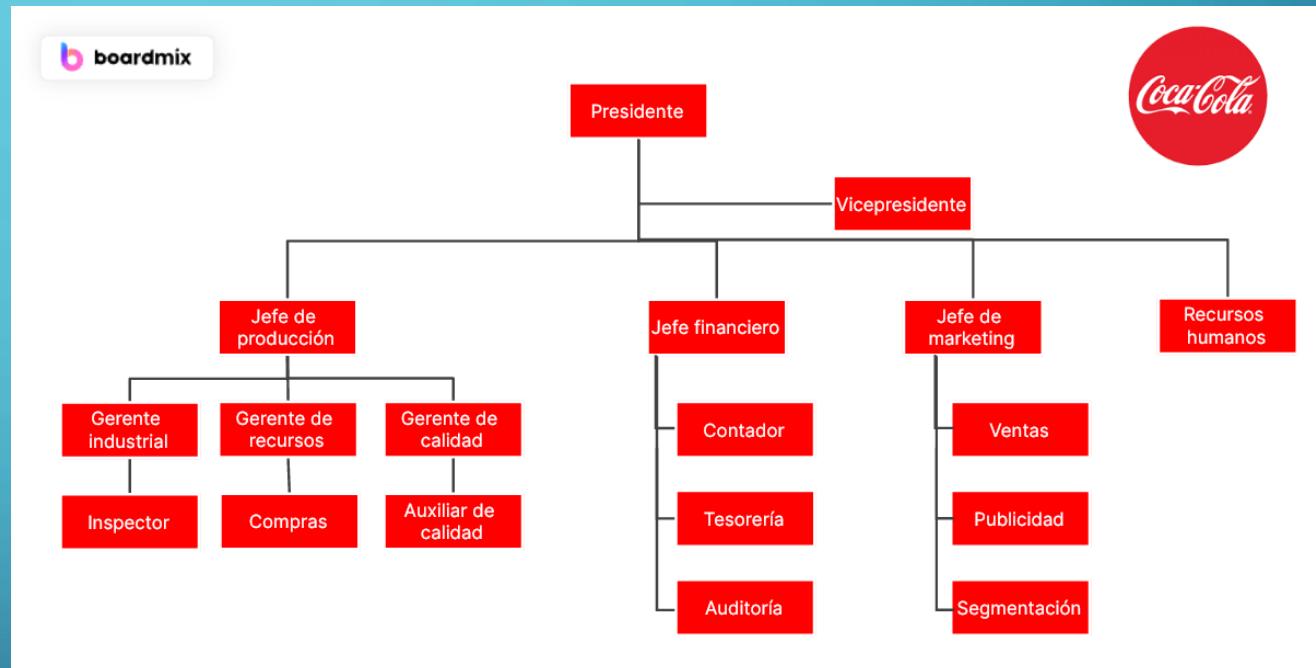
ARBOLES GENERALES

Un árbol se puede definir como una estructura jerárquica aplicada sobre una colección de elementos u objetos llamados nodos, uno de los cuales es conocido como raíz. Además, se crea una relación entre los nodos.



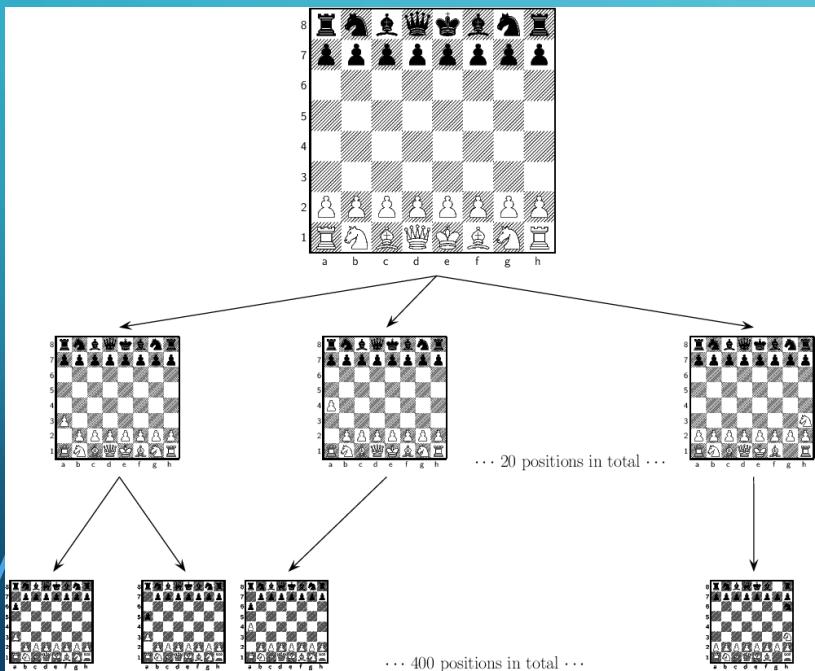
Estos son estructuras recursivas,
ya que a cada subárbol es a su
vez un árbol.

Formalmente se define un árbol
de tipo T como una estructura
homogénea resultado de la
concatenación de un elemento
de tipo T con un numero finito
arboles disjuntos. Una forma
particular de árbol es el árbol
vacío.



OTROS EJEMPLOS DE ARBOLES GENERALES

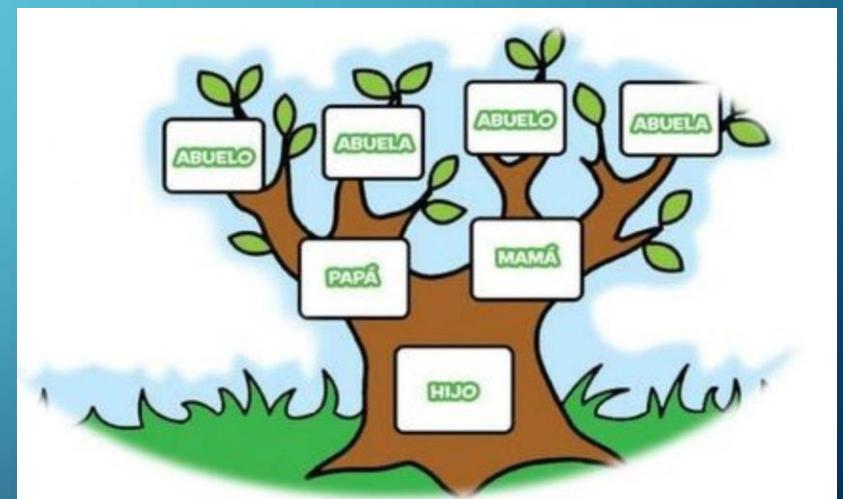
Posibles movimientos en una partida de ajedrez



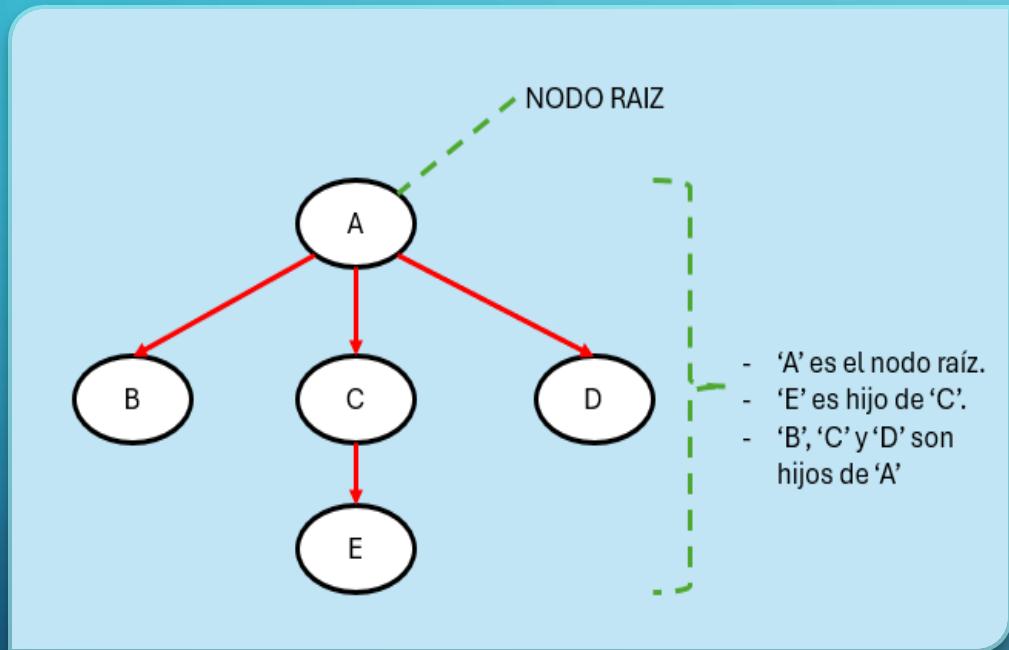
Representación de un JSON

```
JSON
  ↳ widget
    debug : "on"
  ↳ window
    title : "Sample Konfabulator Widget"
    name : "main_window"
    width : 500
    height : 500
  ↳ image
    src : "Images/Sun.png"
    name : "sun1"
    hOffset : 250
    vOffset : 250
    alignment : "center"
  ↳ text
    data : "Click Here"
    size : 36
    style : "bold"
    name : "text1"
    hOffset : 250
    vOffset : 100
    alignment : "center"
    onMouseUp : "sun1.opacity = (sun1.opacity / 100) * 90;"
```

Árbol genealógico

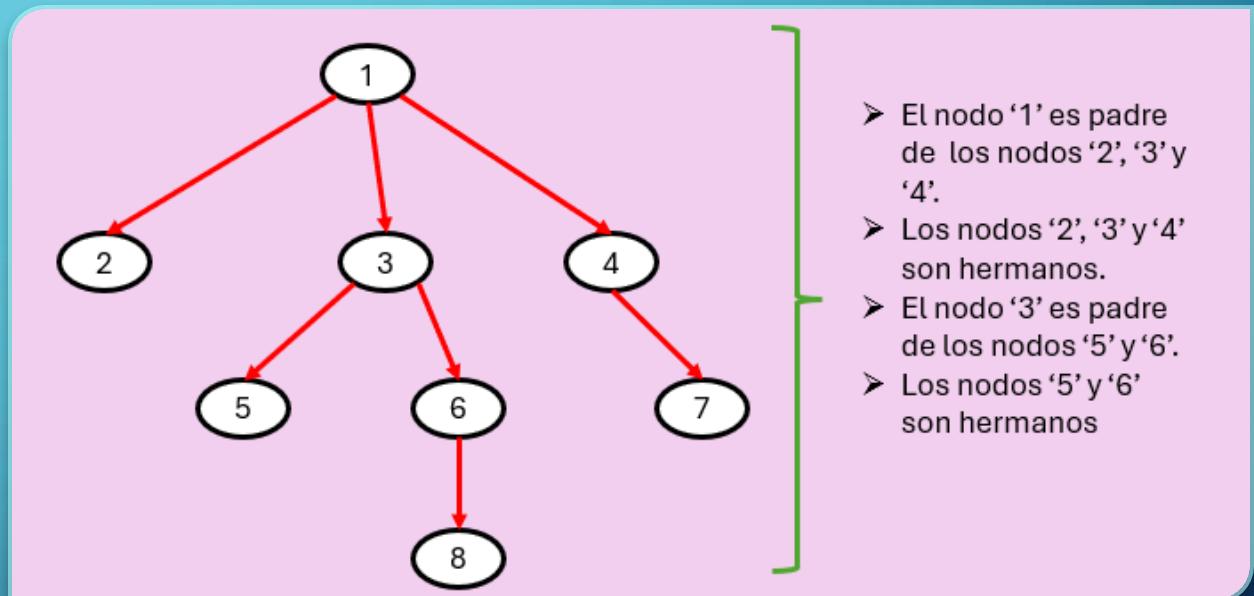


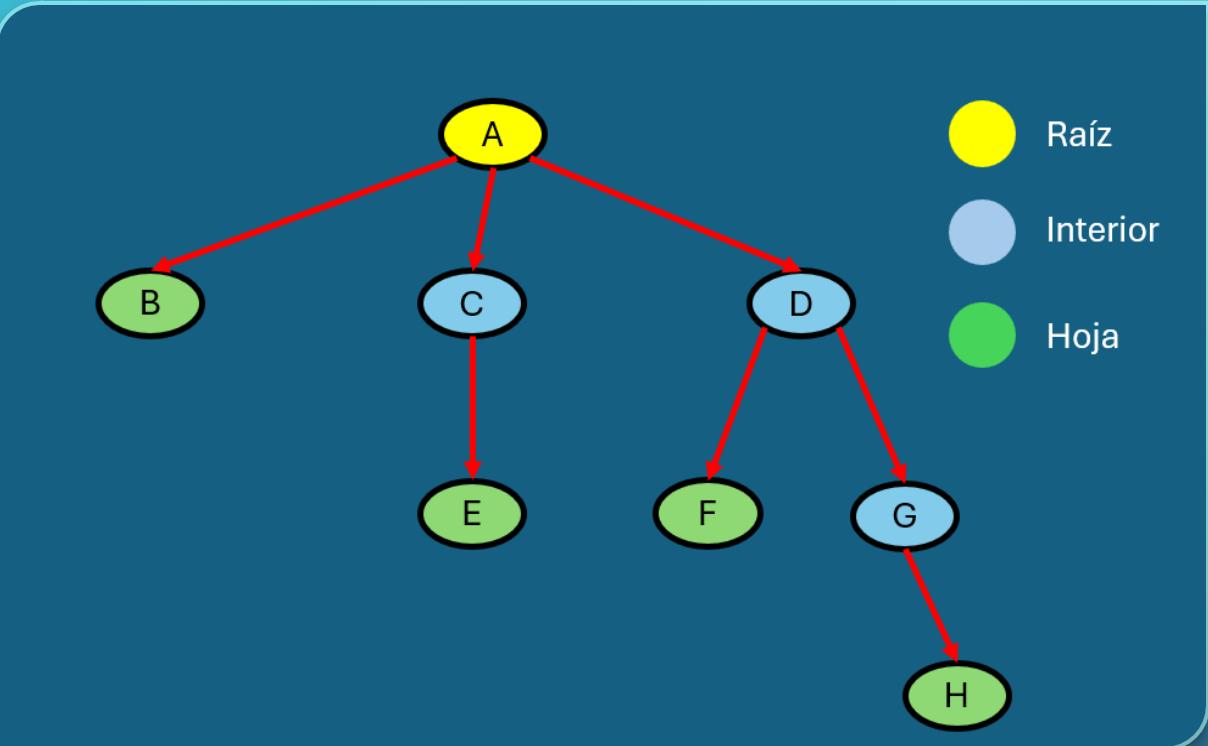
TERMINOLOGÍA / PROPIEDADES DE LOS ARBOLES



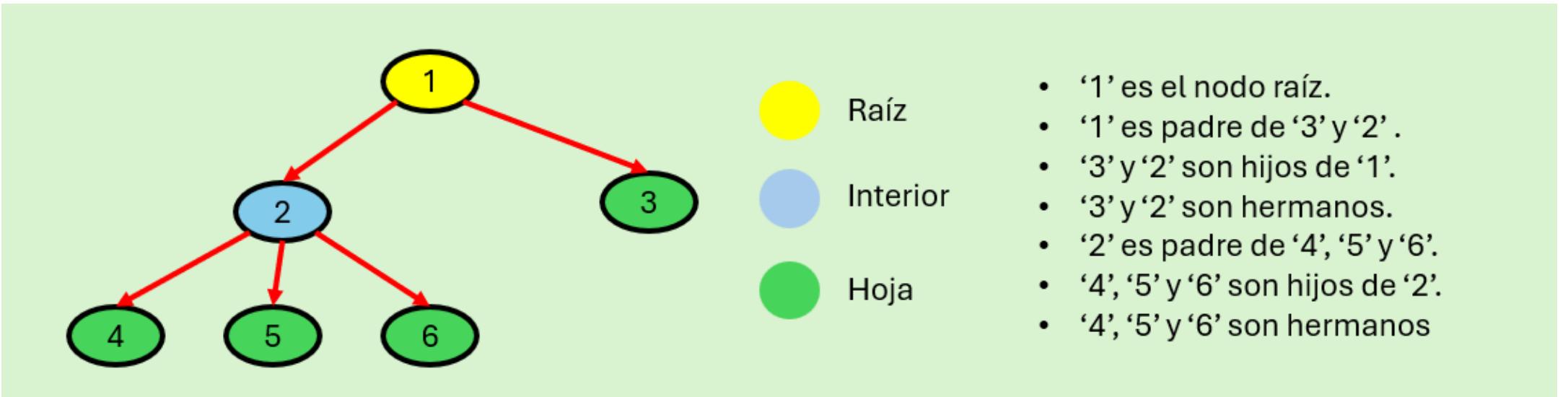
- Todo árbol que no es vacío tiene un único nodo raíz.
- Un nodo X es descendiente directo de un nodo Y, si el nodo X es apuntado por el nodo Y. En este caso es común utilizar la expresión "X es hijo de Y".

- Un nodo X es antecesor directo de un nodo Y, si el nodo X apunta al nodo Y. En este caso es común utilizar la expresión "X es padre de Y".
- Todos los nodos que son descendientes directos (Hijos) de un mismo nodo (Padre) son "hermanos".

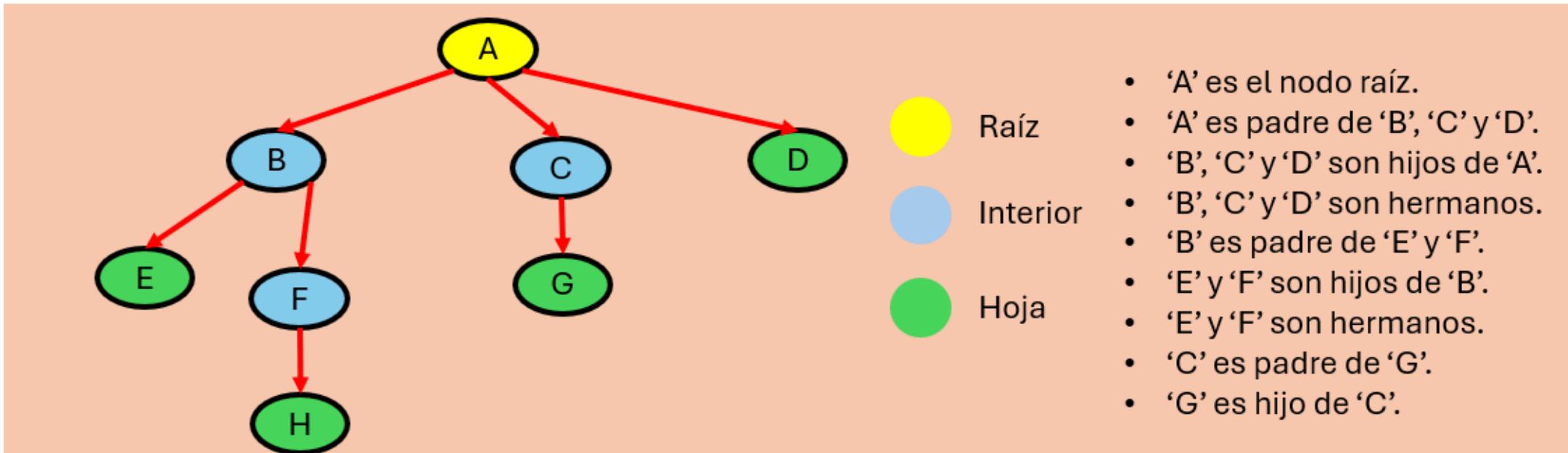




- Todo nodo que no tiene ramificaciones se conoce con el nombre de "terminal" u "hoja".
- Todo nodo que no es raíz ni terminal u hoja se conoce con el nombre de interior.



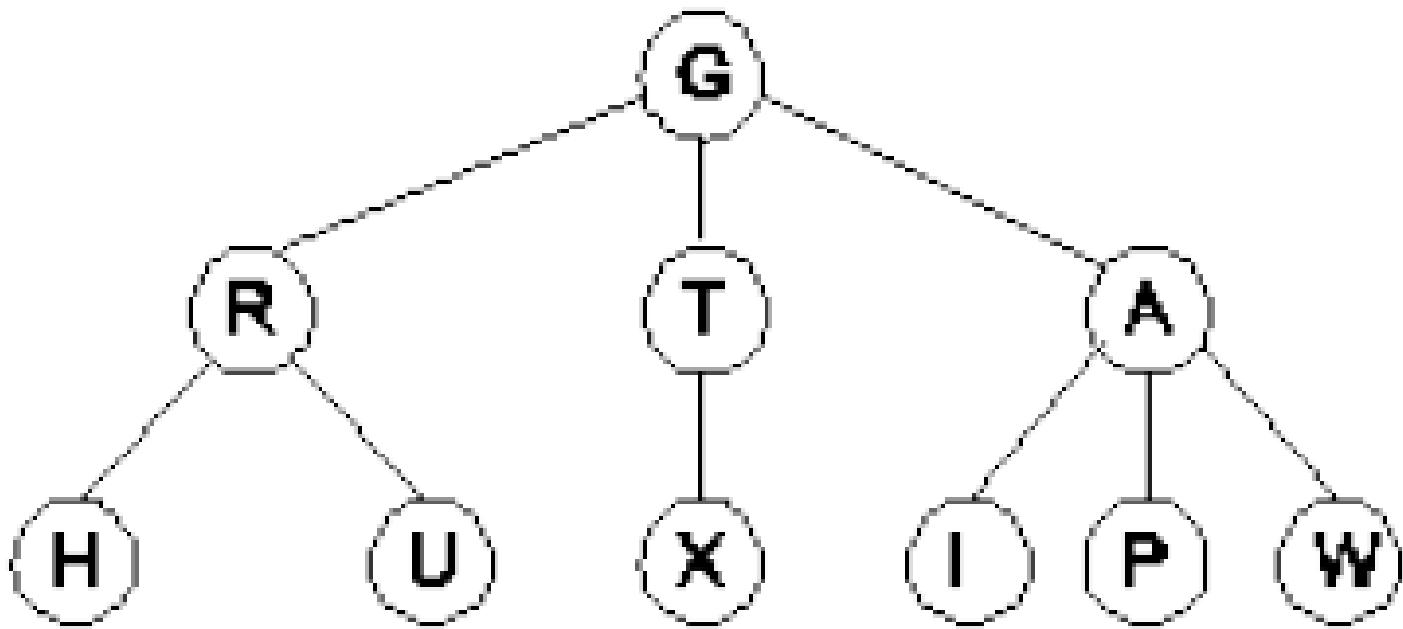
EJEMPLO COMPLETO 1



EJEMPLO COMPLETO 2

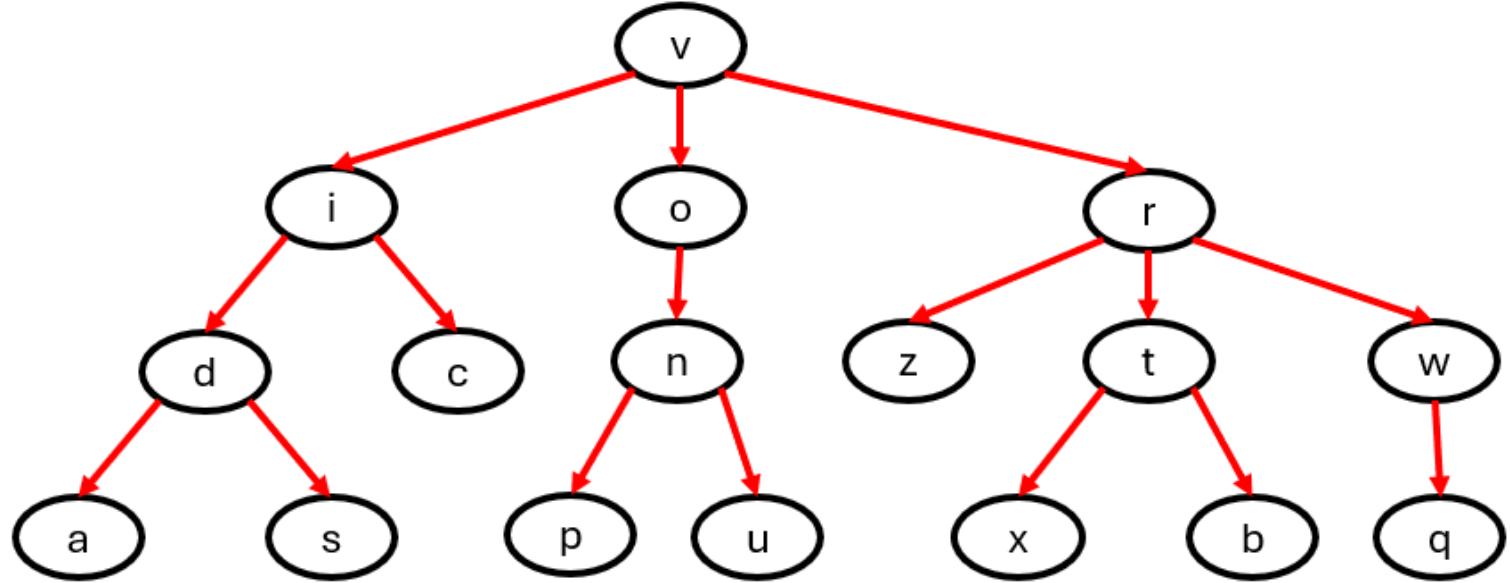
EJERCICIO 1

Determina las propiedades de los nodos del siguiente árbol:



EJERCICIO 2

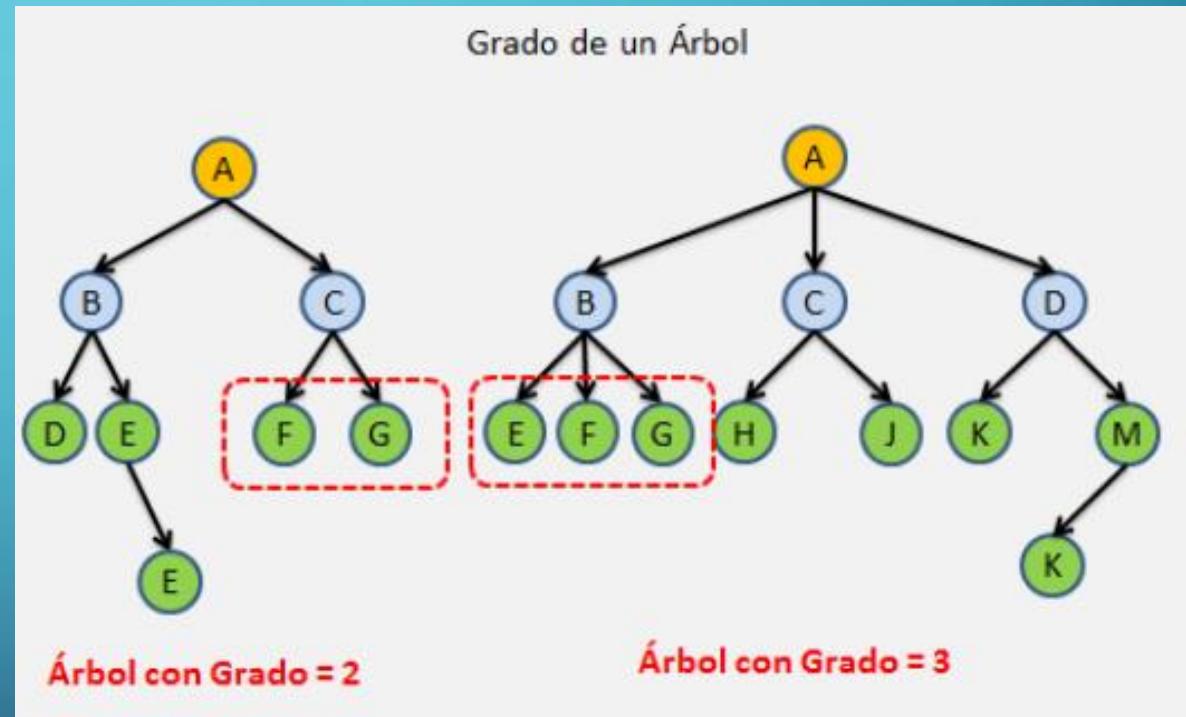
Determina las propiedades de los nodos del siguiente árbol:



TERMINOLOGÍA / CARACTERÍSTICAS DE UN ÁRBOL

GRADO

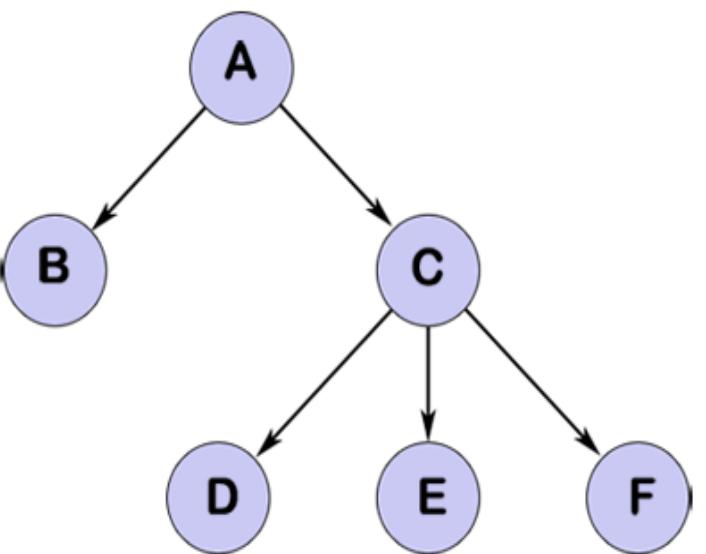
Número de hijos de un nodo y está limitado por el Orden, ya que este indica el número máximo de hijos que puede tener un nodo. El grado de un árbol se define como el máximo grado de todos sus nodos.



Nivel 1

Nivel 2

Nivel 3

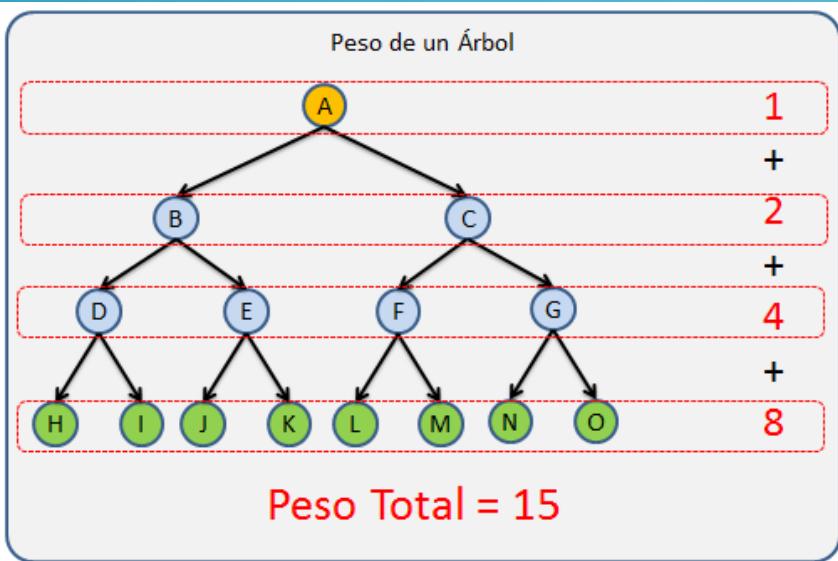
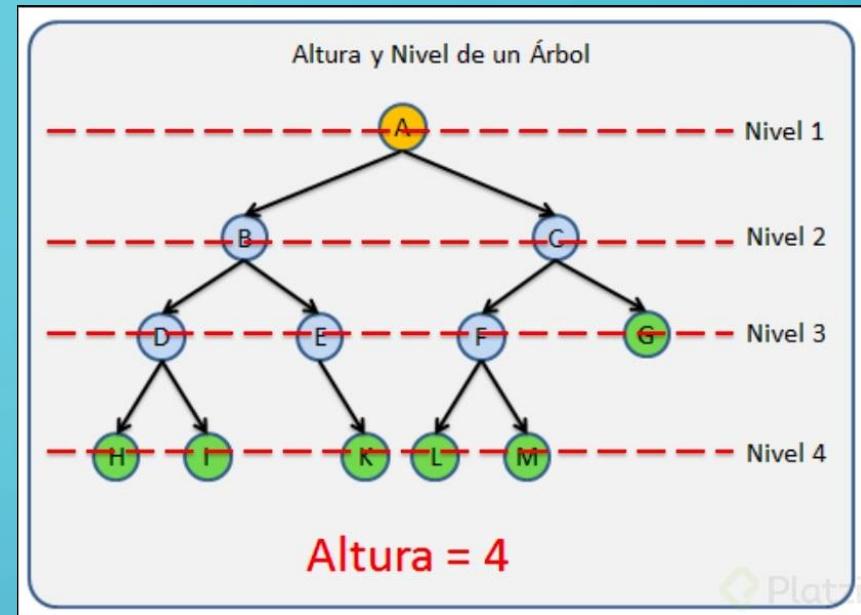


NIVEL

Es el número de arcos que deben ser recorridos para llegar a un determinado nodo. Por definición la raíz tiene nivel 1, aunque también puede considerarse en el nivel 0, lo cual afecta las operaciones relacionadas con el nivel.

ALTURA

Es el máximo número de niveles de todos los nodos del árbol.

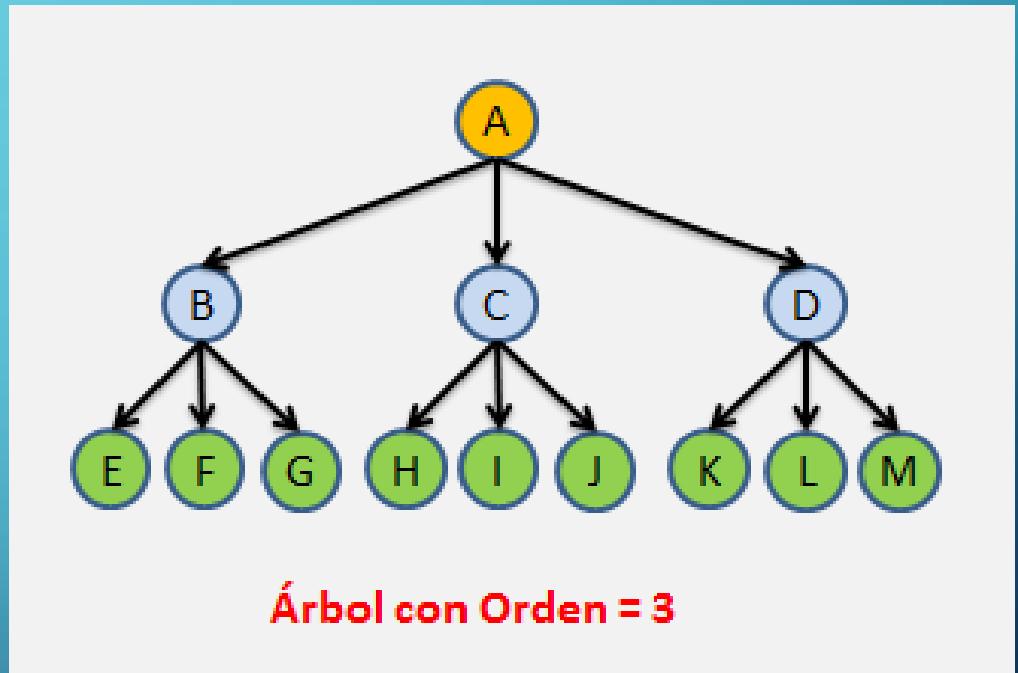


PESO

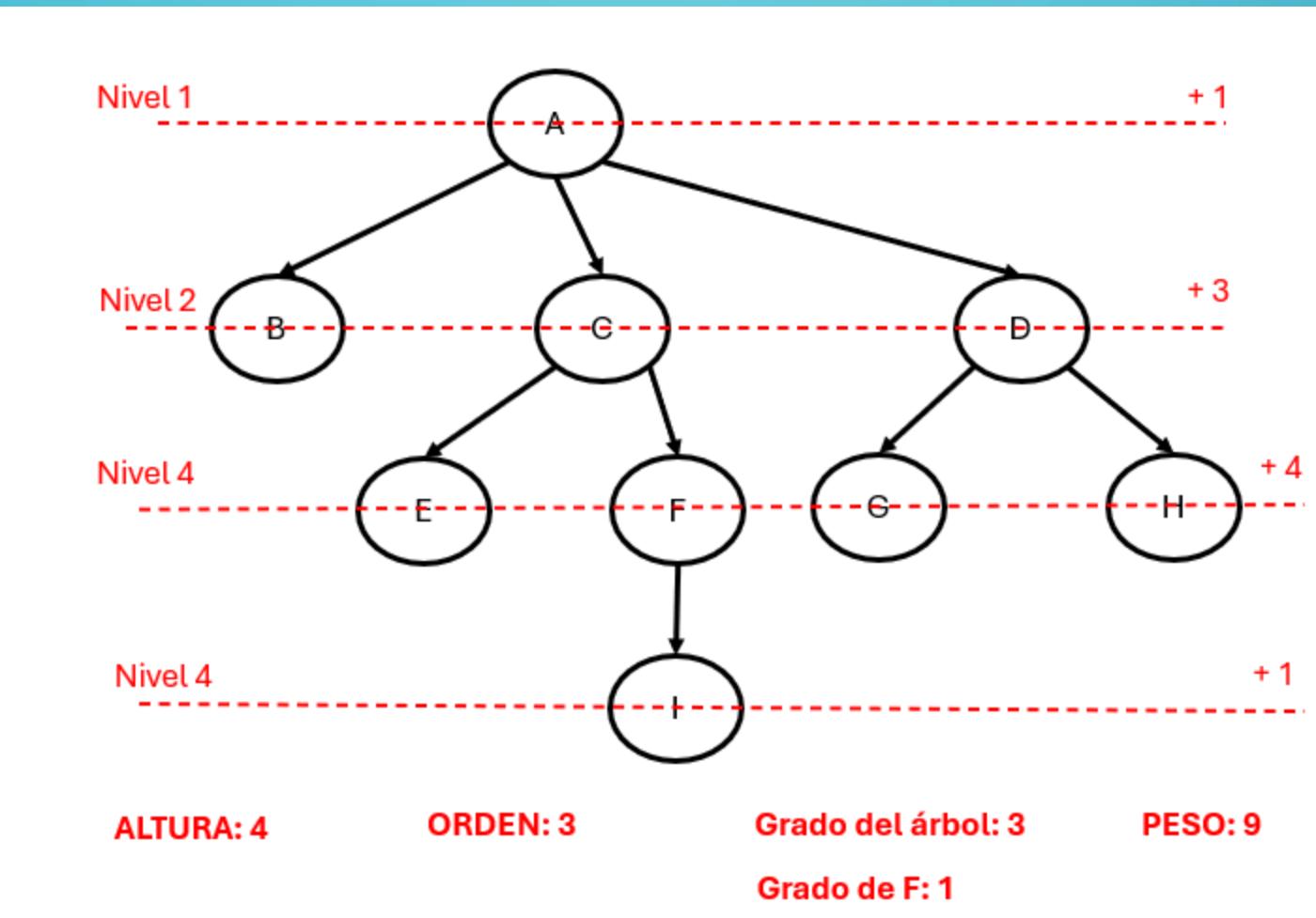
Es el número de nodos que tiene el árbol.

ORDEN

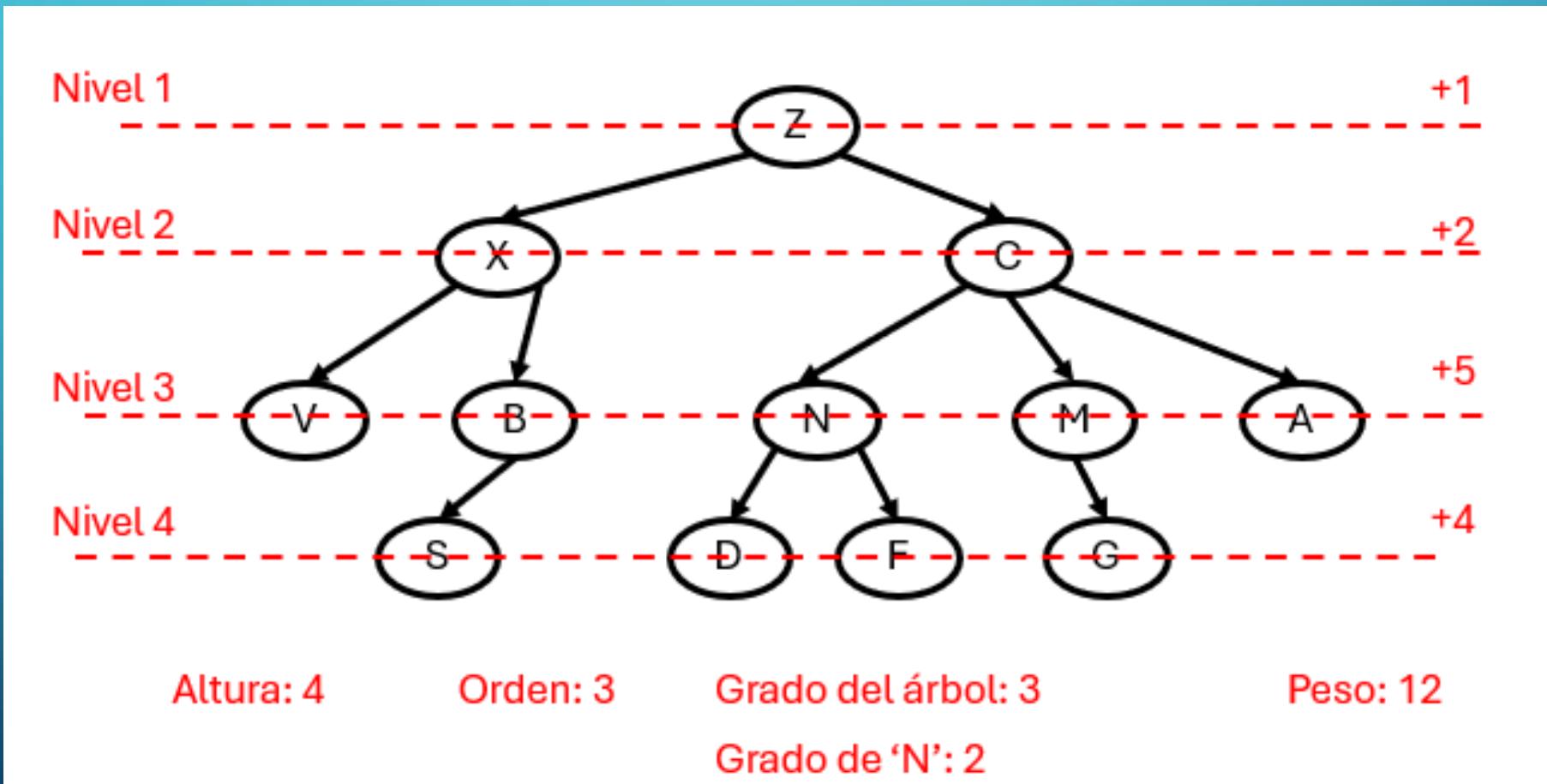
El Orden de un árbol es el número máximo de hijos que puede tener un Nodo. Es una constante que se define antes de crear el árbol.



EJEMPLO COMPLETO 1

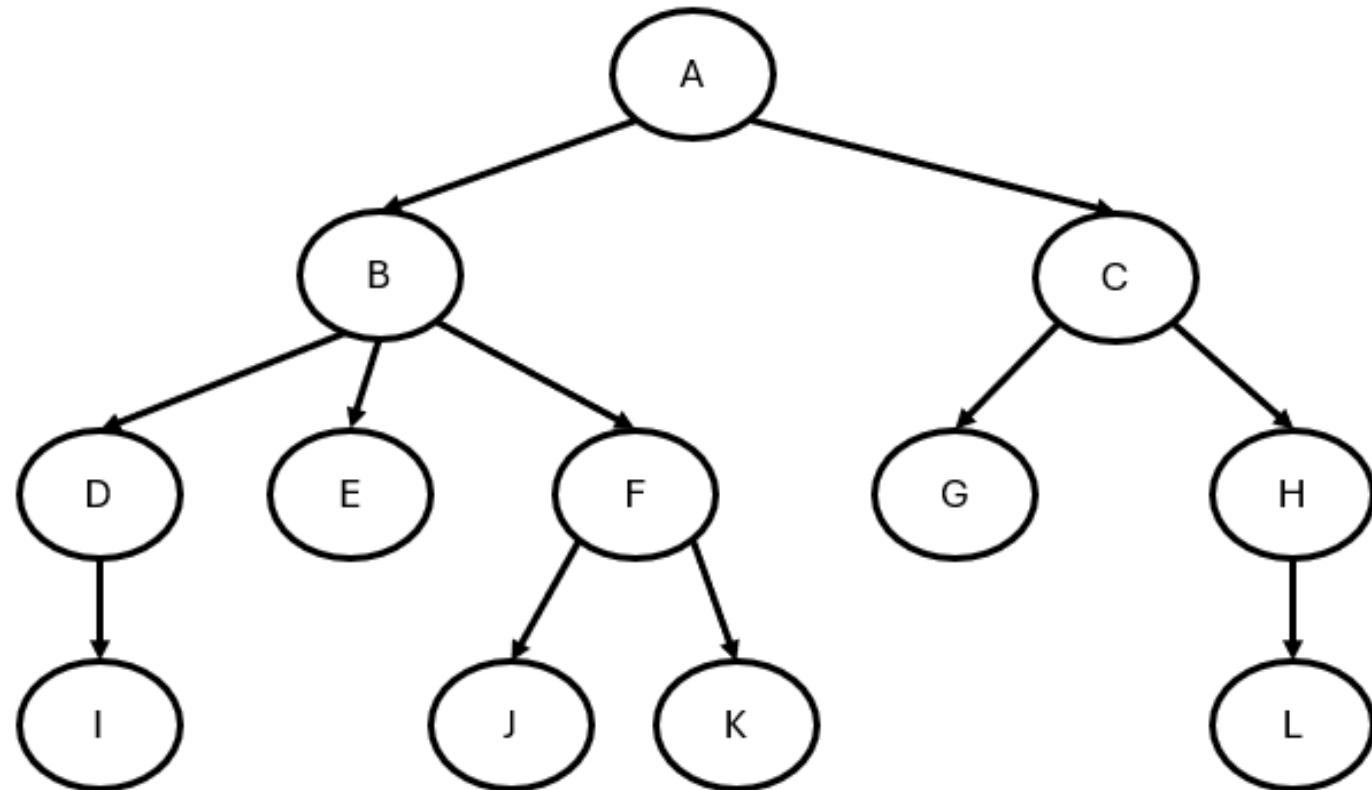


EJEMPLO COMPLETO 2



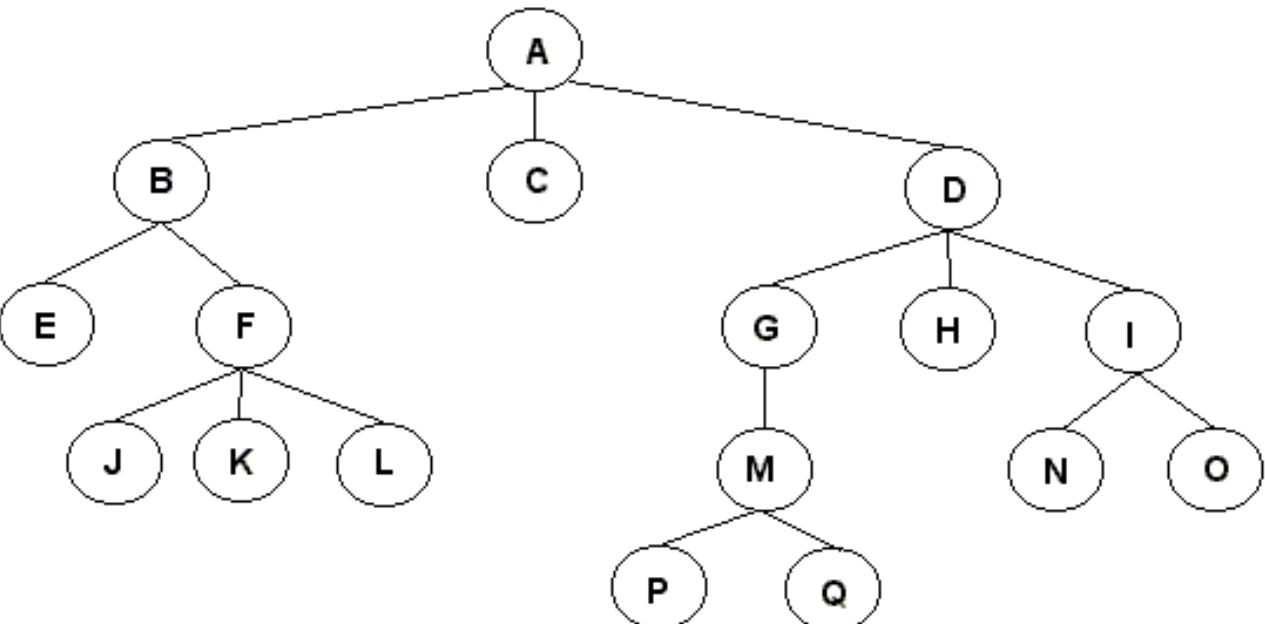
EJERCICIO 1

Determina las propiedades de los nodos y las características del siguiente árbol:



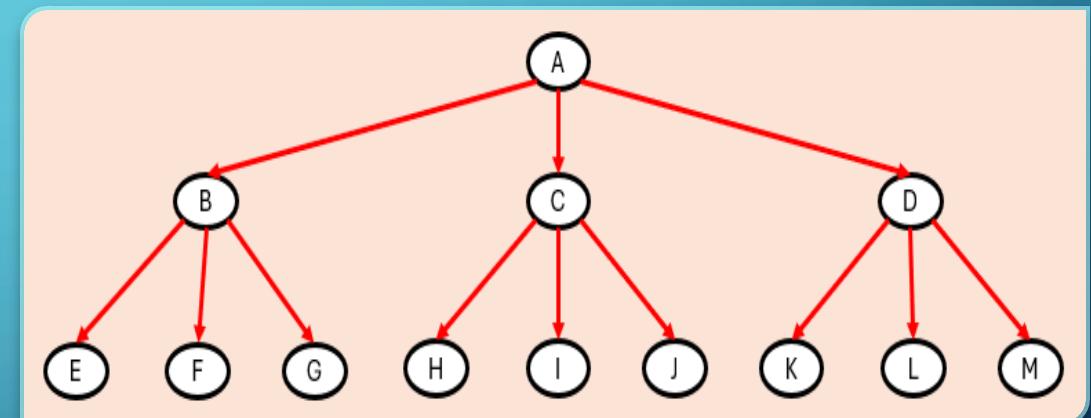
EJERCICIO 2

Determina las propiedades de los nodos y las características del siguiente árbol:



ÁRBOL EXTENDIDO

Aquel en el que el número de hijos de cada nodo es igual al grado del árbol. Si alguno de los nodos del árbol no cumple con esta condición se deben incorporar al mismo tantos nodos especiales como se requiera.



NODOS ESPECIALES

Tienen como objetivo remplazar las ramas vacías o nulas, no pueden tener descendientes y normalmente se representan con la forma de un cuadrado.

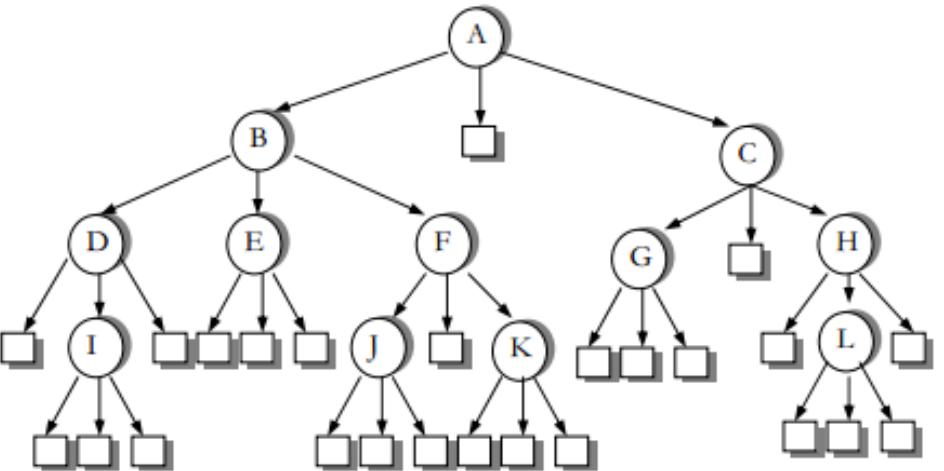
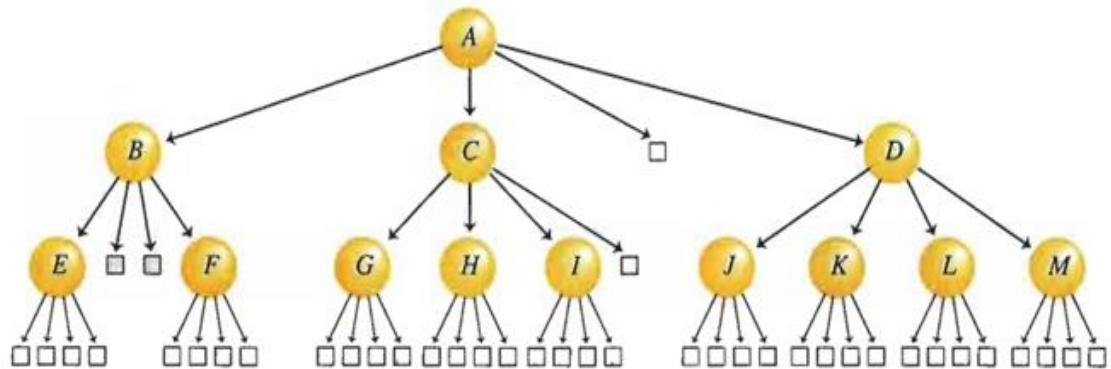
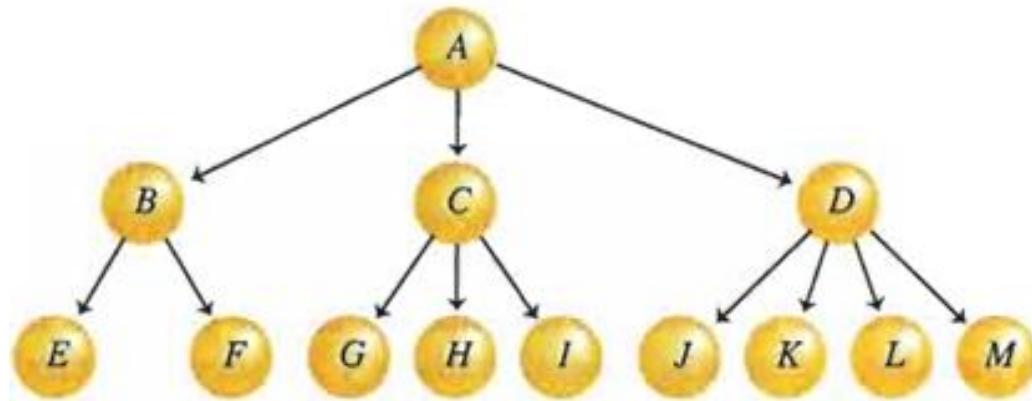
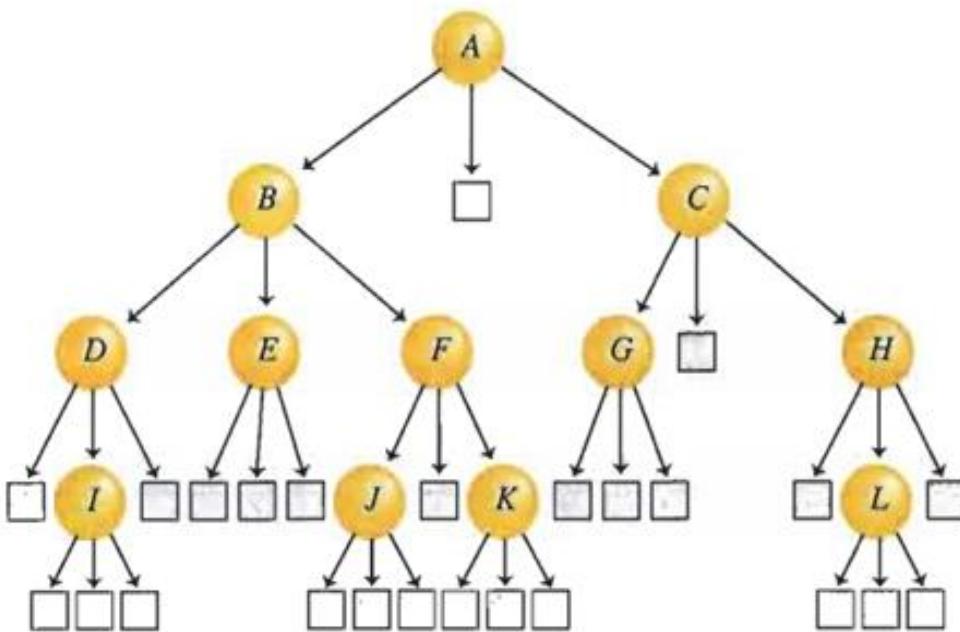
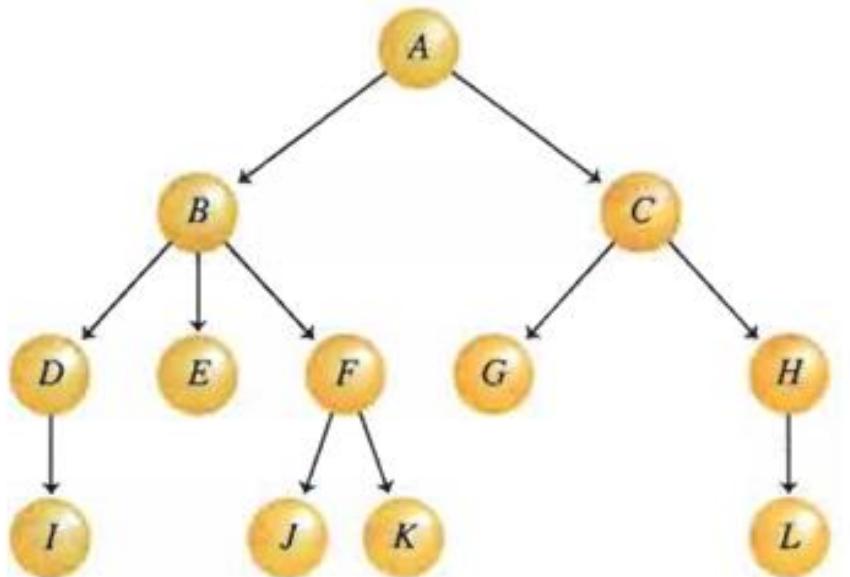


Figura 1.23 Árbol Extendido



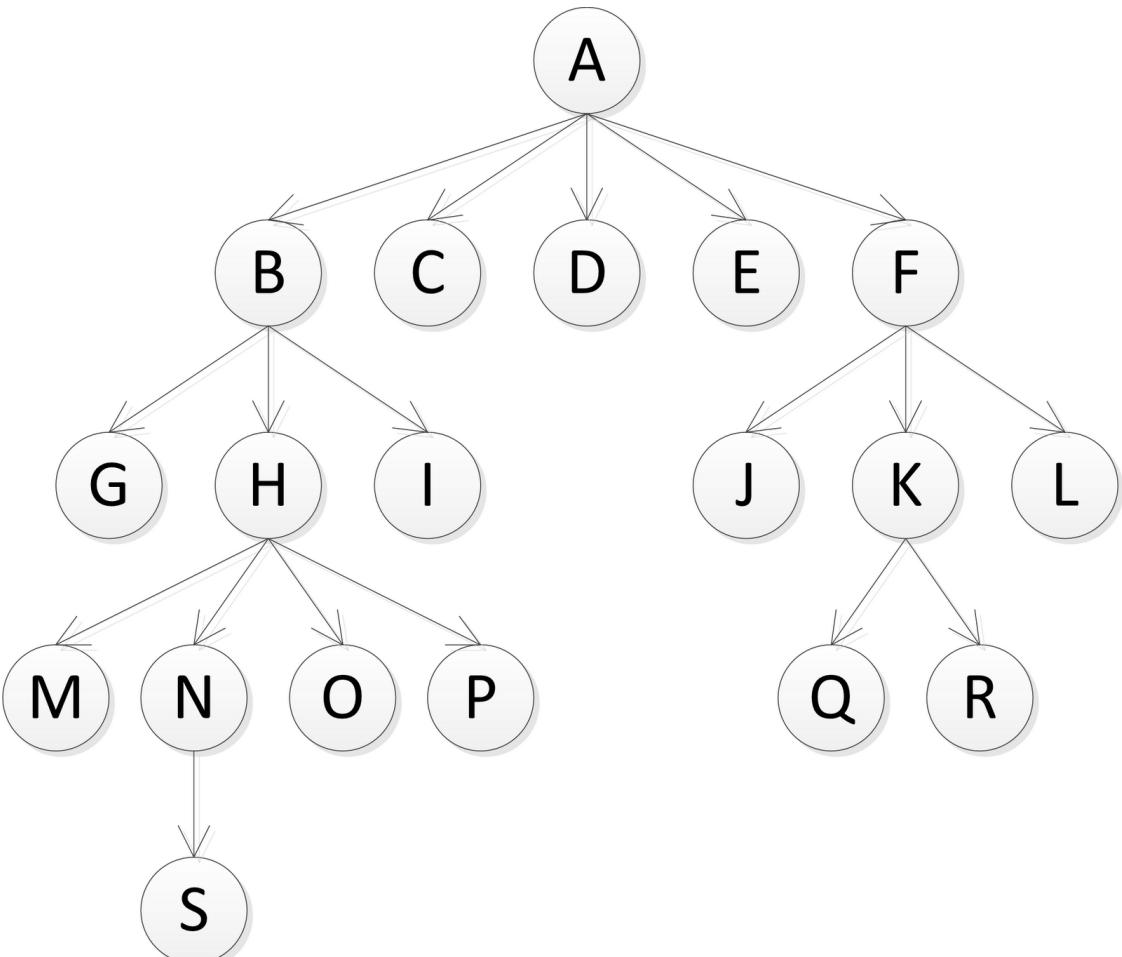
EJEMPLO 1. ÁRBOLES EXTENDIDOS



EJEMPLO 2. ÁRBOLES EXTENDIDOS

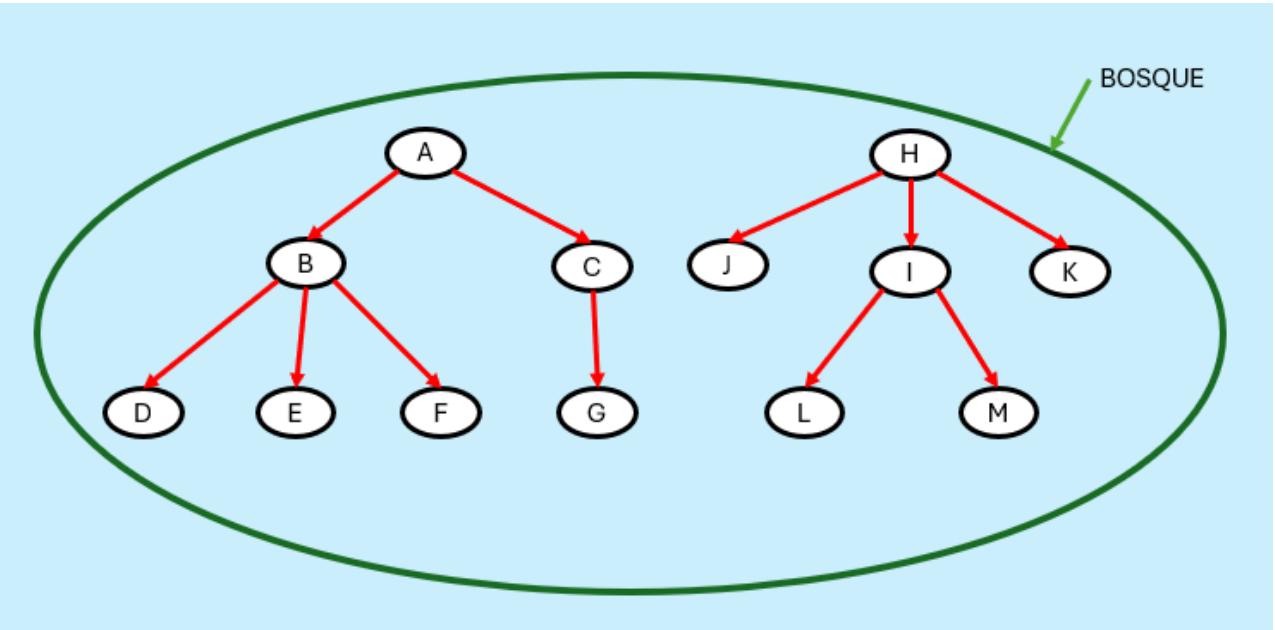
EJERCICIO. ÁRBOLES EXTENDIDOS

Obtener el árbol extendido del siguiente árbol.



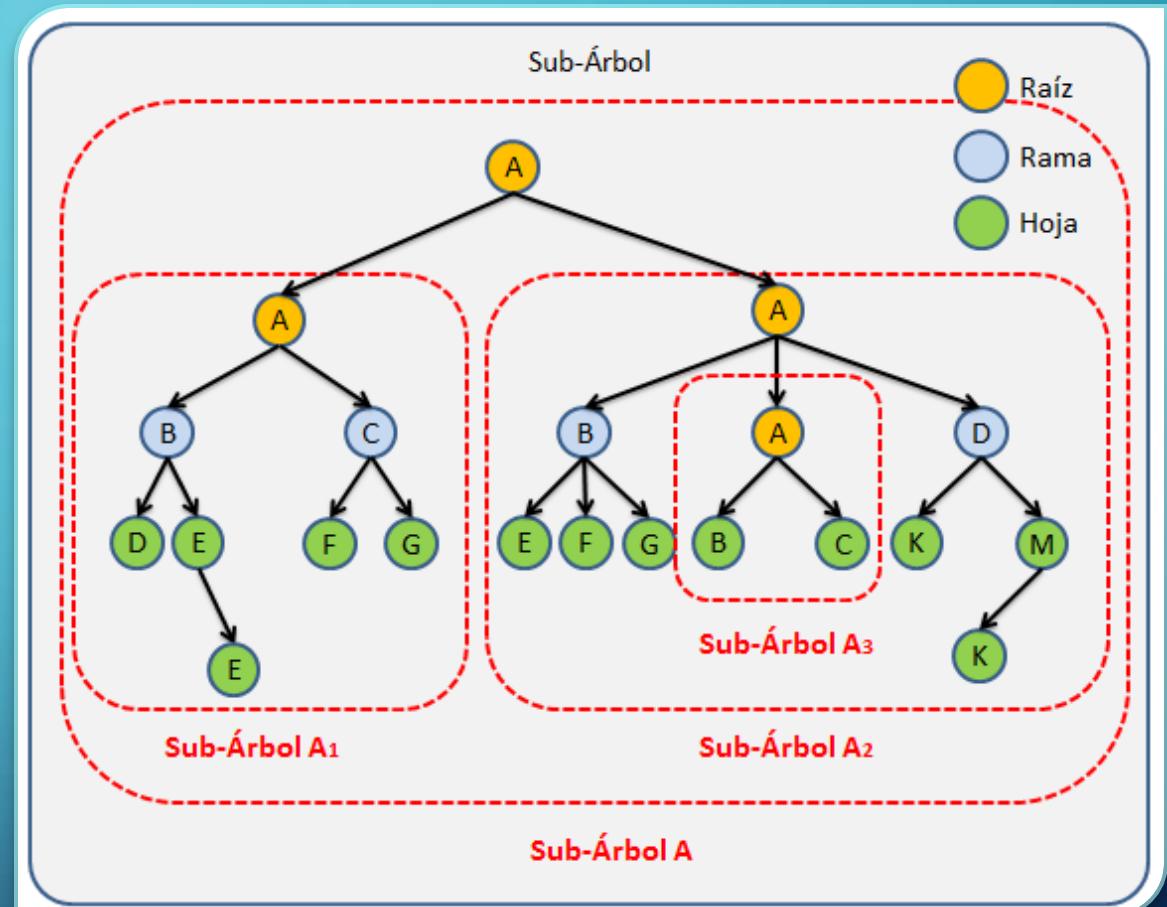
BOSQUES

Es aquella estructura que contiene normalmente un conjunto de uno o más árboles generales. Puede considerarse como una generalización de un árbol, donde en lugar de un solo nodo raíz, tenemos múltiples.



SUBÁRBOL

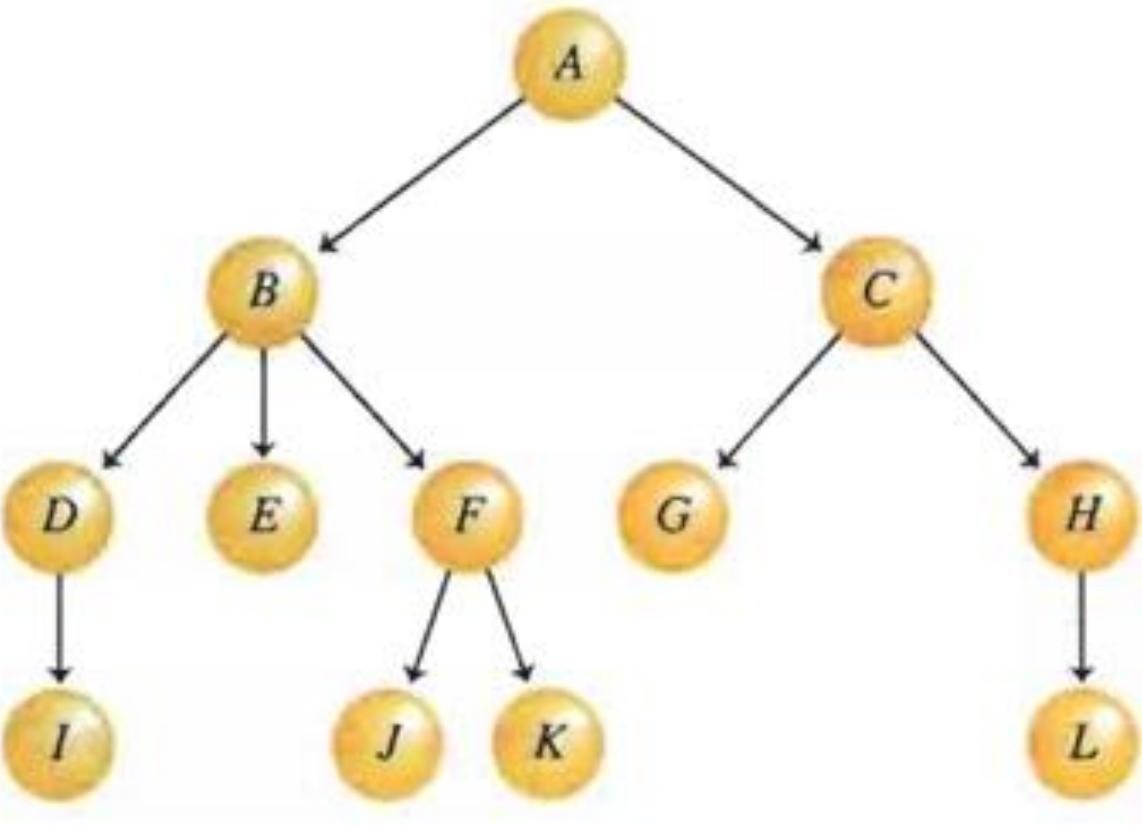
Es todo aquel árbol generado a partir de una sección determinada del árbol en cuestión. Podemos decir que un árbol es un nodo raíz con N-subárboles.



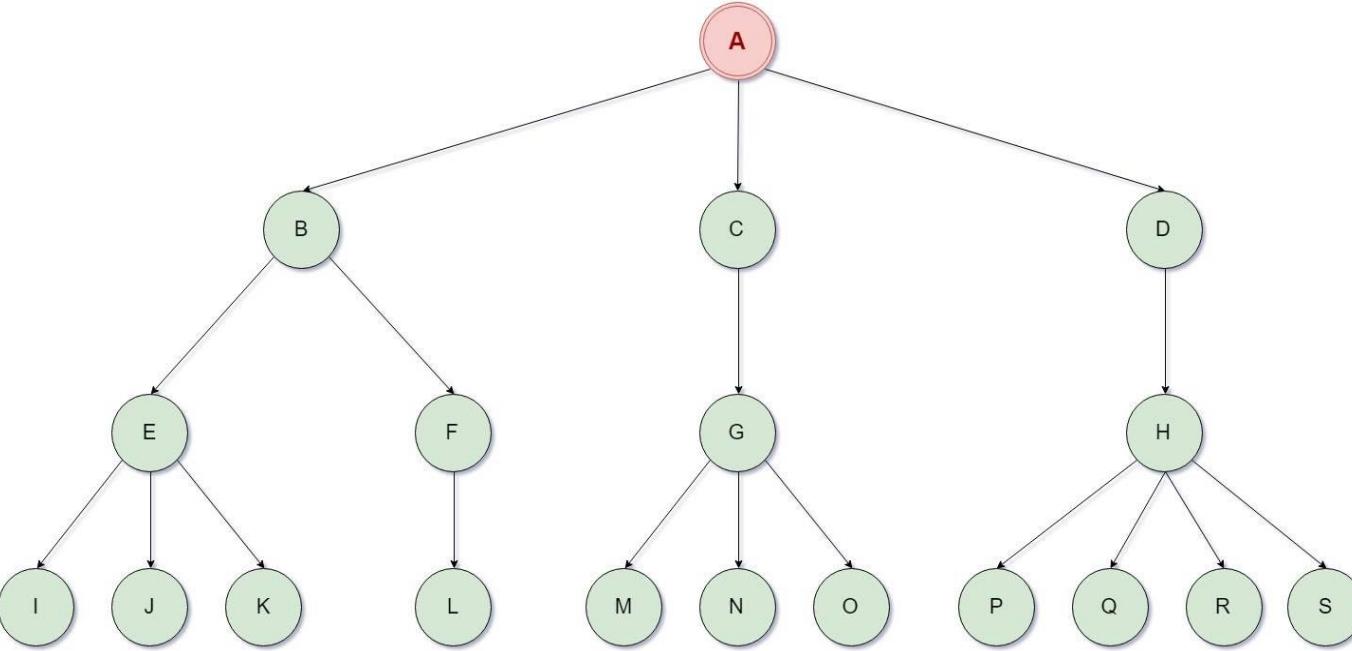
LONGITUD DE CAMINO

El número de arcos que se deben recorrer para llegar desde la raíz hasta el nodo 'X'. Por definición la raíz tiene longitud de camino de 1, sus descendientes directos longitud de camino 2 y así sucesivamente.

EJEMPLO DE LONGITUD DE CAMINO



- El nodo 'B' tiene una longitud de camino de 2.
- El nodo 'L' tiene una longitud de camino de 4.
- El nodo 'A' tiene una longitud de camino de 1.
- El nodo 'F' tiene una longitud de camino de 3.



EJERCICIO DE LONGITUD DE CAMINO

Obtener la longitud de camino de todos los nodo.

LONGITUD DE CAMINO INTERNO

Es la suma de las longitudes de camino de todos los nodos del árbol. Esta permite conocer los caminos que tiene el árbol.

$$LCI = \sum_{i=1}^h n_i * i$$

Donde:

- 'i' representa el nivel del árbol.
- 'h' su altura.
- 'n' el número de nodos en el nivel 'i'.

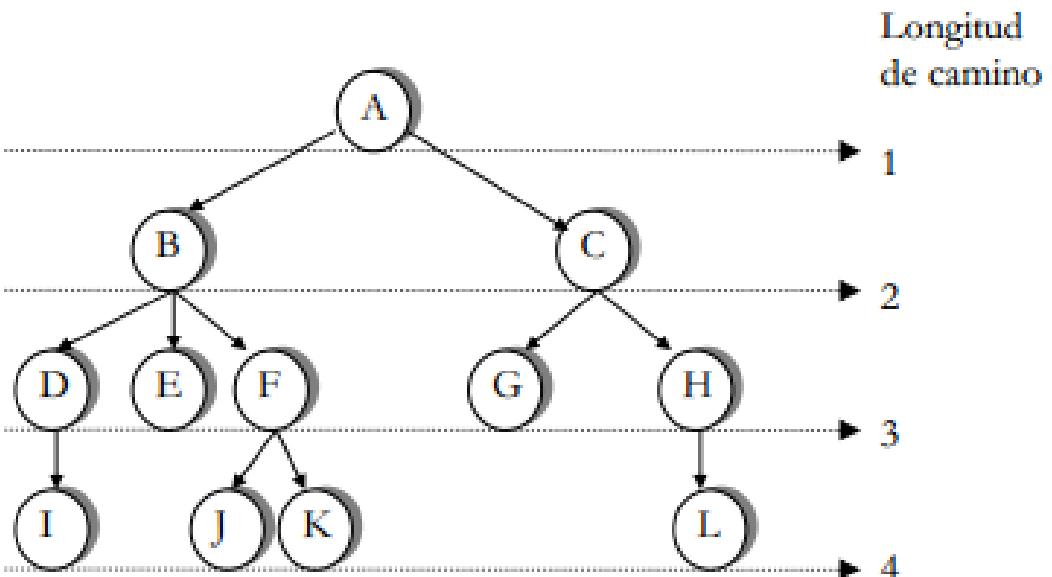


Figura 1.22 longitud de Caminos en un árbol

Longitud
de camino

$$LCI = 1 * 1 + 2 * 2 + 5 * 3 + 4 * 4$$

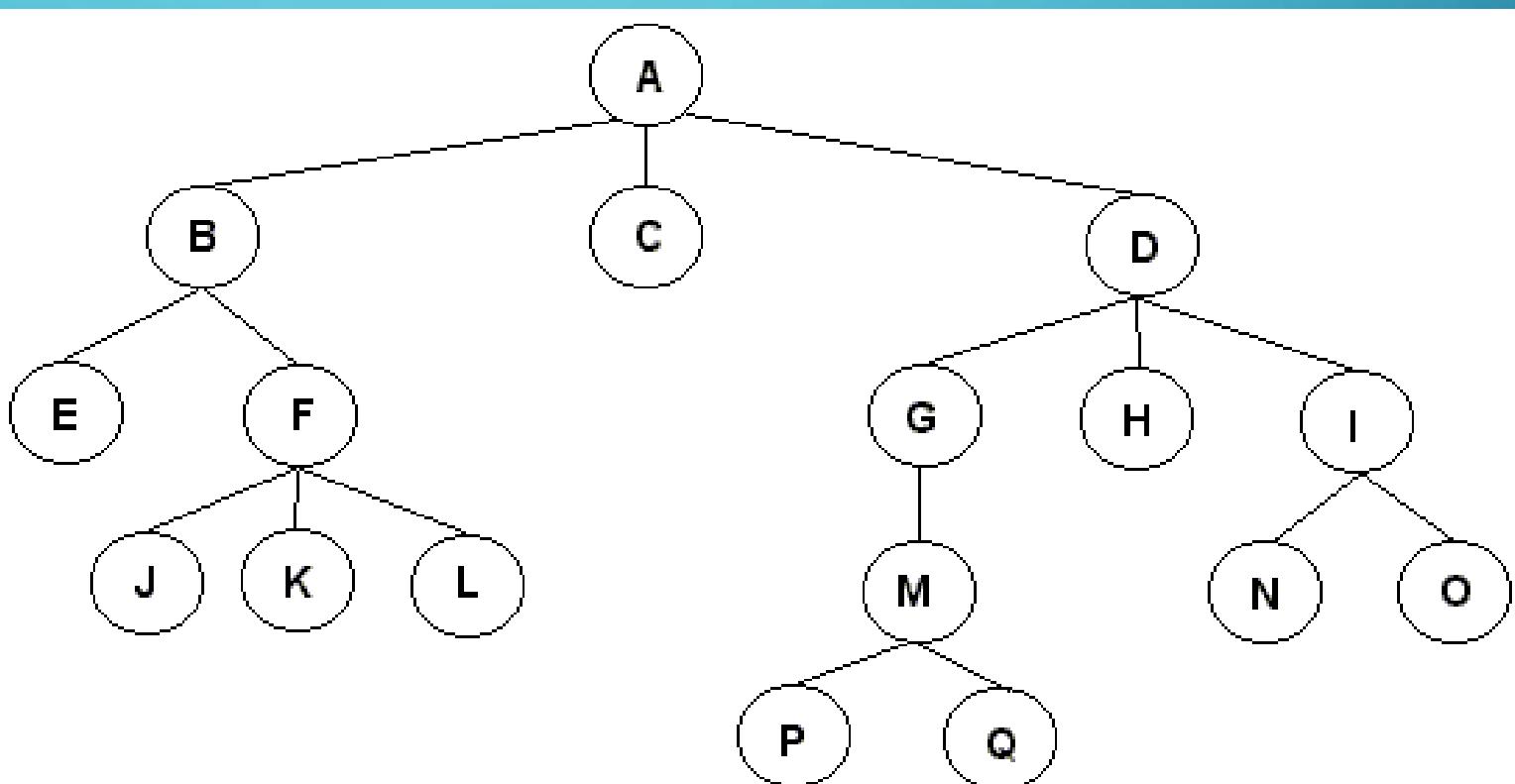
$$LCI = 1 + 4 + 15 + 16$$

$$LCI = 36$$

EJEMPLO DE LONGITUD DE CAMINO INTERNO

EJERCICIO DE LONGITUD DE CAMINO INTERNO

Calcula la longitud de camino interno del siguiente árbol:

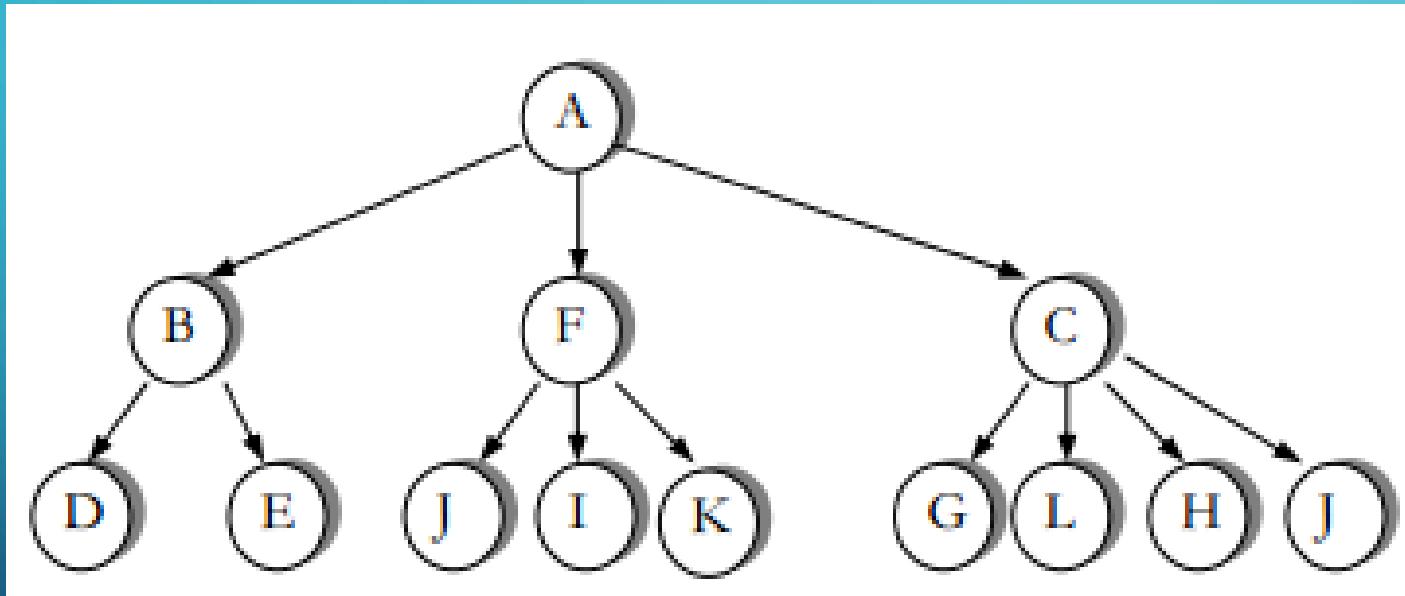


MEDIA DE LA LONGITUD DE CAMINO INTERNO

$$LCIM = \frac{LCI}{n}$$

Esta se calcula dividiendo la longitud de camino interno entre el número de nodos del árbol. Esta es importante porque permite conocer, en promedio, el número de decisiones que se deben tomar para llegar a un determinado nodo partiendo desde la raíz.

EJEMPLO DE MEDIA DE LA LONGITUD DE CAMINO INTERNO



$$LCI = \sum_{i=1}^h n_i * i$$

$$LCI = 1 * 1 + 3 * 2 + 9 * 3$$

$$LCI = 1 + 6 + 27$$

$$LCI = 34$$

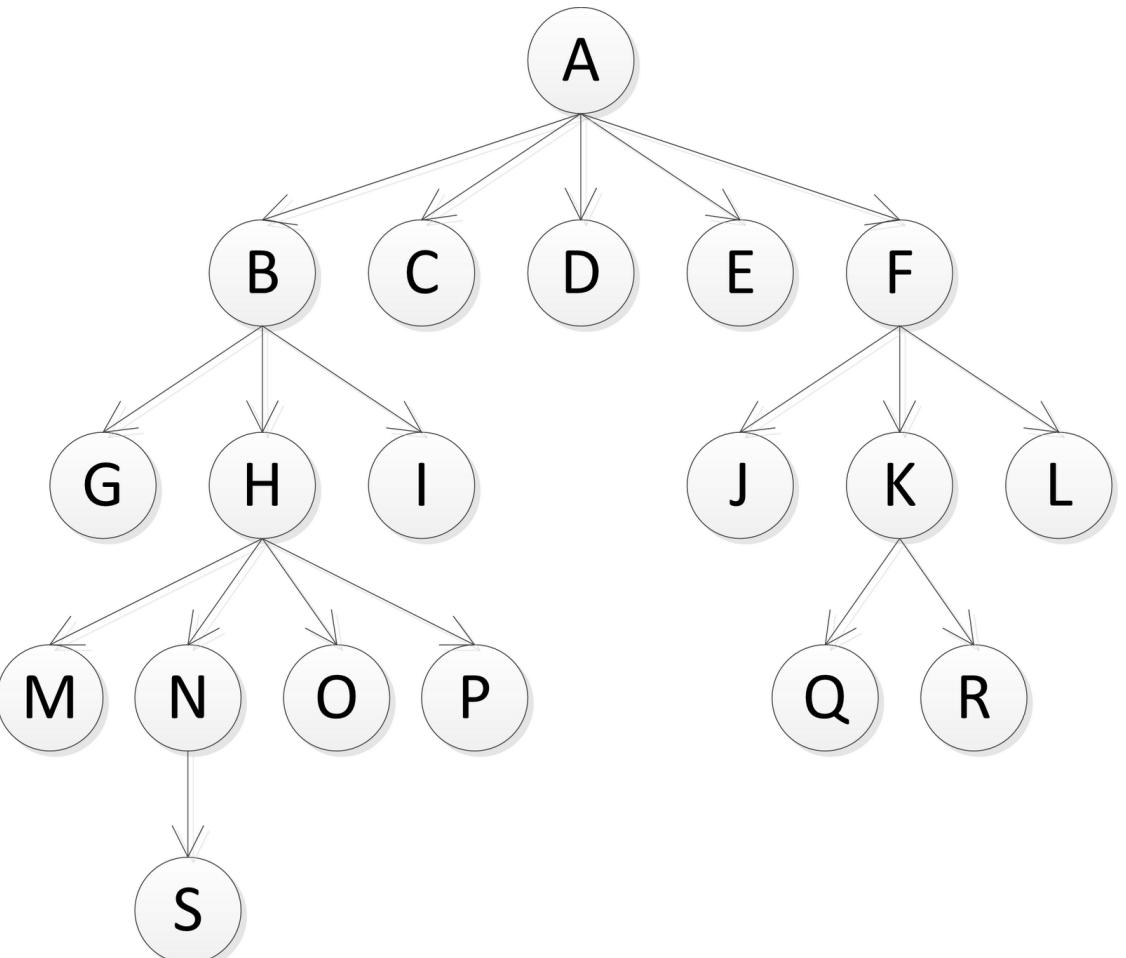
$$LCIM = \frac{LCI}{n}$$

$$LCIM = \frac{34}{13}$$

$$LCIM = 2.61538$$

EJERCICIO DE MEDIA LA DE LONGITUD DE CAMINO INTERNO

Calcula la media de la
longitud de camino
interno del siguiente
árbol:



LONGITUD DE CAMINO EXTERNO

$$LCE = \sum_{i=2}^{h+1} n_{ei} * i$$

Donde:

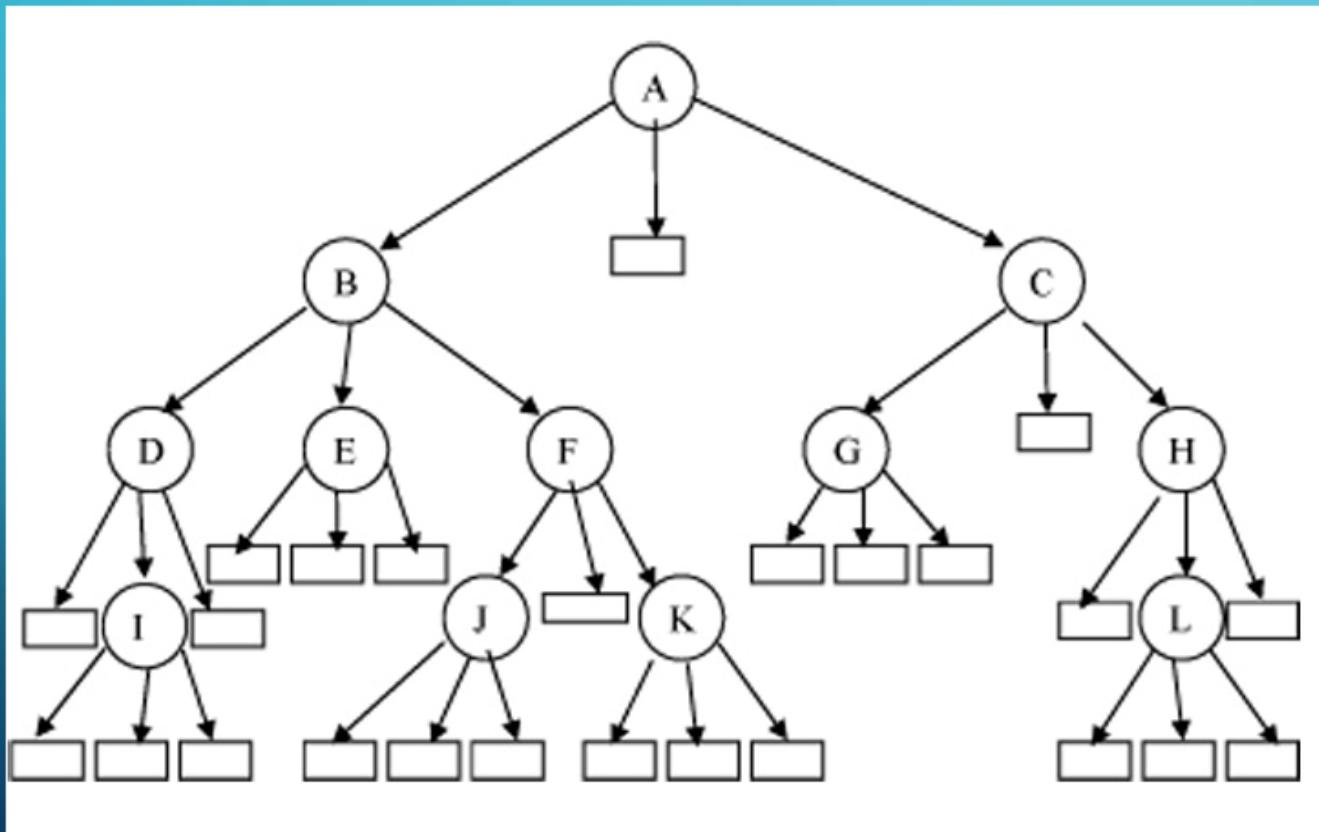
- 'i' representa el nivel del árbol.
- 'h' su altura.
- ' n_{ei} ' el número de nodos especiales en el nivel 'i'.

Se define como la suma de las longitudes de camino de todos los nodos especiales del árbol. Se calcula mediante la formula:

Observación: 'i' comienza desde 2, ya que la raíz se encuentra en el nivel 1 y no puede ser un nodo especial.

EJEMPLO DE LONGITUD DE CAMINO EXTERNO

Expresaremos el árbol en su forma extendida, para poder realizar el cálculo:



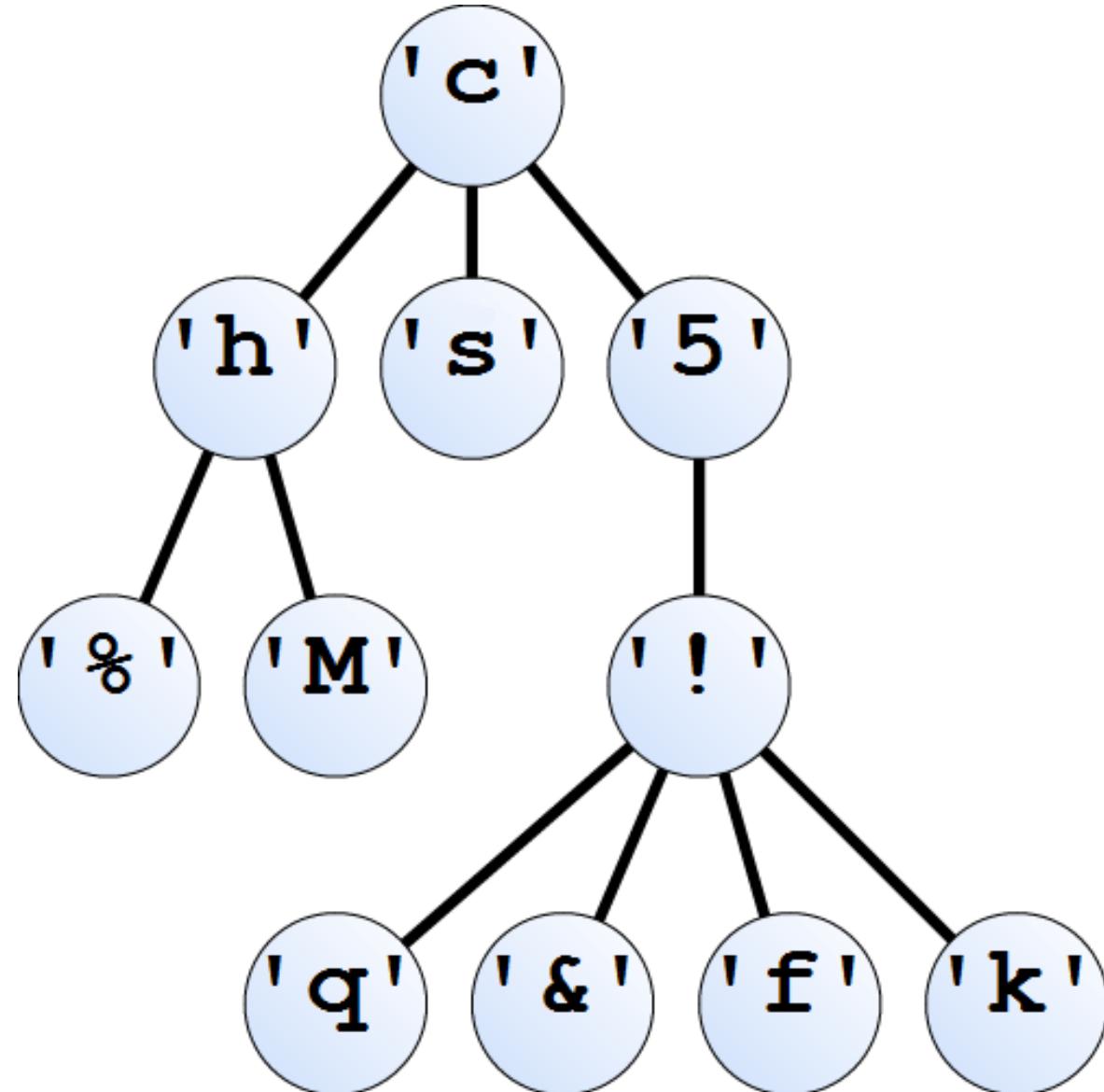
$$LCE = \sum_{i=2}^{h+1} n_{ei} * i$$

$$LCE = 1 * 2 + 1 * 3 + 11 * 4 + 12 * 5$$

$$LCE = 109$$

EJERCICIO DE LONGITUD DE CAMINO EXTERNO

Hallar la longitud de
camino externo del
siguiente árbol:



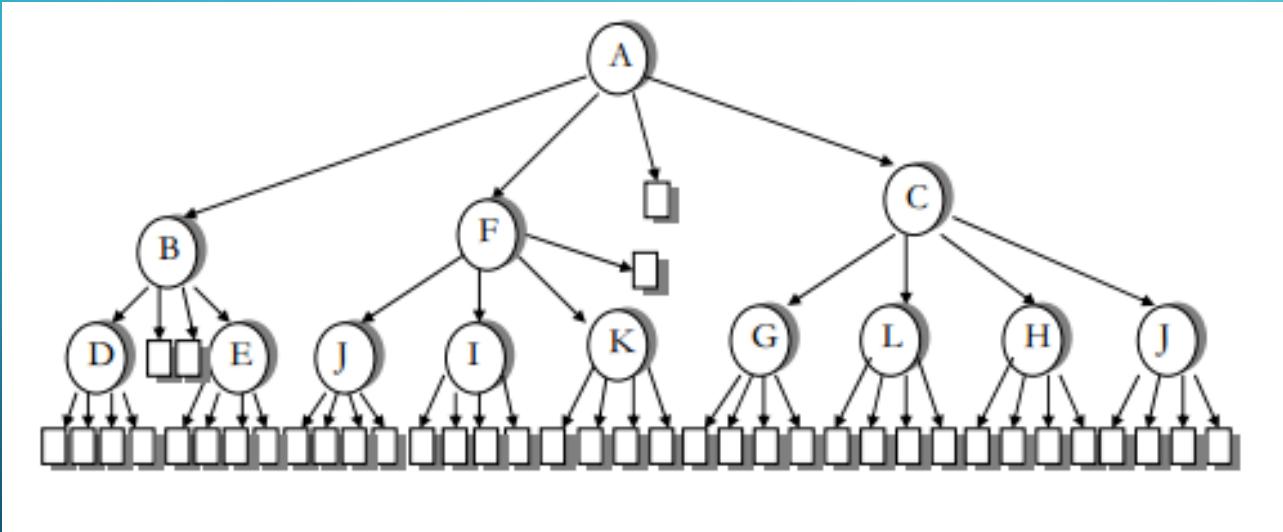
MEDIA DE LA LONGITUD DE CAMINO EXTERNO

Se calcula dividiendo la longitud de camino externo entre el número de nodos especiales del árbol. Esta indica el número de arcos que se deben recorrer en promedio para llegar, partiendo desde la raíz, a un nodo especial cualquiera del árbol.

$$LCEM = \frac{LCE}{n_e}$$

EJEMPLO DE MEDIA DE LA LONGITUD DE CAMINO EXTERNO

Expresaremos el árbol en su forma extendida, para poder realizar el cálculo:



$$LCE = \sum_{i=2}^{h+1} n_{ei} * i$$

$$LCE = 1 * 2 + 3 * 3 + 36 * 4$$

$$LCE = 155$$

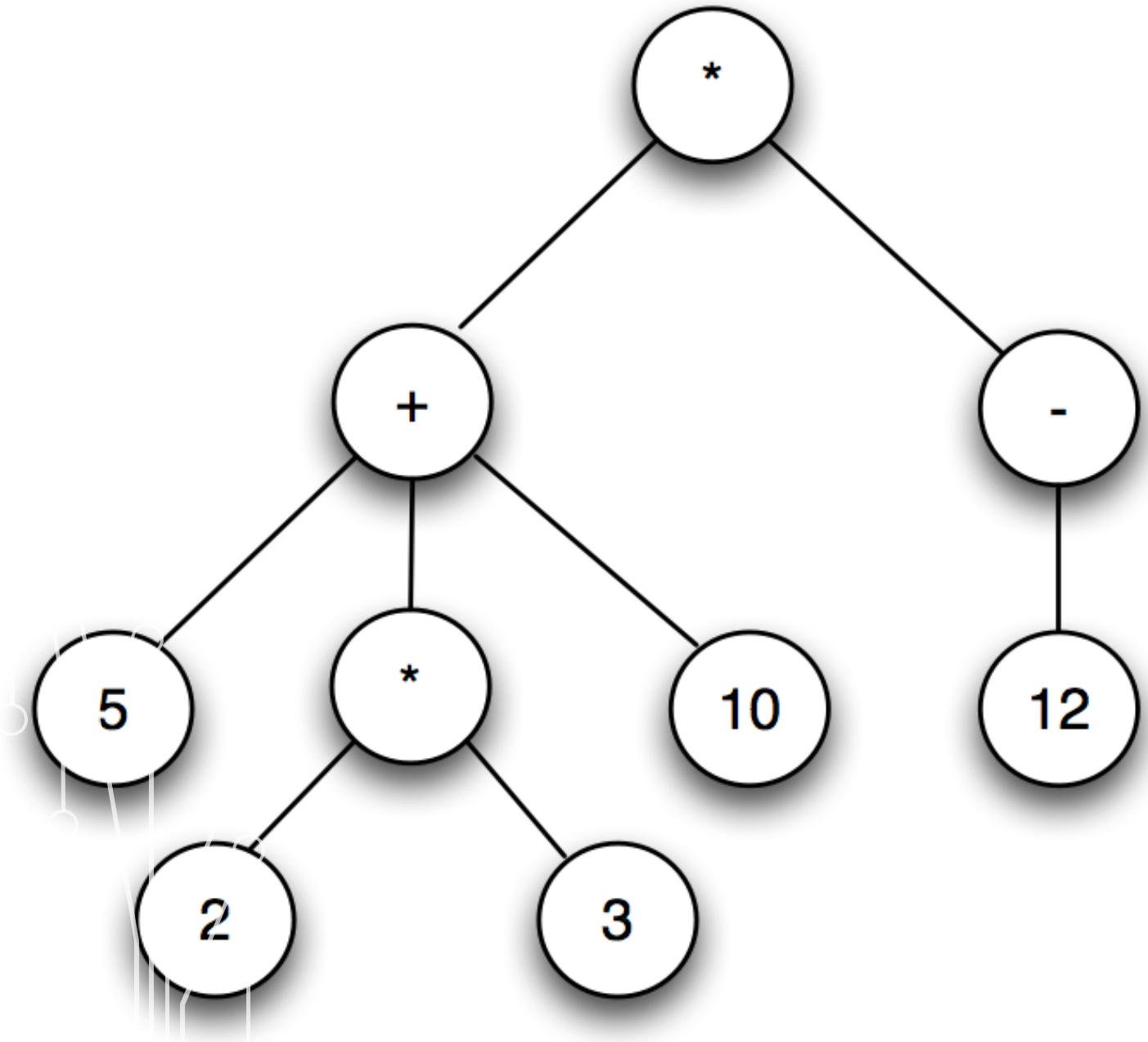
$$LCEM = \frac{LCE}{n_e}$$

$$LCEM = \frac{155}{40}$$

$$LCEM = 3.875$$

EJERCICIO DE MEDIA DE LA LONGITUD DE CAMINO EXTERNO

Calcula la media de la
longitud de camino
externo del siguiente
árbol:



REPRESENTACIONES DE UN ÁRBOL

Es posible representar un árbol de diferentes formas y todas ellas se consideran equivalentes.

Estas son:

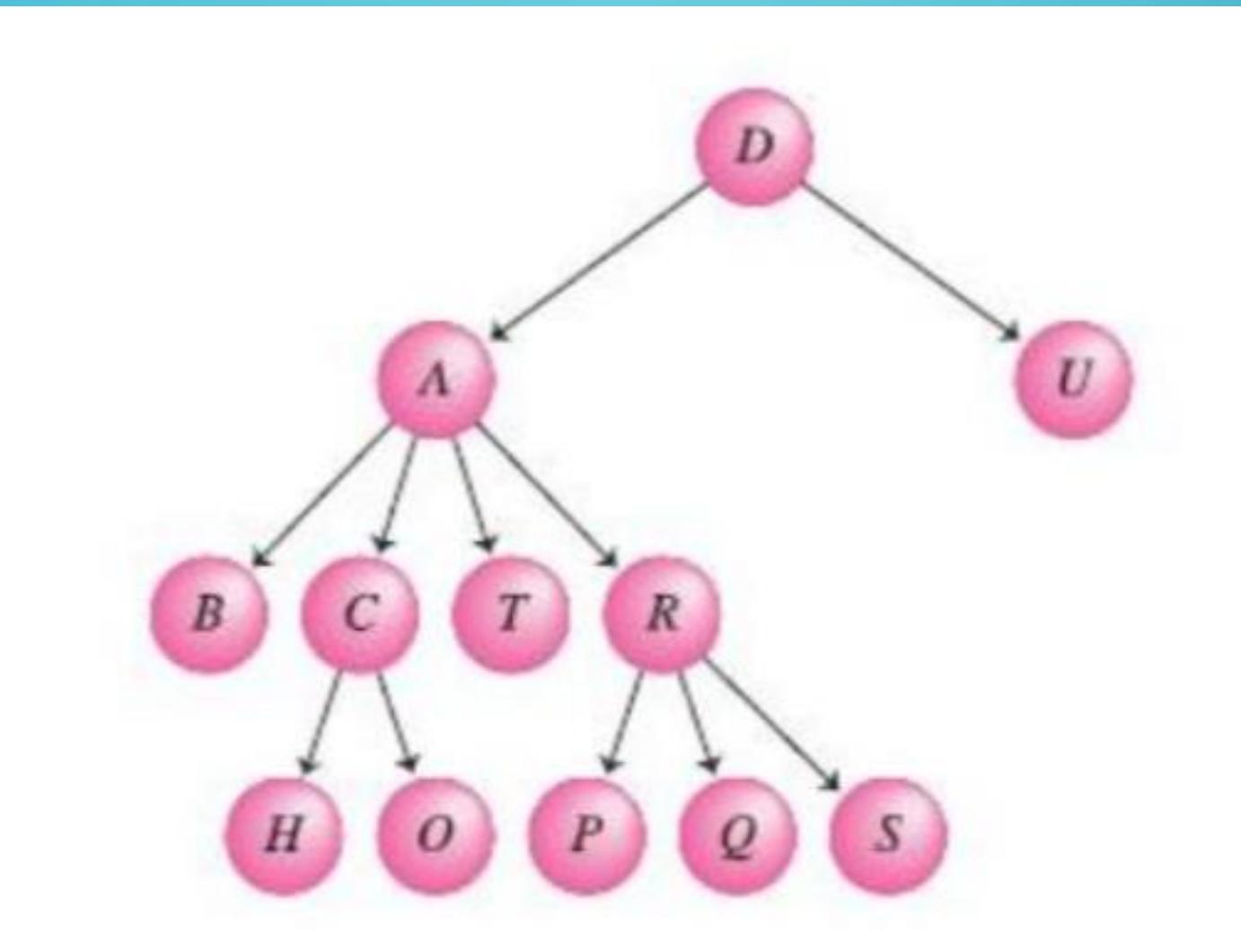
- Diagrama de Venn.
- Notación de Dewey.
- De listas de paréntesis.
- Indentada.
- Niveles de profundidad.

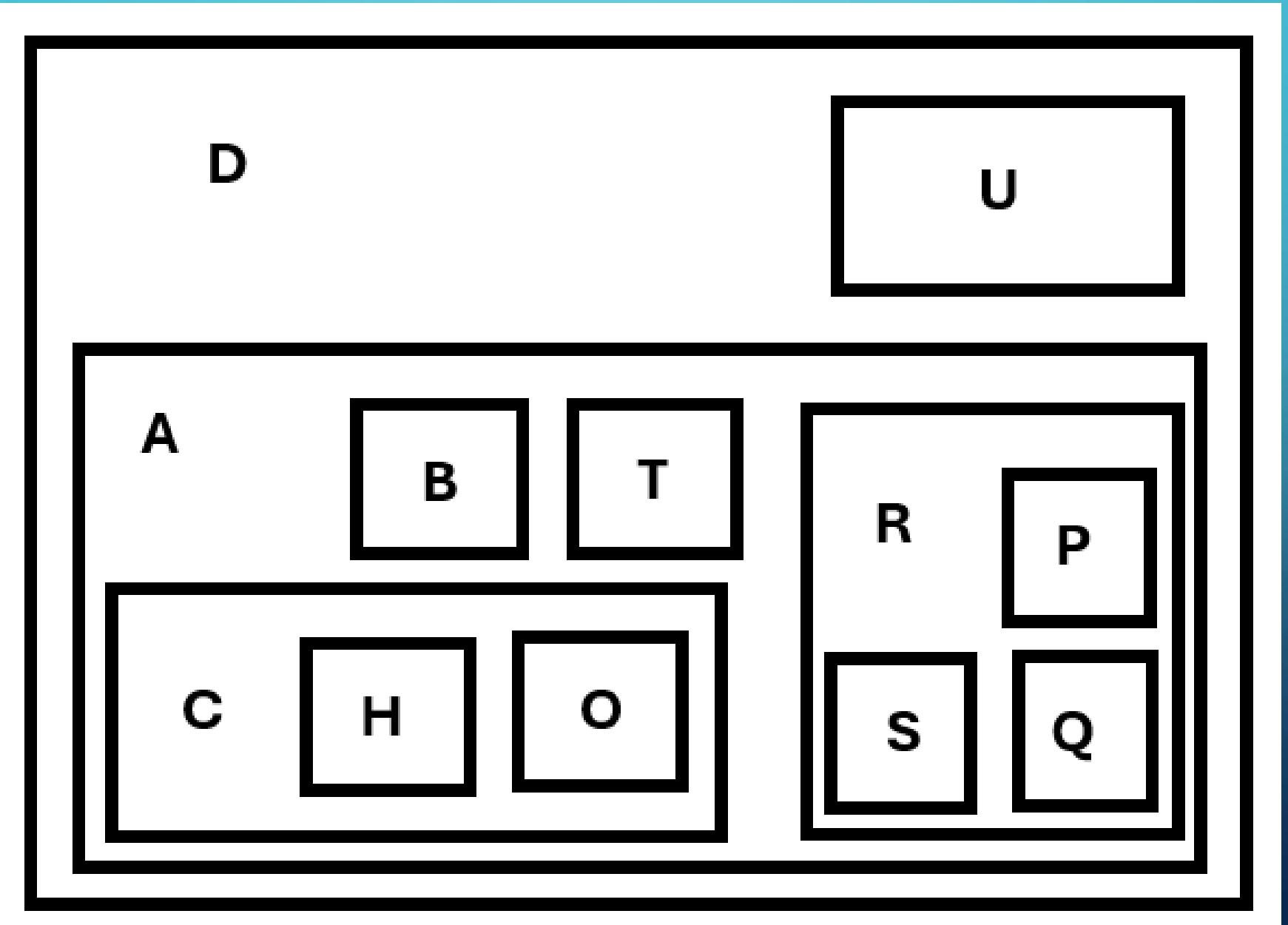
REPRESENTACIÓN DE UN ÁRBOL A UN DIAGRAMA DE VENN

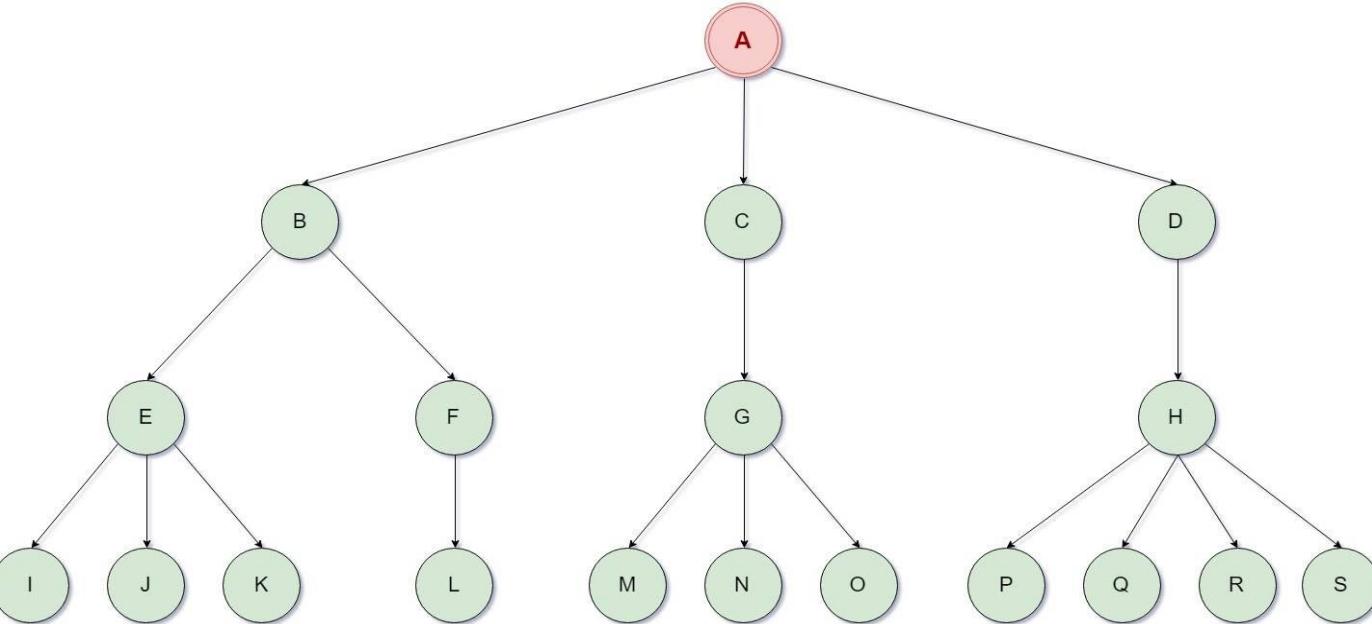
Pasos que seguir para representar un árbol a un diagrama de Venn.

1. Identificar el nodo raíz y considerarlo como el universo en el diagrama de Venn, representándolo con un círculo grande.
2. Representar los nodos hijos del nodo raíz como círculos dentro del círculo que representa el universo.
3. Para cada nodo hijo, dibujar círculos dentro de él para representar a sus hijos, siguiendo el mismo patrón. Si un nodo no tiene hijos, no se dibujará nada dentro de su círculo.
4. Colocar etiquetas en cada círculo para indicar claramente a qué nodo corresponde cada subconjunto.
5. Comprobar la jerarquía del diagrama de Venn con el árbol para asegurarse de que las relaciones de subconjunto sean correctas.

EJEMPLO DE REPRESENTACIÓN DE UN ÁRBOL A UN DIAGRAMA DE VENN







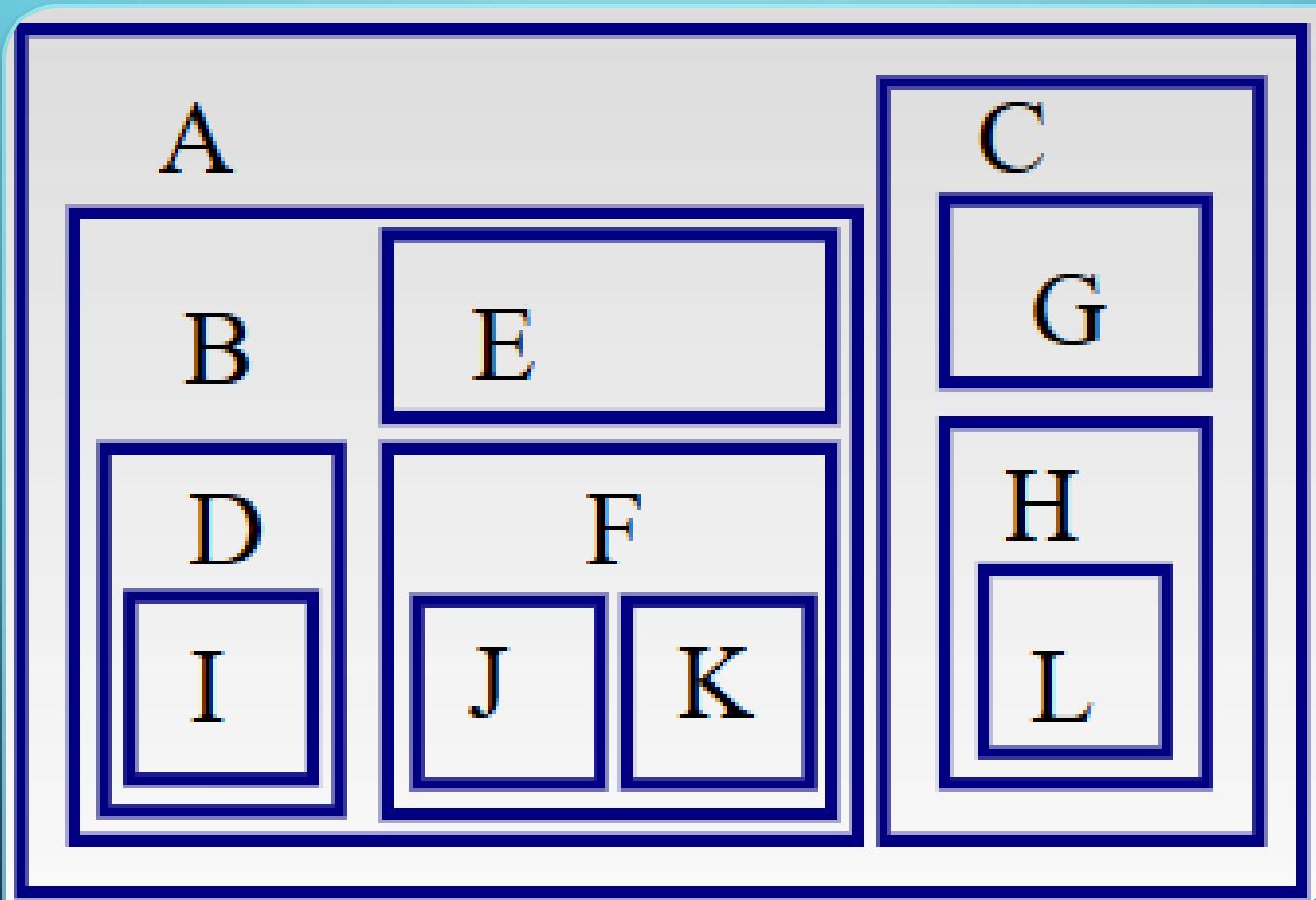
EJERCICIO DE REPRESENTACIÓN DE UN ÁRBOL A UN DIAGRAMA DE VENN

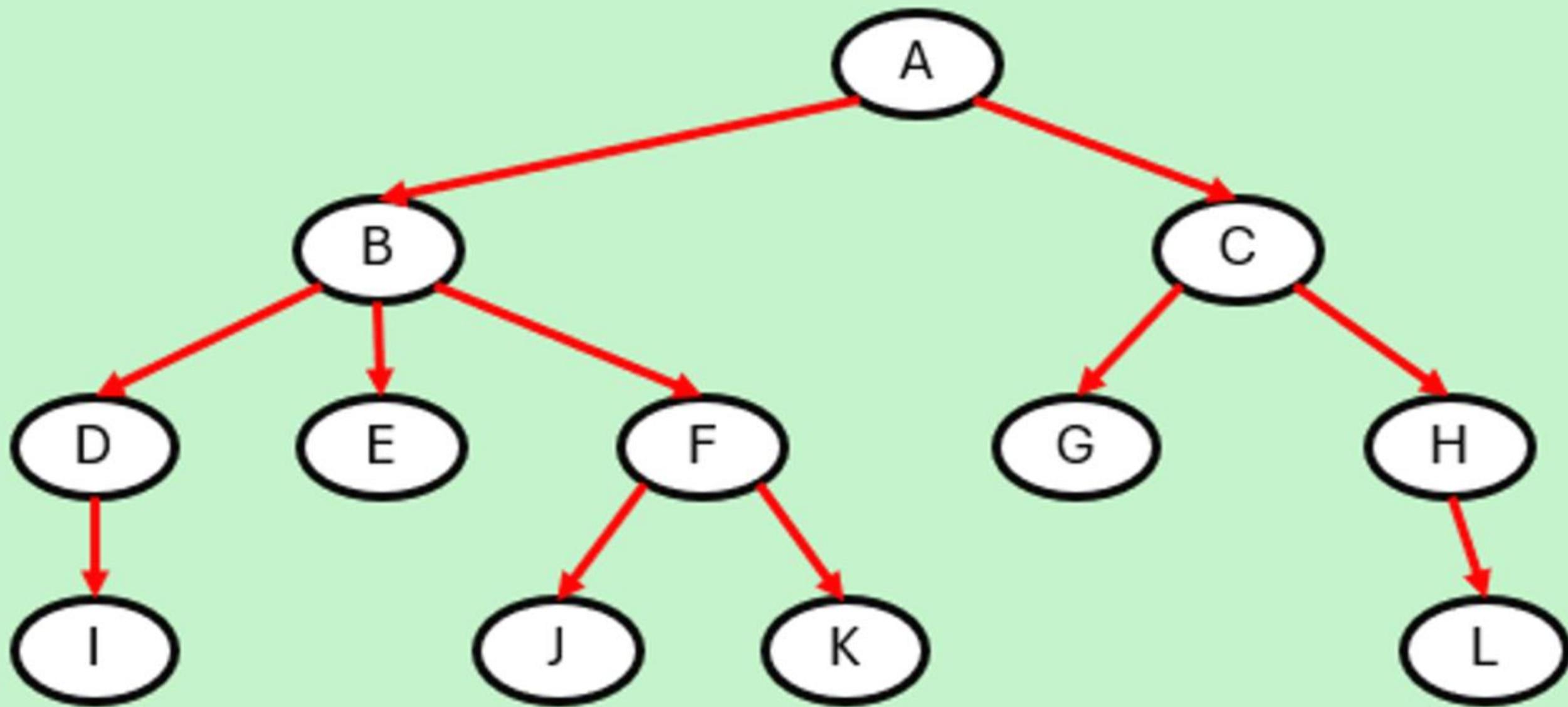
REPRESENTACIÓN DE UN DIAGRAMA DE VENN A UN ÁRBOL

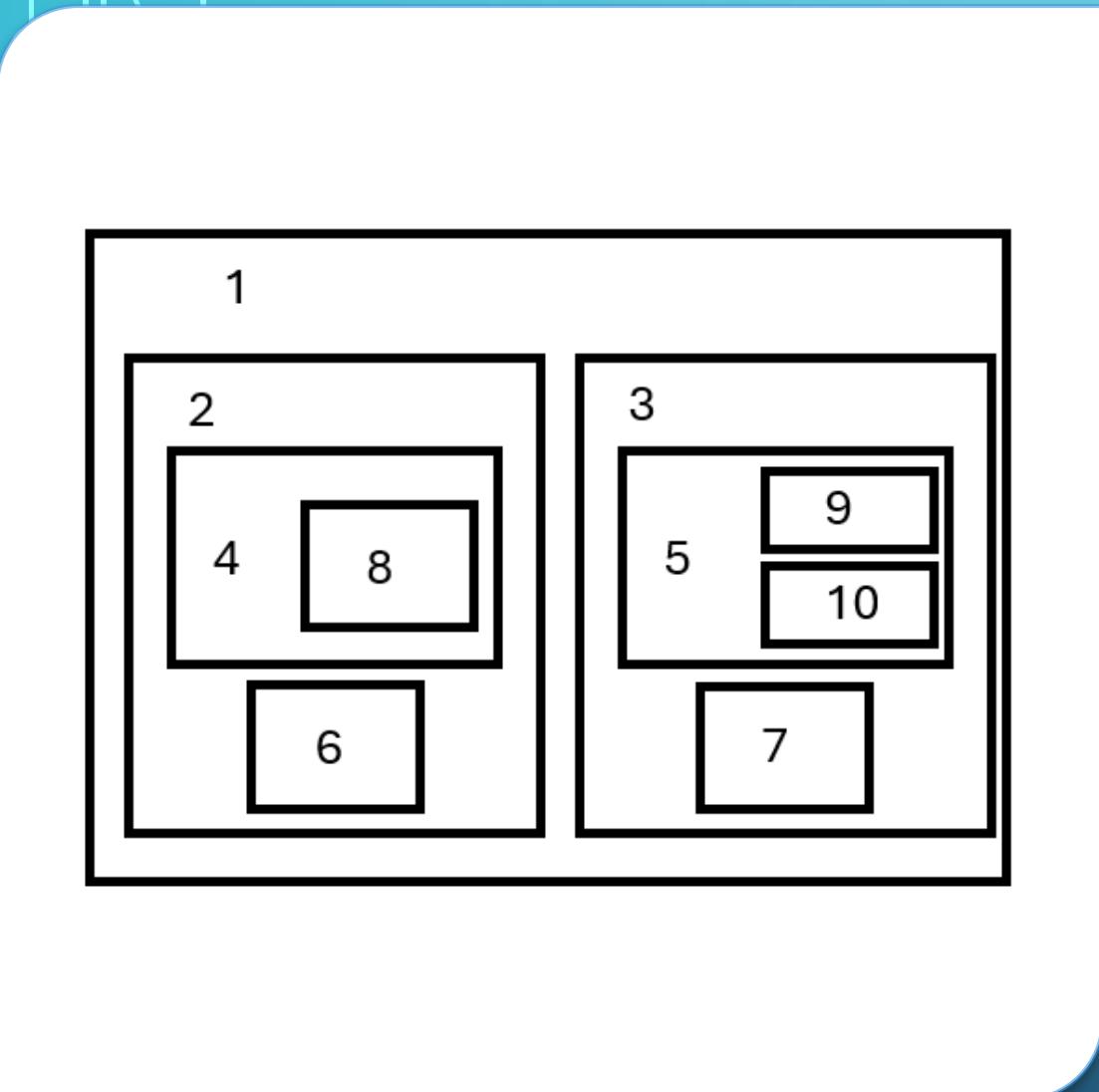
Pasos que seguir para representar un diagrama de Venn en un árbol.

1. Identificar el universo del diagrama de Venn, el cual representara el nodo raíz del árbol.
2. Establecer la jerarquía de subconjuntos representando los círculos dentro del universo, donde los círculos completamente dentro de otros corresponden a nodos hijos.
3. Asignar nodos a los subconjuntos del diagrama de Venn, transformando cada subconjunto en un nodo del árbol, siguiendo las relaciones de inclusión.
4. Crear nodos hijos conectándolos a sus respectivos nodos padres para formar la estructura jerárquica del árbol.
5. Eliminar intersecciones, ya que en el árbol no se representan, manteniendo solo las relaciones de inclusión directa entre los subconjuntos.
6. Asignar etiquetas a los nodos del árbol para identificar claramente qué subconjunto representa cada uno.
7. Comprobar la estructura verificando que la jerarquía del árbol coincida con las relaciones del diagrama de Venn.

EJEMPLO DE UNA REPRESENTACIÓN DE UN DIAGRAMA DE VENN A UN ÁRBOL







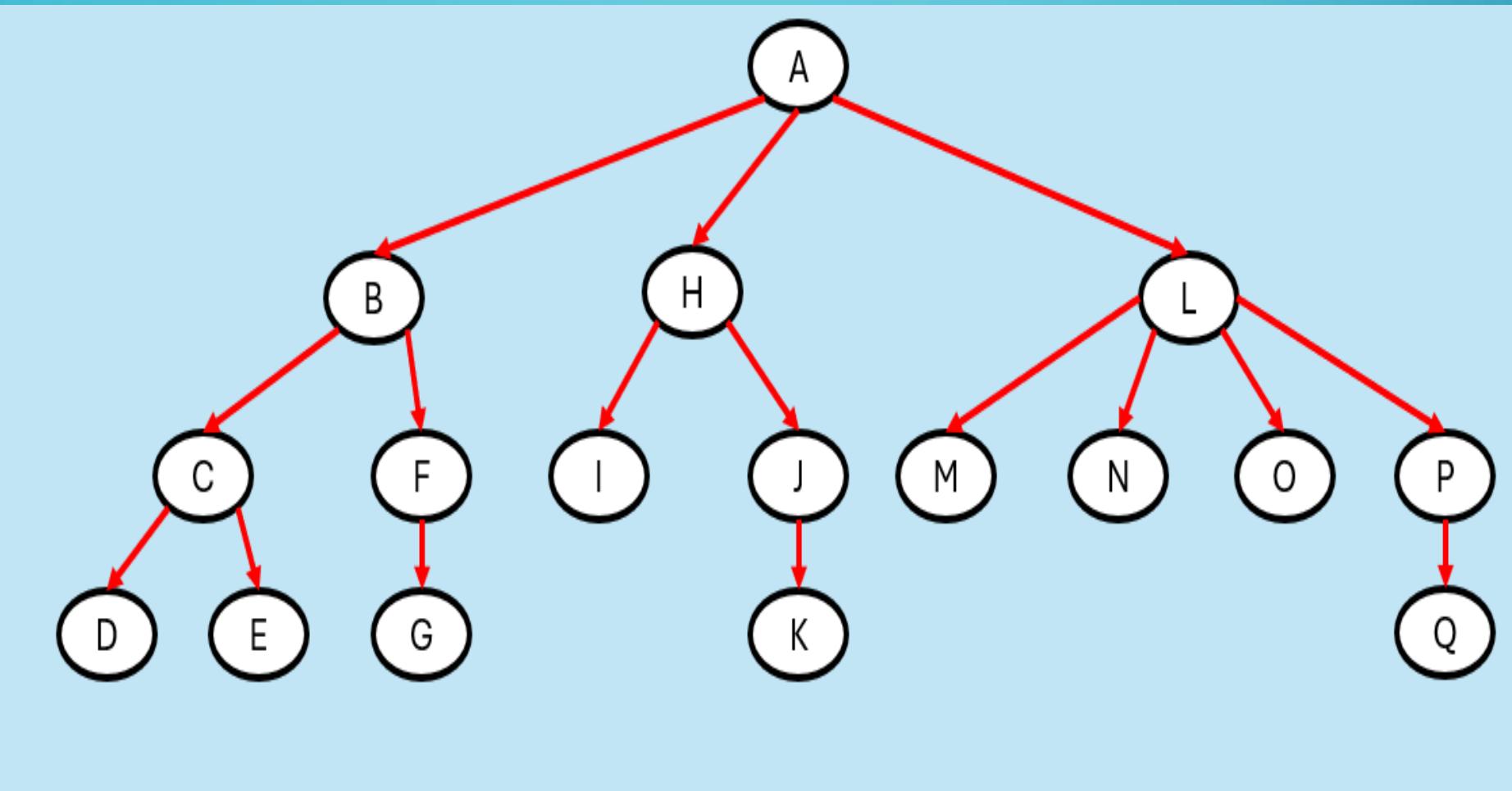
EJERCICIO UNA DE
REPRESENTACIÓN DE UN
DIAGRAMA DE VENN A
UN ÁRBOL

REPRESENTACIÓN DE UN ÁRBOL A LA NOTACIÓN DEWEY

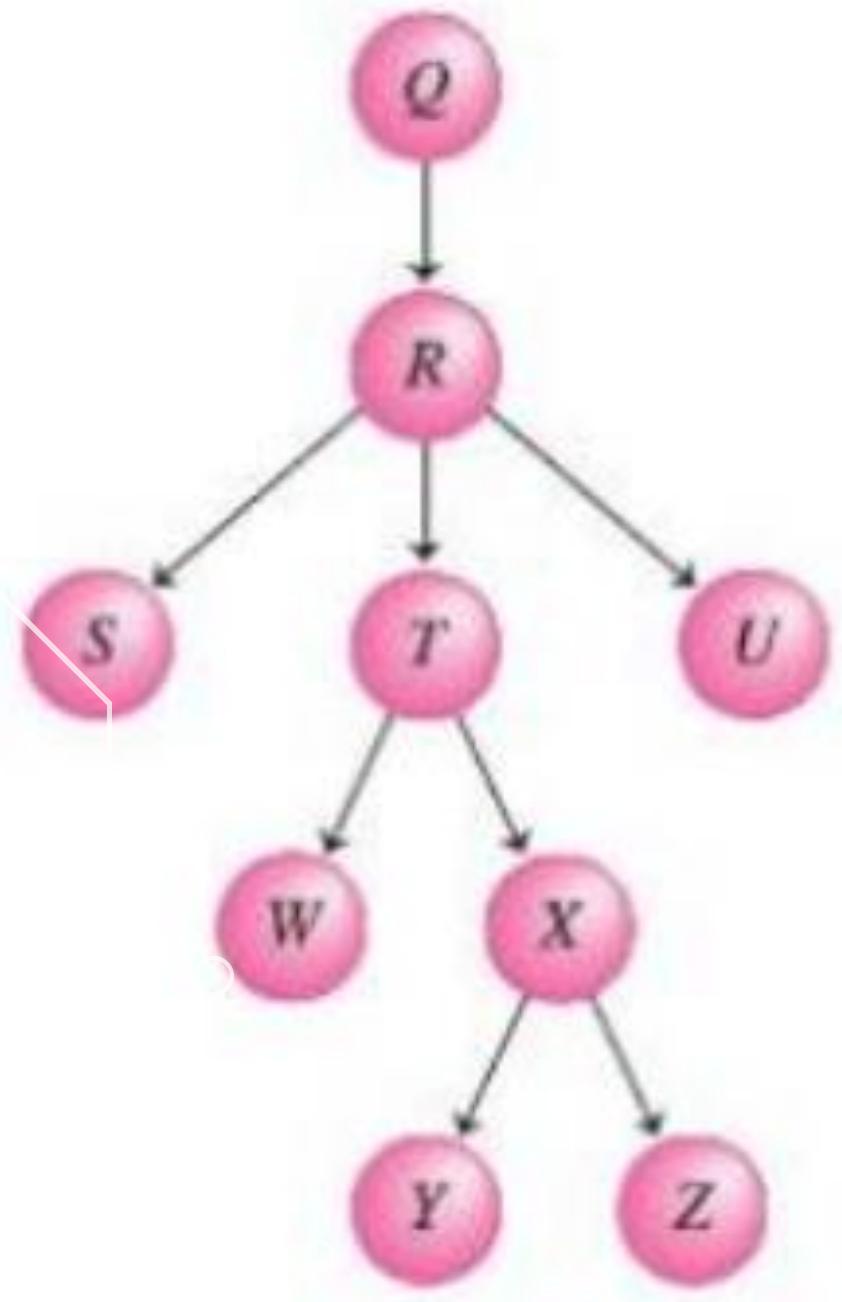
Pasos que seguir para representar un árbol a la notación de Dewey.

1. Identifica el nodo raíz y asígnale un numero (ejemplo: "1")
2. Asigna números secuenciales a los hijos del nodo raíz, separados por puntos (ejemplo: "1.1", "1.2").
3. Asigna números a los hijos de los nodos, siguiendo el mismo patrón(ejemplo: "1.1.1")
4. Verifica que los números reflejen correctamente la jerarquía del árbol.
5. Para armar la notación Dewey, comienza con el nodo raíz seguido de una coma, luego escribe su hijo más cercano con su número y etiqueta (por ejemplo, "1.1: B"). Continúa agregando los demás hijos de forma secuencial, manteniendo el mismo patrón de numeración (por ejemplo, "1.1.1: C, 1.1.1.1: D, 1.1.2: E") hasta llegar a la última hoja del árbol. Así de esa forma hasta terminar con el ultimo hijo del nodo raíz.

EJEMPLO DE REPRESENTACIÓN DE UN ÁRBOL A LA NOTACIÓN DEWEY



1.A, 1.1.B, 1.1.1.C, 1.1.1.1.D,
1.1.1.2.E, 1.1.2.F, 1.1.2.1.G, 1.2.H,
1.2.1.I, 1.2.2.J, 1.2.2.1.K, 1.3.L,
1.3.1.M, 1.3.2.N, 1.3.3.O, 1.3.4.P,
1.3.4.1.Q.



EJERCICIO DE REPRESENTACIÓN DE UN ÁRBOL A LA NOTACIÓN DEWEY

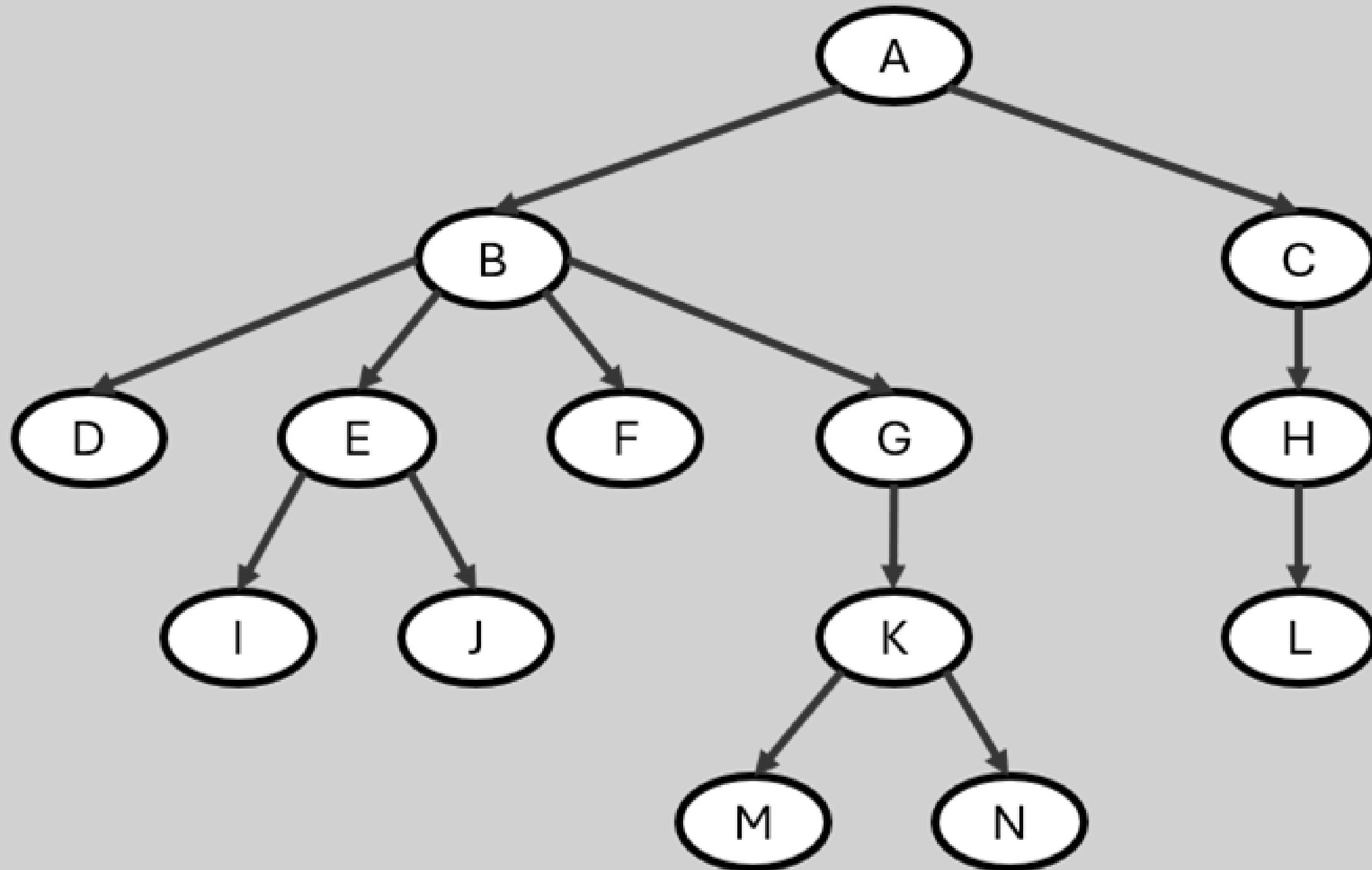
REPRESENTACIÓN DE LA NOTACIÓN DE DEWEY A ÁRBOL

Pasos que seguir para representar la notación de Dewey a un árbol.

1. Identifica el nodo raíz localizando el número más simple (generalmente "1") en la notación de Dewey.
2. Crea el nodo raíz y asígnale la etiqueta correspondiente.
3. Analiza la jerarquía de los nodos observando los números secuenciales y los puntos para entender la estructura.
4. Construye los hijos del nodo raíz buscando los nodos con la notación "1.x" y agrégalos de forma secuencial.
5. Añade los subnodos recursivamente siguiendo la notación "1.x.y" para cada nivel jerárquico.
6. Verifica la jerarquía asegurándote de que los nodos siguen el orden secuencial correcto.

EJEMPLO DE REPRESENTACIÓN DE LA NOTACIÓN DE DEWEY A ÁRBOL

1.A, 1.1.B, 1.1.1.D, 1.1.2.E,
1.1.2.1.Y, 1.1.2.2.J, 1.1.3.F, 1.1.4.G,
1.1.4.1.K, 1.1.4.1.1.M, 1.1.4.1.2.N,
1.2.C, 1.2.1.H, 1.2.1.1.L



EJERCICIO DE REPRESENTACIÓN DE LA NOTACIÓN DE DEWEY A ÁRBOL

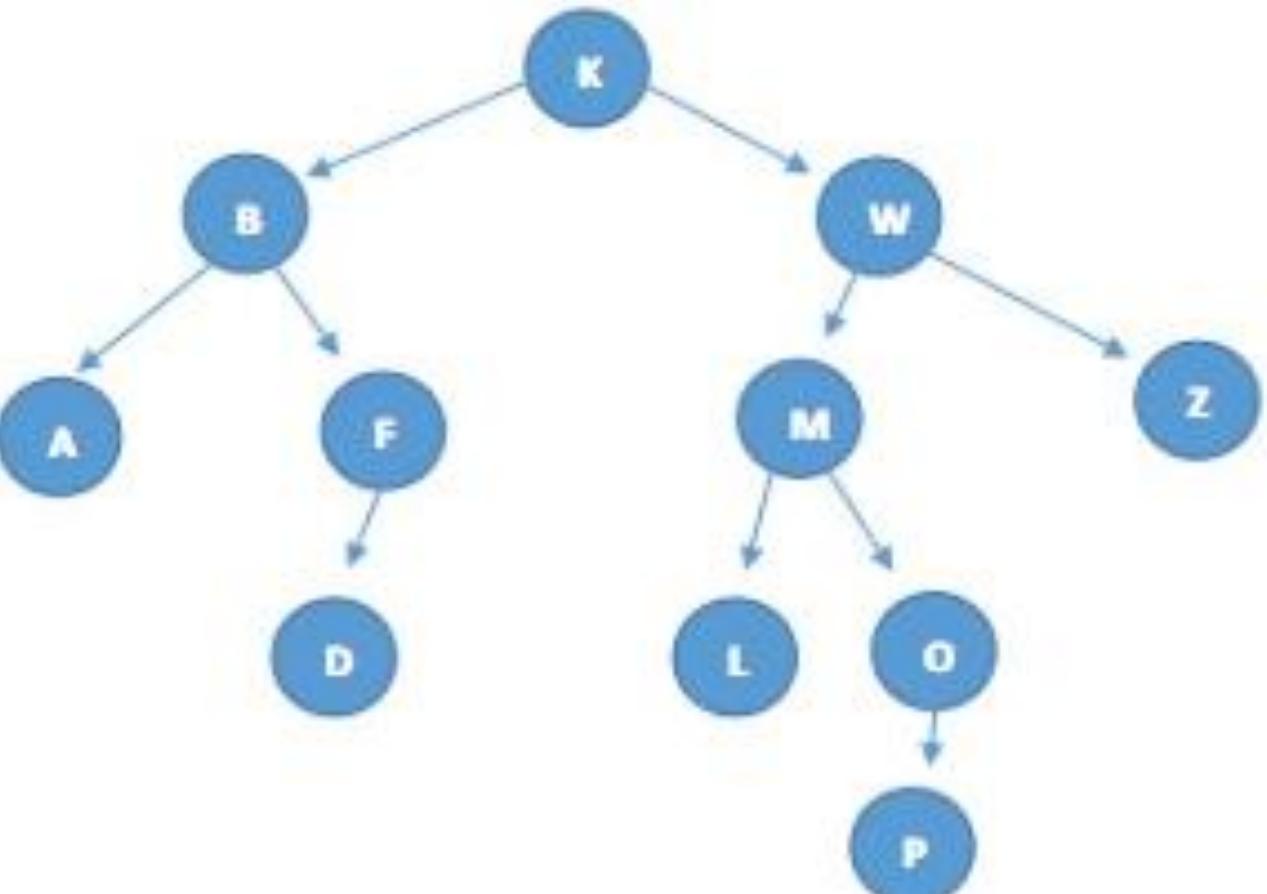
1.Z, 1.1.A, 1.1.1.B, 1.1.1.1.C, 1.1.1.1.1.D,
1.1.1.1.2.E, 1.1.2.F, 1.1.2.1.G, 1.1.2.1.1.H,
1.1.2.1.2.I, 1.1.3.J, 1.1.3.1.K, 1.1.3.1.1.L,
1.1.3.1.1.1.M, 1.2.N, 1.2.1.O, 1.2.1.1.P,
1.2.1.1.1.Q, 1.2.1.1.1.R

REPRESENTACIÓN DE ÁRBOL A LA DE LISTA DE PARÉNTESIS

Pasos que seguir para representar un árbol a la de lista de paréntesis.

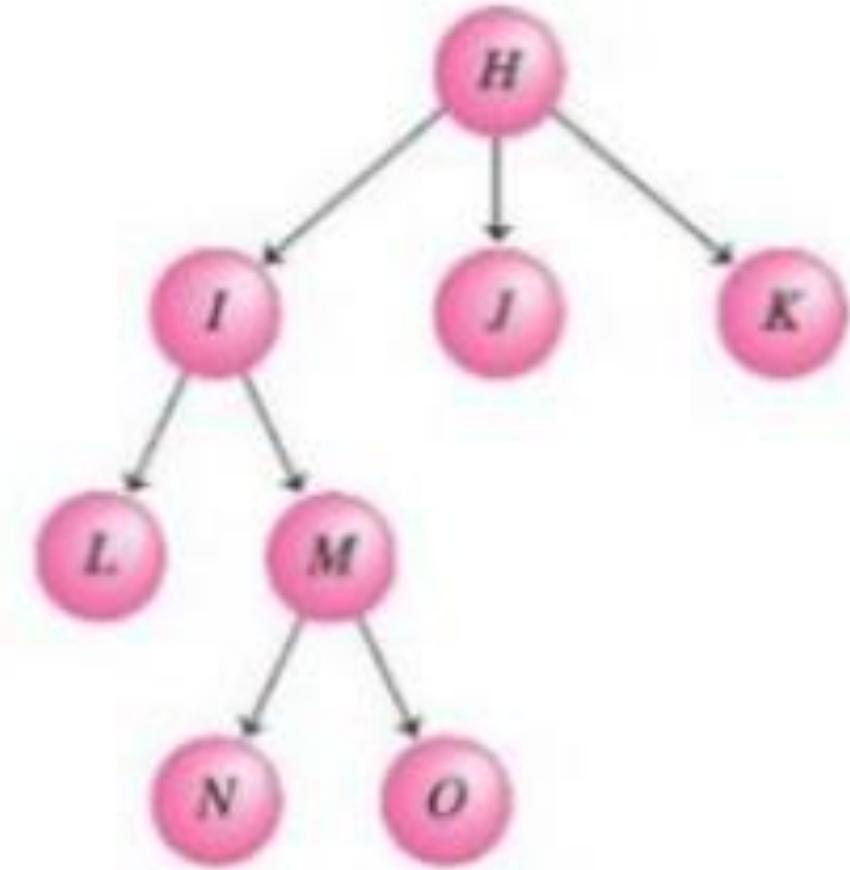
1. Abrir paréntesis e iniciar con el nodo raíz y escribir su etiqueta para representar el punto de partida del árbol.
2. Si el nodo tiene hijos, abrir un paréntesis y escribir las etiquetas de los hijos en el orden en que aparecen en el árbol.
3. Para cada hijo que tenga descendientes, repetir el proceso de abrir paréntesis y listar sus hijos, cerrando el paréntesis cuando se hayan incluido todos.
4. Continuar este proceso recursivamente hasta que todos los nodos y sus relaciones hayan sido representados.
5. Cerrar todos los paréntesis abiertos y revisar que la estructura mantenga la jerarquía correcta del árbol.

EJEMPLO DE REPRESENTACIÓN DE ÁRBOL A LA DE LISTA DE PARÉNTESIS



(K (B (A,F(D))), W (M (L,O(P)),Z)))

EJERCICIO DE REPRESENTACIÓN DE ÁRBOL A LA DE LISTA DE PARÉNTESIS



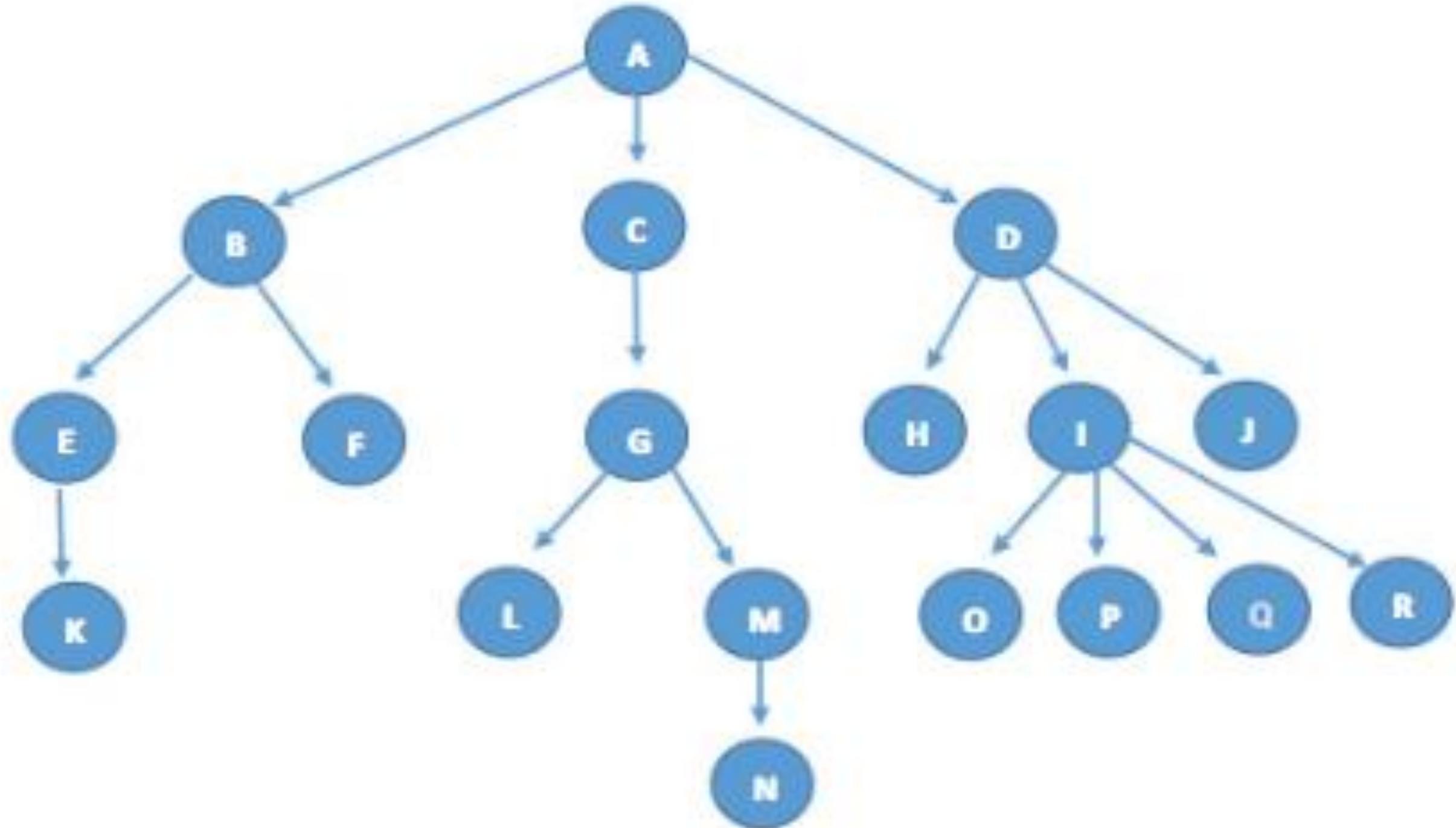
REPRESENTACIÓN DE LISTA DE PARÉNTESIS A ÁRBOL

Pasos que seguir para representar una lista de paréntesis a árbol.

1. Identificar la etiqueta inicial fuera de los paréntesis y asignarla como el nodo raíz del árbol.
2. Leer de izquierda a derecha, creando nodos hijos cada vez que se abre un paréntesis después de una etiqueta.
3. Agregar los nodos hijos bajo el nodo actual y mover el foco hacia el último hijo añadido.
4. Si aparece un paréntesis de cierre, regresar el foco al nodo padre más cercano.
5. Continuar hasta que todos los nodos hayan sido procesados y la jerarquía esté completa.

EJEMPLO DE REPRESENTACIÓN DE LISTA DE PARÉNTESIS A ÁRBOL

(A(B(E(K),F), C(G(L,M(N)))), D(H,I,(O,P,Q,R),J)))



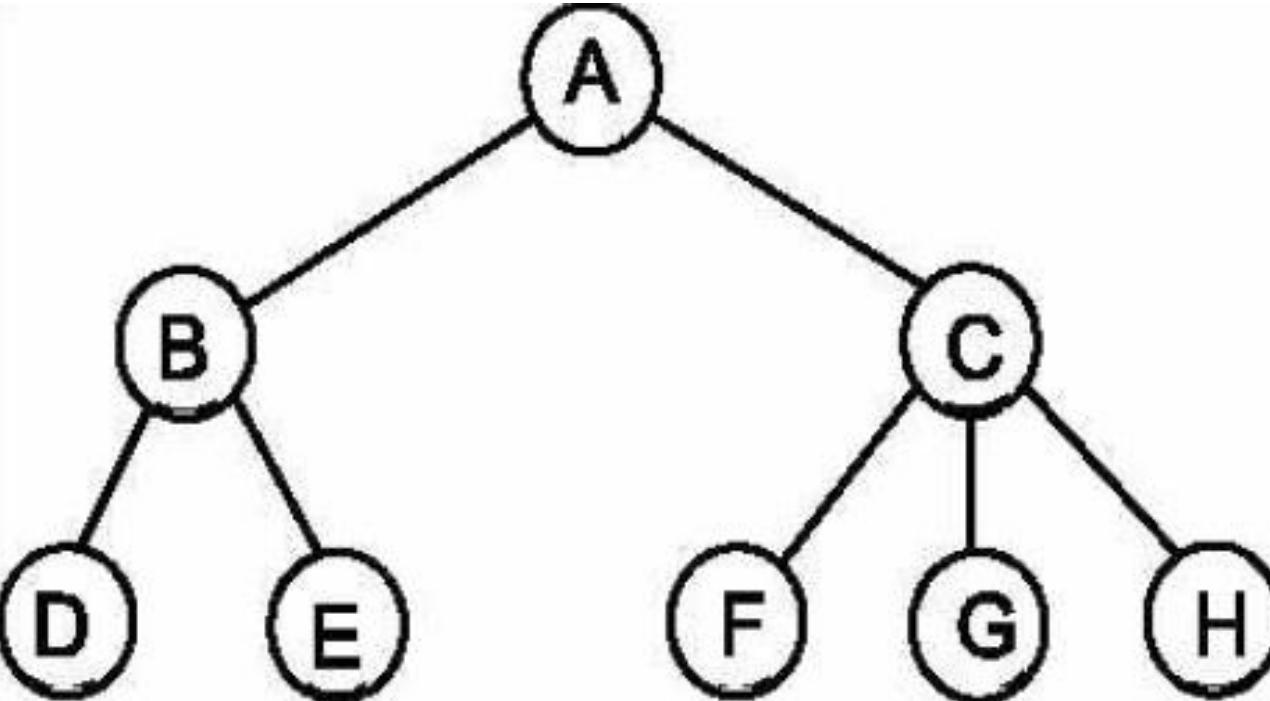
EJERCICIO DE REPRESENTACIÓN DE LISTA DE PARÉNTESIS A ÁRBOL

(25(20(10(8)), 23(21)), 90(80(62(47(32)))), 100))

REPRESENTACIÓN DE UN ÁRBOL A UNA INDENDATA

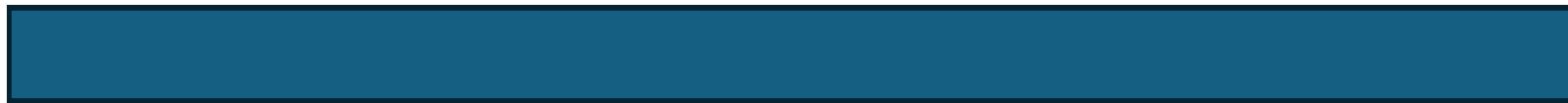
Pasos que seguir para representar un árbol a una indentada.

1. Identifica el nodo raíz y asígnale el nivel 1 para iniciar la organización jerárquica.
2. Asigna niveles a los demás nodos recorriendo el árbol; los hijos directos serán nivel 2, los siguientes nivel 3, y así sucesivamente.
3. Calcula la longitud de las barras horizontales según el nivel del nodo, haciendo que las barras sean más cortas a medida que el nivel aumenta.
4. Genera las barras desde la derecha seguido de la etiqueta identificadora del nodo, siguiendo una estructura de preorden (Primero raíz, luego hijos izquierdos, al final hijos derechos), para todos los nodos del arbol.



EJEMPLO DE REPRESENTACIÓN DE UN ÁRBOL
A UNA INDENDATA

A



B



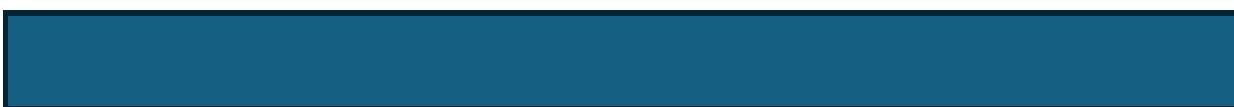
D



E



C



F



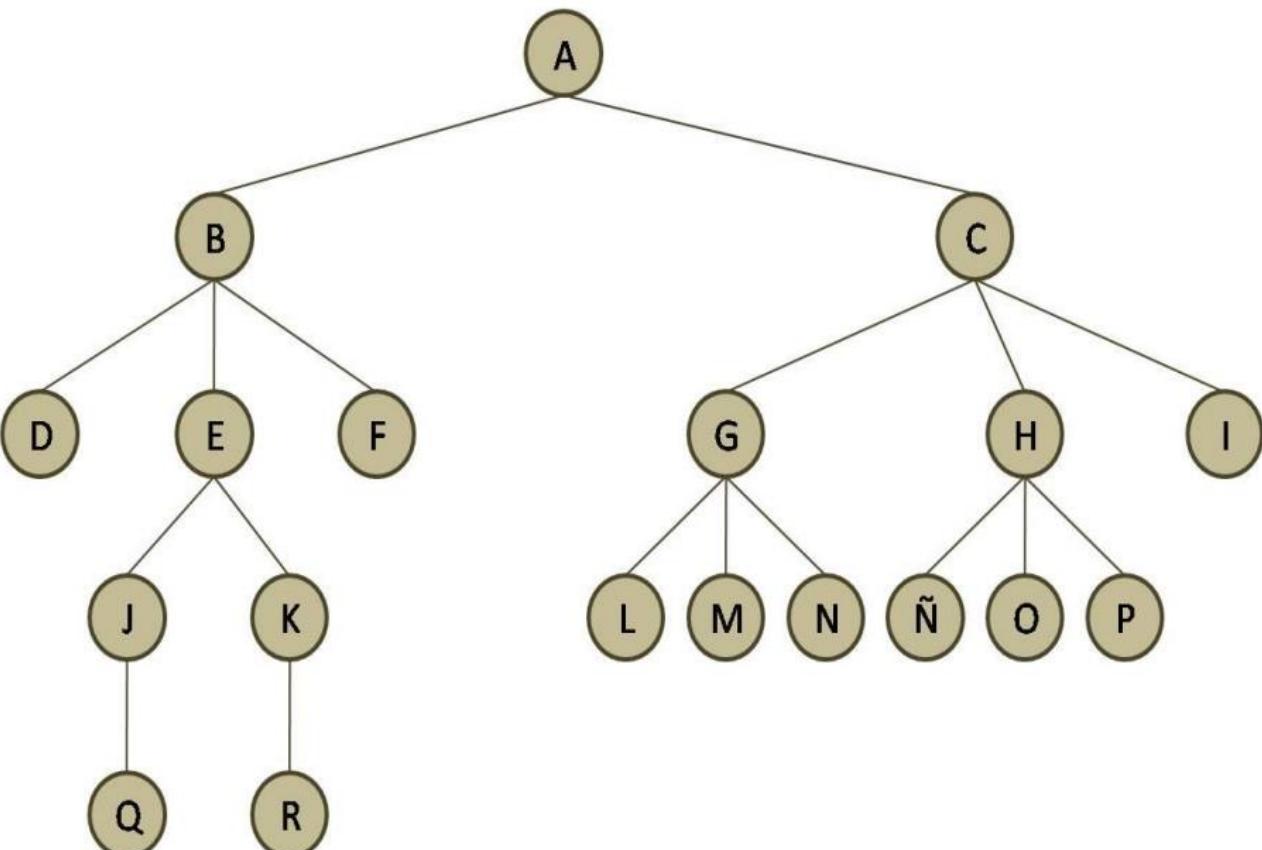
G



H



EJERCICIO DE REPRESENTACIÓN DE UN ÁRBOL A UNA INDENDATA

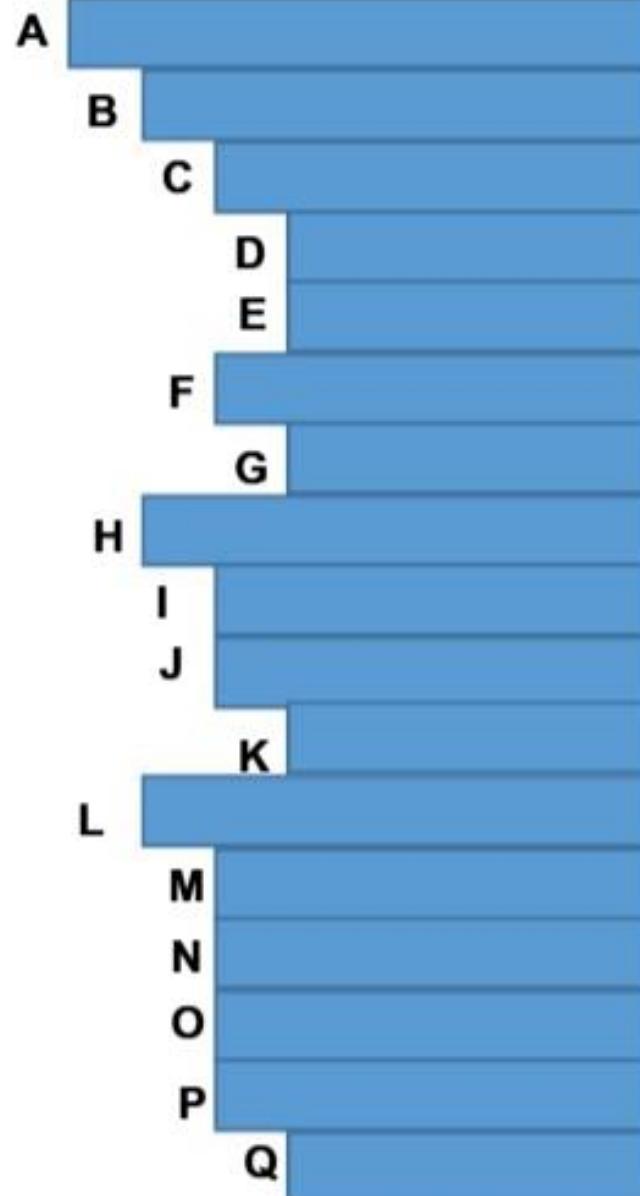


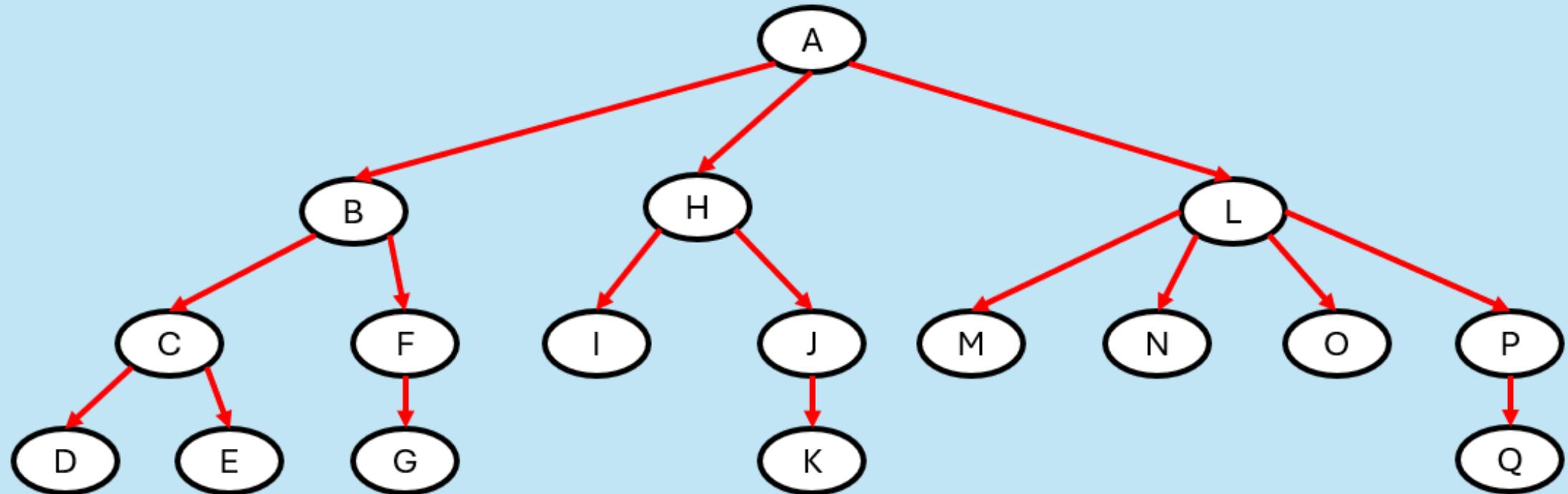
REPRESENTACIÓN DE UNA INDENTADA A UN ÁRBOL

Pasos que seguir para representar una indentación a un árbol.

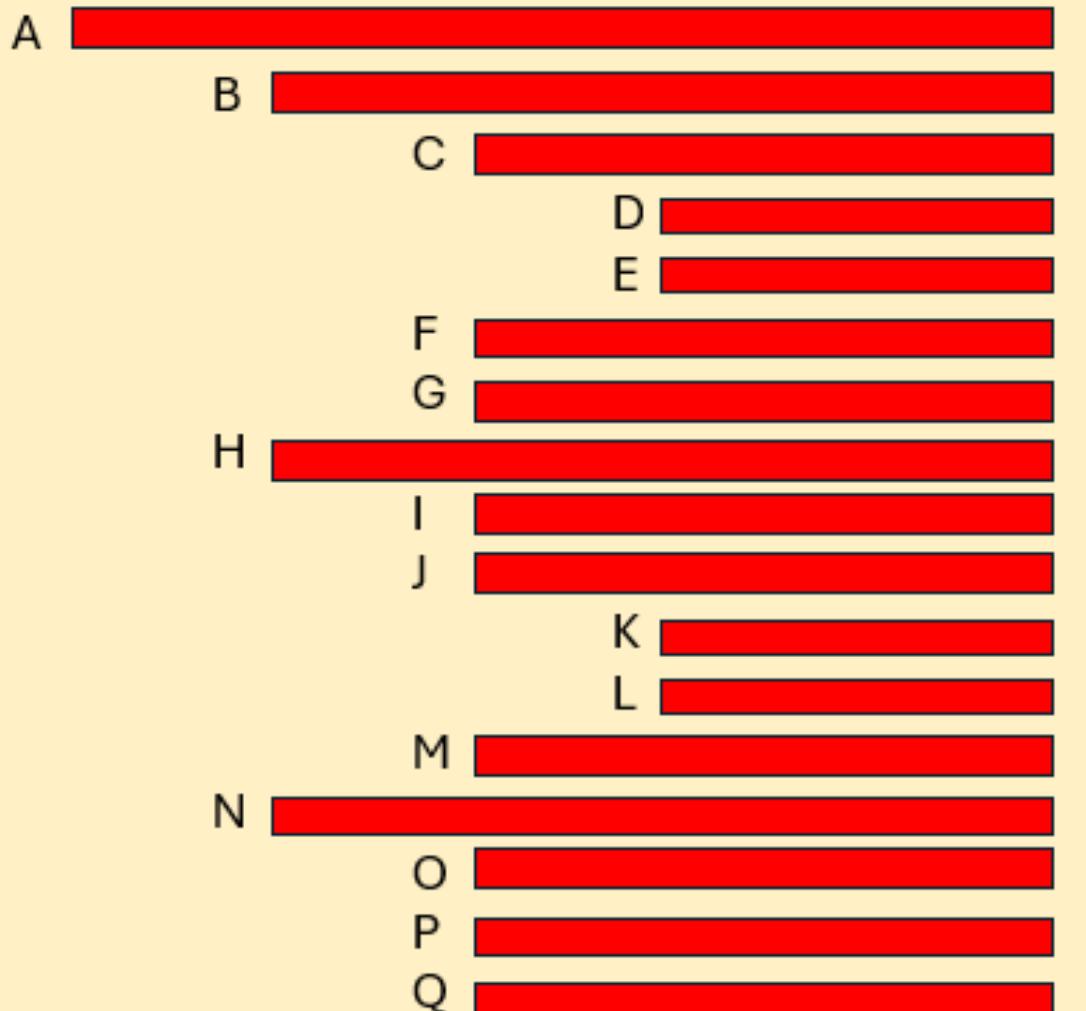
1. Identifica el nodo raíz como el primer elemento con la barra mas grande y posícnalo en el nivel 1 del árbol.
2. Determina el nivel de cada nodo según la longitud de su barra; los nodos con barras más largas estarán en niveles superiores y los de barras más cortas en niveles más profundos.
3. Conecta los nodos según su nivel. Un nodo se convierte en hijo del último nodo de nivel superior encontrado antes de él.
4. Repite el proceso para todos los nodos, asegurándose de mantener la jerarquía correcta según las longitudes de las barras.

EJEMPLO DE REPRESENTACIÓN DE UNA INDENTADA A UN ÁRBOL





EJERCICIO DE REPRESENTACIÓN DE UNA INDENTADA A UN ÁRBOL

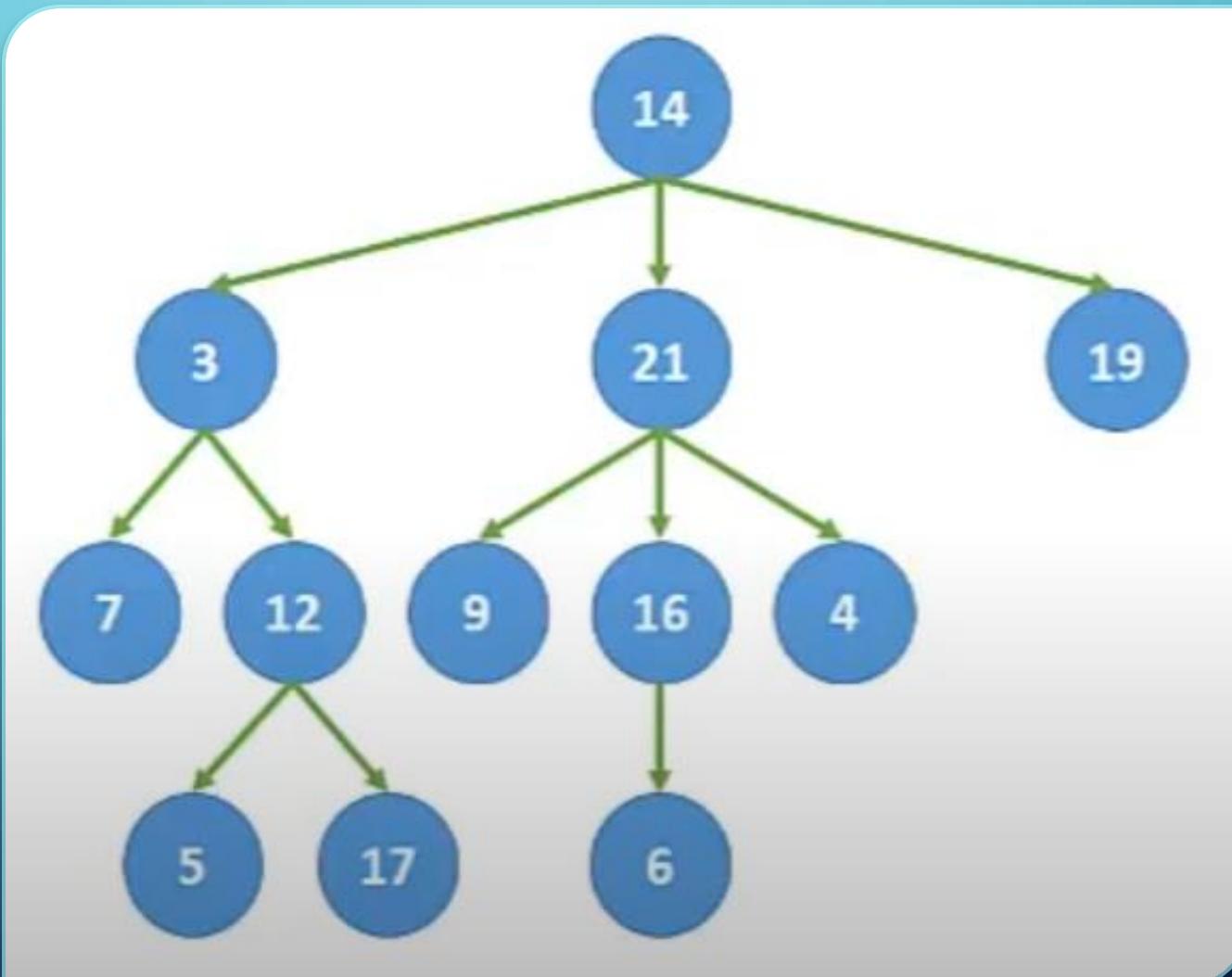


REPRESENTACIÓN DE UN ÁRBOL A UNA POR NIVELES DE PROFUNDIDAD

Pasos que seguir para representar un árbol a una por niveles de profundidad.

1. La raíz es el primer nodo del árbol y se coloca en el nivel más alto. No tiene sangría. Se numera la raíz como "1" para representar que es el primer nivel del árbol.
2. Los nodos hijos de la raíz o de cualquier nodo se colocan en una nueva línea con un incremento en la sangría. Cada nivel de profundidad se representa con un incremento en la sangría.
3. Los hijos de un nodo se numeran de acuerdo con el número de nodos previos de ese nivel. Por ejemplo, si el nodo "1" tiene dos hijos, estos serán "1.1" y "1.2". Si el hijo "1.1" tiene más hijos, estos se numeran "1.1.1", "1.1.2", y así sucesivamente. Los hijos se colocan justo debajo de su nodo padre con la sangría correspondiente.
4. Para cada nodo hijo, repite los pasos anteriores

EJEMPLO DE REPRESENTACIÓN DE UN ÁRBOL A UNA POR NIVELES



1.'14'

1.1.'3'

1.1.1.'7'

1.1.2.'12'

1.1.2.1.'5'

1.1.2.2.'17'

1.2.'21'

1.2.1.'9'

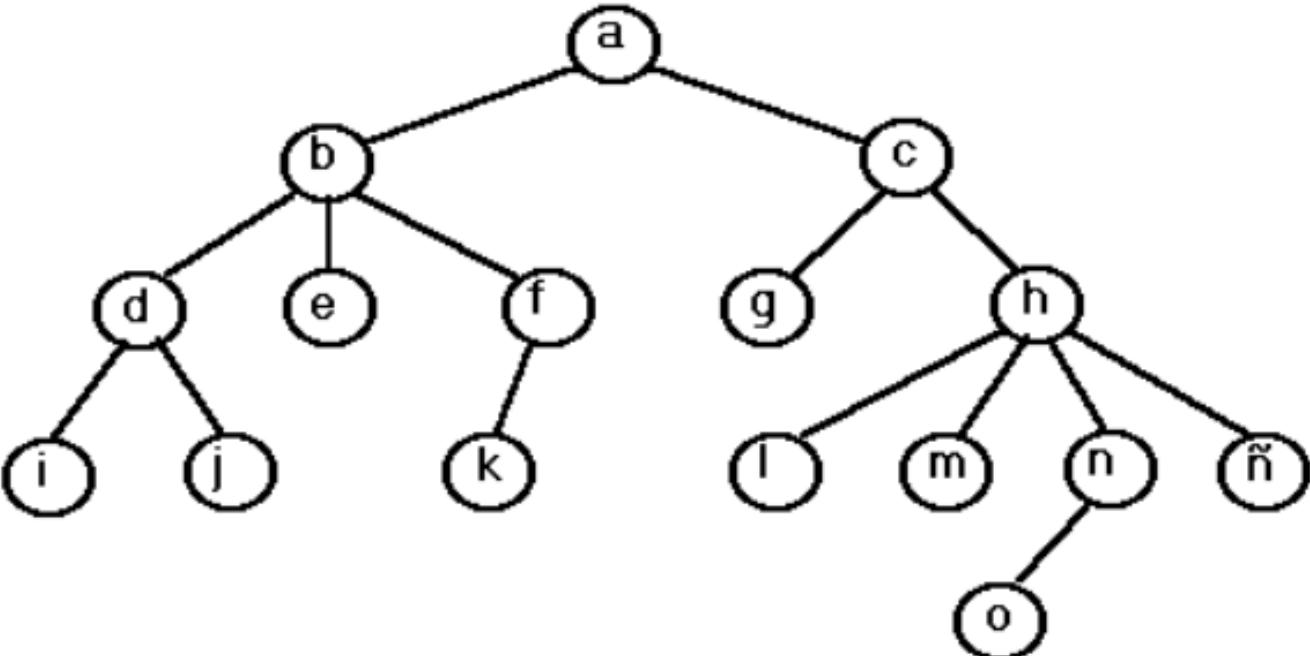
1.2.2.'16'

1.2.2.1.'6'

1.2.3.'4'

1.3.'19'

EJERCICIO DE REPRESENTACIÓN DE UN ÁRBOL A UNA POR NIVELES



REPRESENTACIÓN DE NIVELES DE PROFUNDIDAD A UN ÁRBOL

Pasos que seguir para una representación por niveles de profundidad pasarla a árbol.

1. En la representación por niveles, la raíz es el primer elemento y se coloca sin indentación. En el árbol coloca la raíz en la parte superior con su etiqueta correspondiente.
2. Los hijos de la raíz están en el siguiente nivel, con un incremento en la sangría. En el árbol gráfico, coloca estos hijos debajo de la raíz, conectándolos con líneas desde la raíz.
3. Para los hijos que tienen más hijos, coloca esos hijos en el siguiente nivel. En la representación por niveles, estos se numeran como "1.1.1", "1.1.2", etc. En el árbol gráfico, coloca estos hijos debajo de su nodo padre y conéctalos con líneas.
4. Repite el proceso para todos los nodos. Continúa conectando cada hijo con líneas hacia su nodo padre y numerando adecuadamente, manteniendo la jerarquía clara.

EJEMPLO DE UNA REPRESENTACIÓN DE NIVELES DE PROFUNDIDAD A UN ÁRBOL

1. Raíz

1.1 Hijo 1

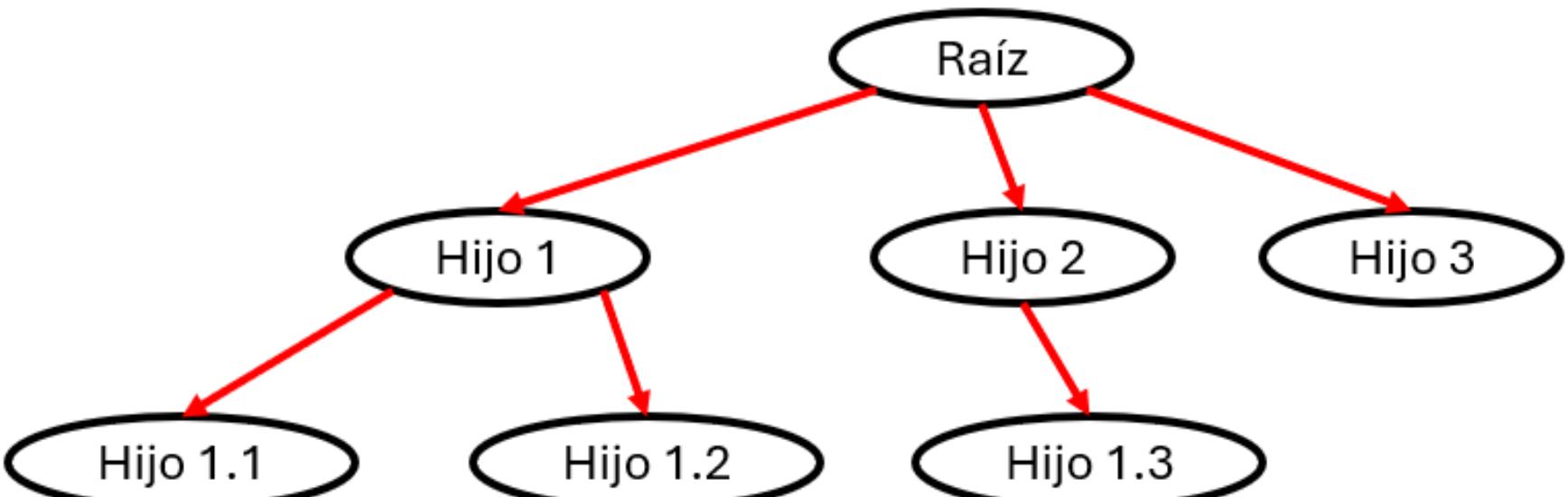
1.1.1 Hijo 1.1

1.1.2 Hijo 1.2

1.2 Hijo 2

1.2.1 Hijo 2.1

1.3 Hijo 3



EJERCICIO DE UNA REPRESENTACIÓN DE NIVELES DE PROFUNDIDAD A UN ÁRBOL

1. Raíz

1.1 Hijo 1

 1.1.1 Hijo 1.1

 1.1.2 Hijo 1.2

 1.1.2.1 Hijo 1.2.1

1.2 Hijo 2

 1.2.1 Hijo 2.1

 1.2.2 Hijo 2.2

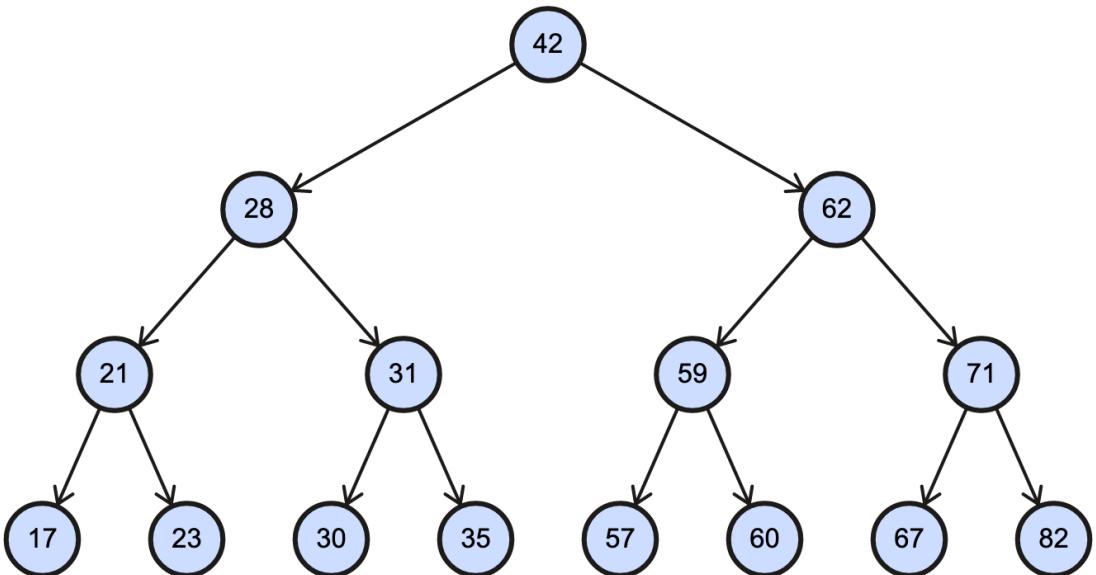
 1.2.2.1 Hijo 2.2.1

1.3 Hijo 3

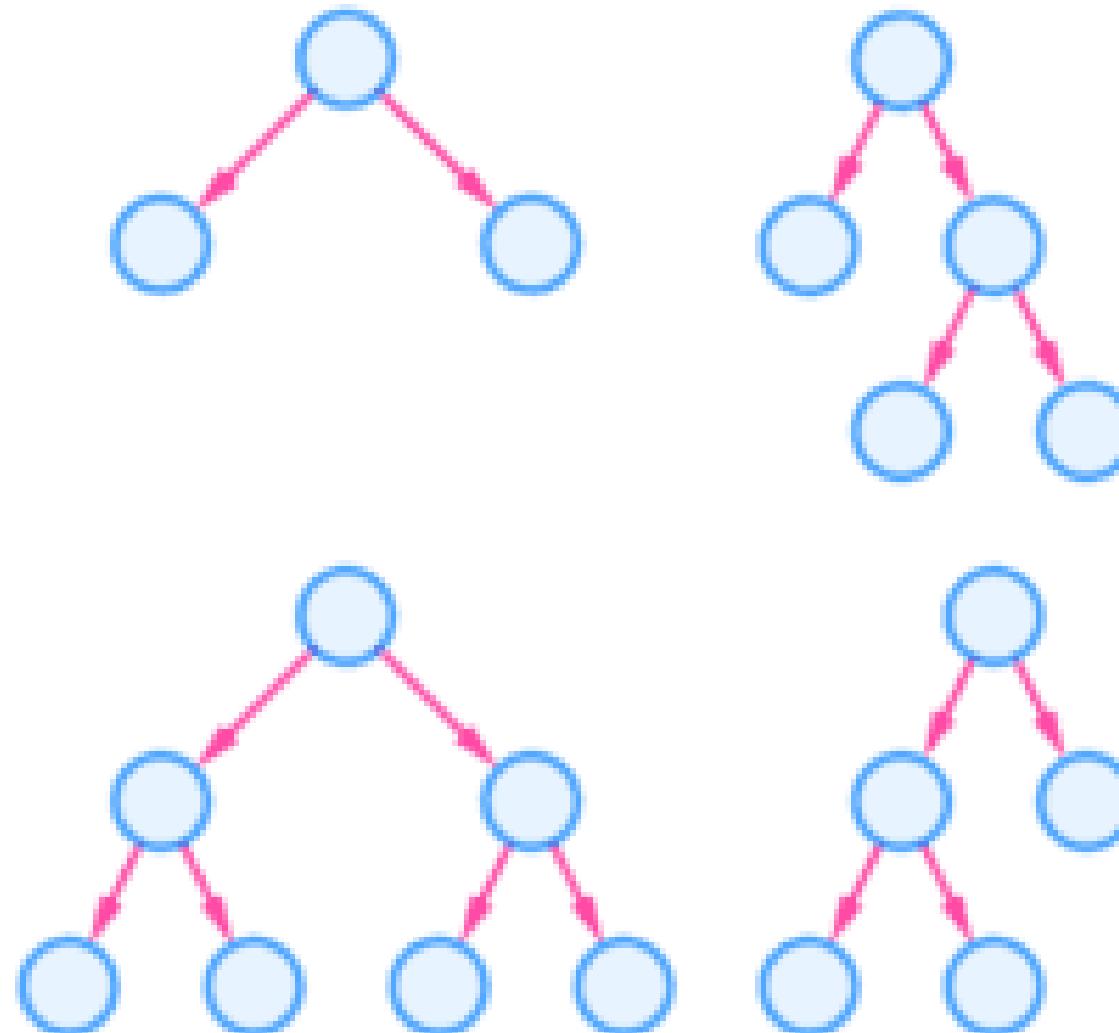
 1.3.1 Hijo 3.1

ÁRBOLES BINARIOS

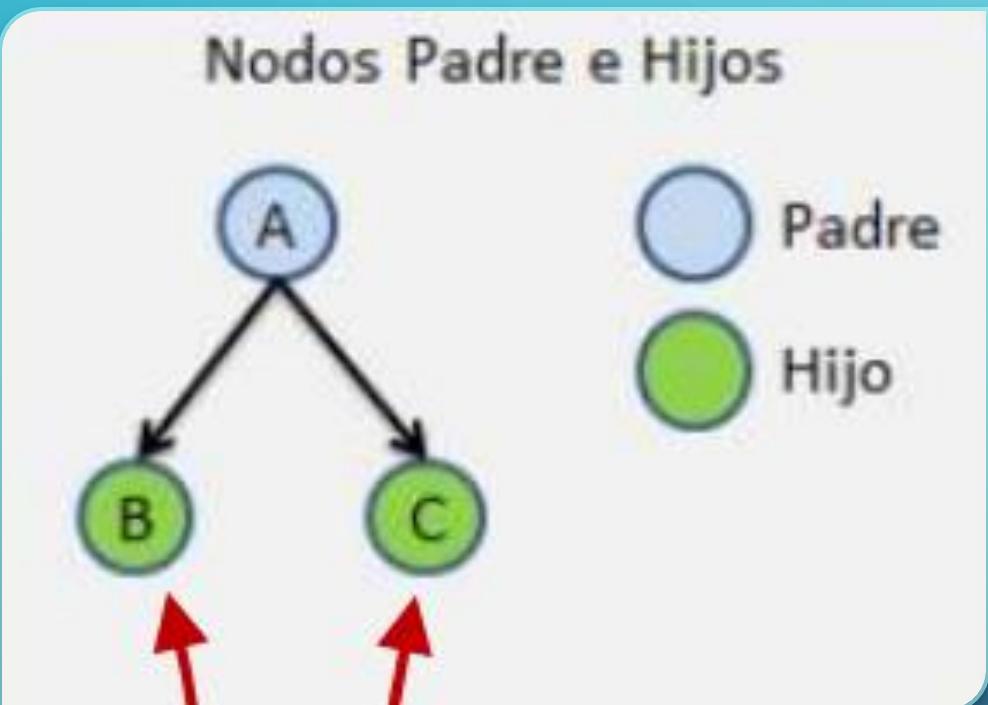
Un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre "binario").



Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno.
Usos comunes de los árboles binarios son los árboles binarios de búsqueda, los montículos binarios y Codificación de Huffman.



TERMINOLOGIA



Un **árbol binario** es un árbol en el que ningún nodo puede tener más de dos subárboles. En un árbol binario cada nodo puede tener cero, uno o dos hijos (subárboles).

Se conoce el nodo de la izquierda como **hijo izquierdo** y el nodo de la derecha como **hijo derecho**.

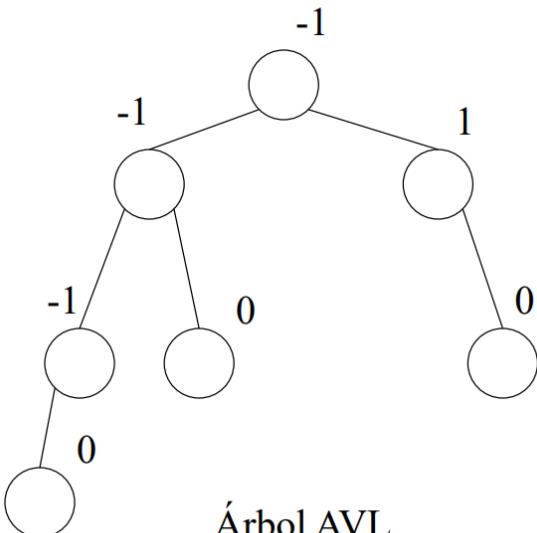
EQUILIBRIO DE ÁRBOLES

Los árboles equilibrados o balanceados surgen para mejorar el rendimiento de operaciones que involucren una búsqueda.

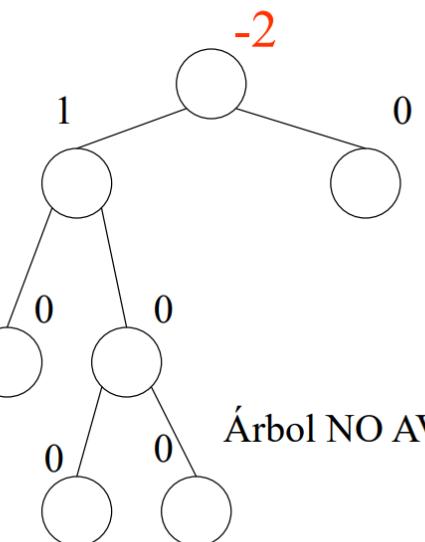
La principal característica de estos es la de realizar reacomodos o balanceos después de inserciones o eliminaciones de elementos.

Un ejemplo de árbol equilibrado es el Árbol de Adelson-Velskii y Landis: Árbol AVL

Árboles AVL



Árbol AVL

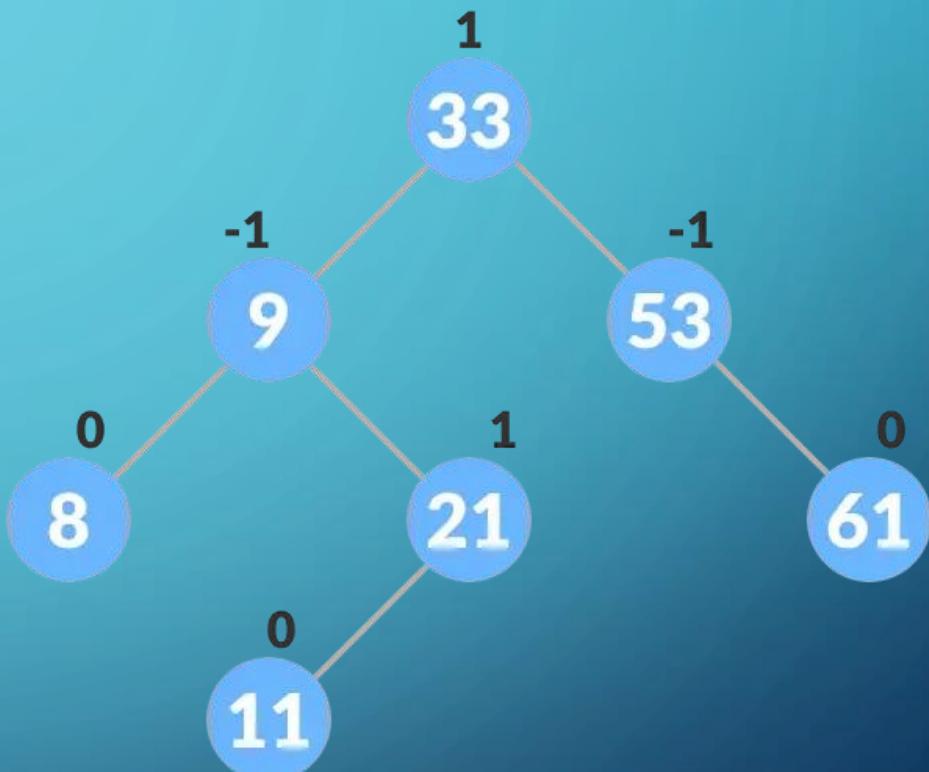


Árbol NO AVL

EQUILIBRIO DE ÁRBOLES (ÁRBOLES AVL)

El árbol AVL es un árbol binario de búsqueda con una condición de equilibrio:

- Las alturas de los 2 subárboles para cada nodo no difieren en más de una unidad.
- Factor de equilibrio o balance de un nodo se define como altura del subárbol izquierdo menos altura del subárbol derecho para ese nodo
- Cada nodo del árbol AVL puede tener un balance de $-1, 0, 1$



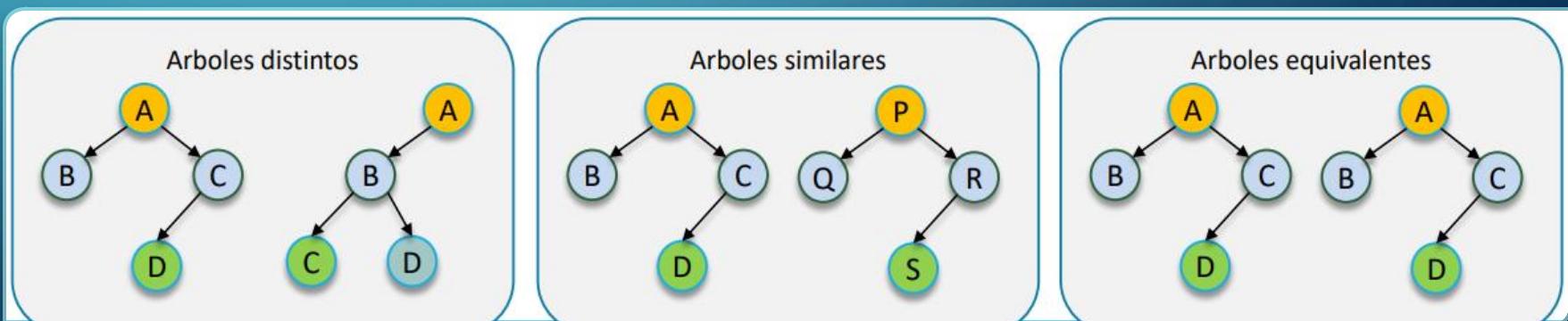
TIPOS DE ÁRBOLES BINARIOS

Los árboles pueden clasificarse tomando en cuenta su estructura y funcionamiento. A continuación, se presentan los tipos de árboles más utilizados

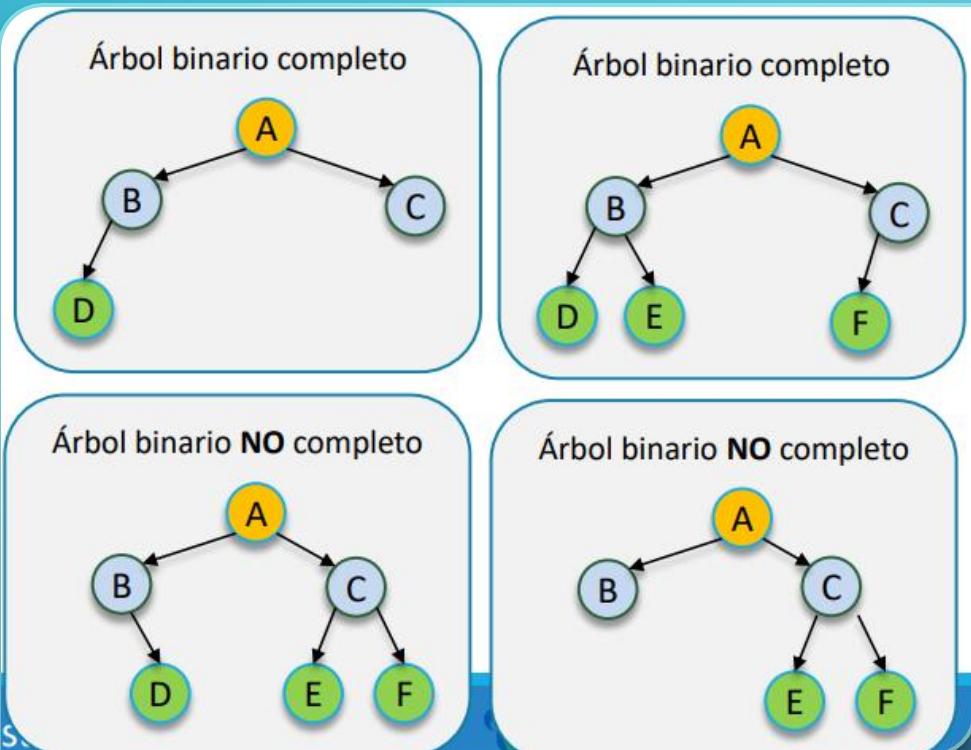
- Arboles binarios distintos
- Arboles binarios similares
- Arboles binarios equivalentes
- Arboles binarios completos
- Arboles binarios llenos
- Arboles binarios degenerados
- Arboles binarios de búsqueda
- Arboles equilibrados

ÁRBOLES BINARIOS DISTINTOS, SIMILARES Y EQUIVALENTES

- Árboles binarios distintos: Dos árboles binarios son distintos cuando sus estructuras son diferentes.
- Árboles binarios similares: Dos árboles binarios son similares cuando sus estructuras son idénticas, pero la información que contienen sus nodos difiere entre sí.
- Árboles binarios equivalentes: Los árboles binarios equivalentes se definen como aquellos que son similares y además los nodos contienen la misma información.



ÁRBOLES BINARIOS COMPLETOS

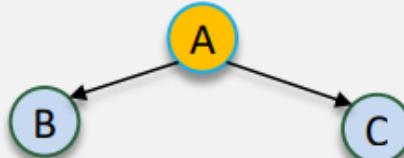


Un **árbol binario completo** de profundidad n es un árbol en el que, para cada nivel, del 0 al nivel $n-1$ tiene un conjunto lleno de nodos y todos los nodos hoja a nivel n ocupan las posiciones más a la izquierda del árbol

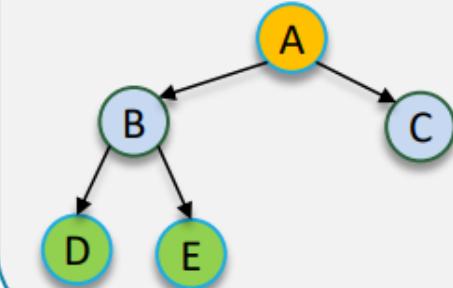
ÁRBOLES BINARIOS LLENOS

Es un árbol lleno donde todos los nodos tienen cero o dos hijos. Es decir, no existe un nodo que tenga un solo hijo.

Árbol binario lleno



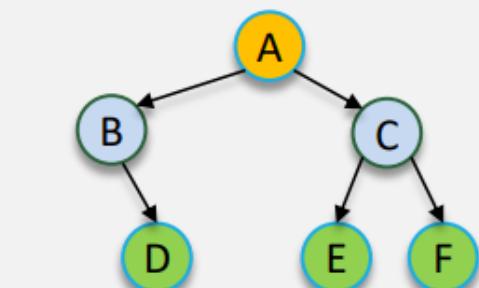
Árbol binario lleno



Árbol binario **NO** lleno



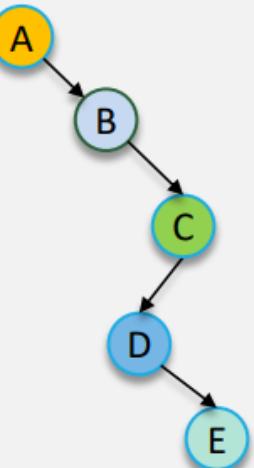
Árbol binario **NO** lleno



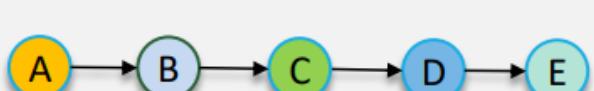
ÁRBOLES BINARIOS DEGENERADOS

Es un tipo especial denominado árbol degenerado en el que hay un solo nodo hoja y cada nodo no hoja sólo tiene un hijo. Un árbol degenerado es equivalente a una lista enlazada.

Árbol binario degenerado

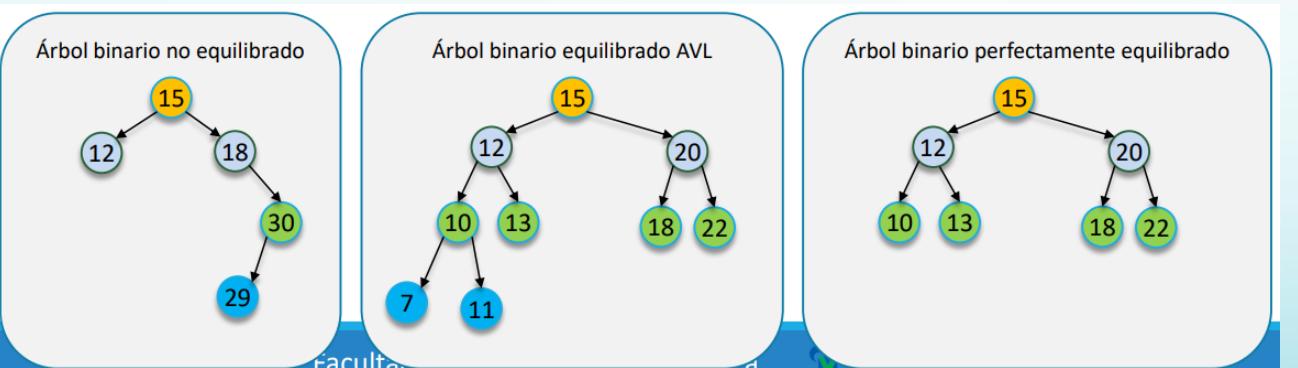


Lista enlazada

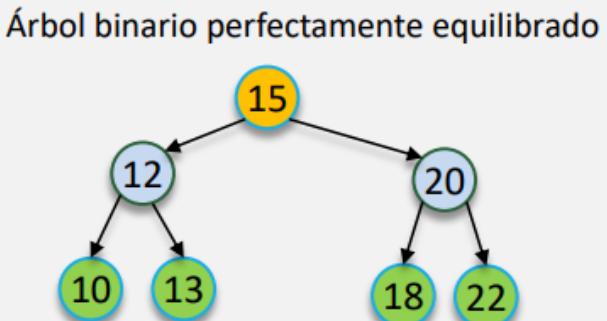


ÁRBOLES BINARIOS EQUILIBRADOS

Cuando un árbol binario de búsqueda crece descontroladamente hacia un extremo su rendimiento puede disminuir considerablemente. Para mantener la eficiencia de operación surgen los árboles equilibrados o balanceados. Estos pueden realizar acomodos o balanceos después de inserciones o eliminaciones de elementos.



ARBOLES BINARIOS PERFECTAMENTE EQUILIBRADOS



Un árbol perfectamente equilibrado es un árbol binario en el que, para todo nodo, el número de nodos en el subárbol izquierdo y el número de nodos en el subárbol derecho difieren como mucho en una unidad.

TRANSFORMACIÓN DE UN ÁRBOL GENERAL A UN ÁRBOL BINARIO

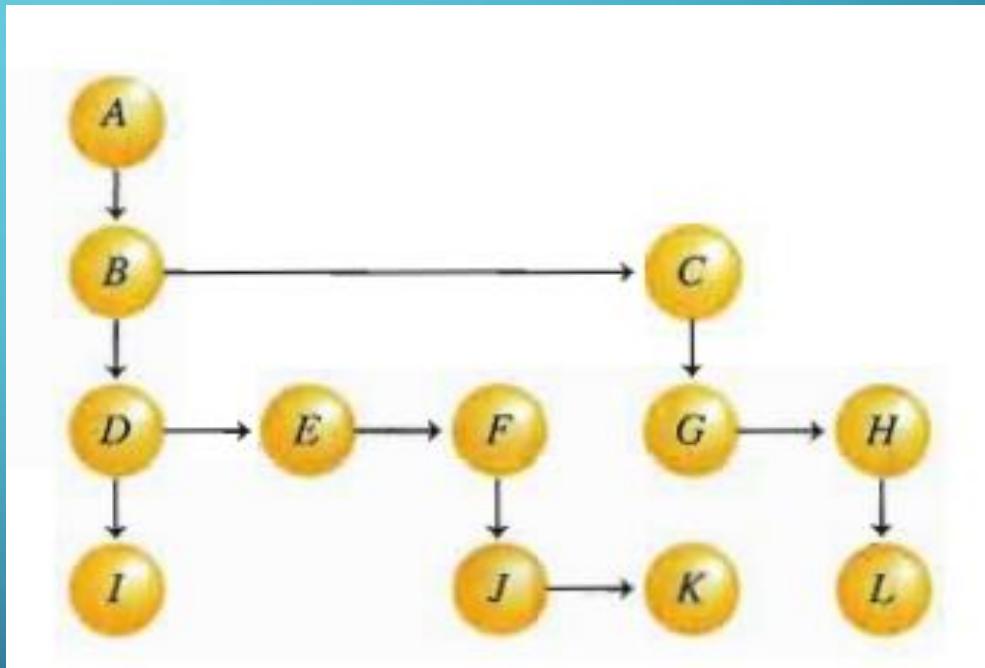
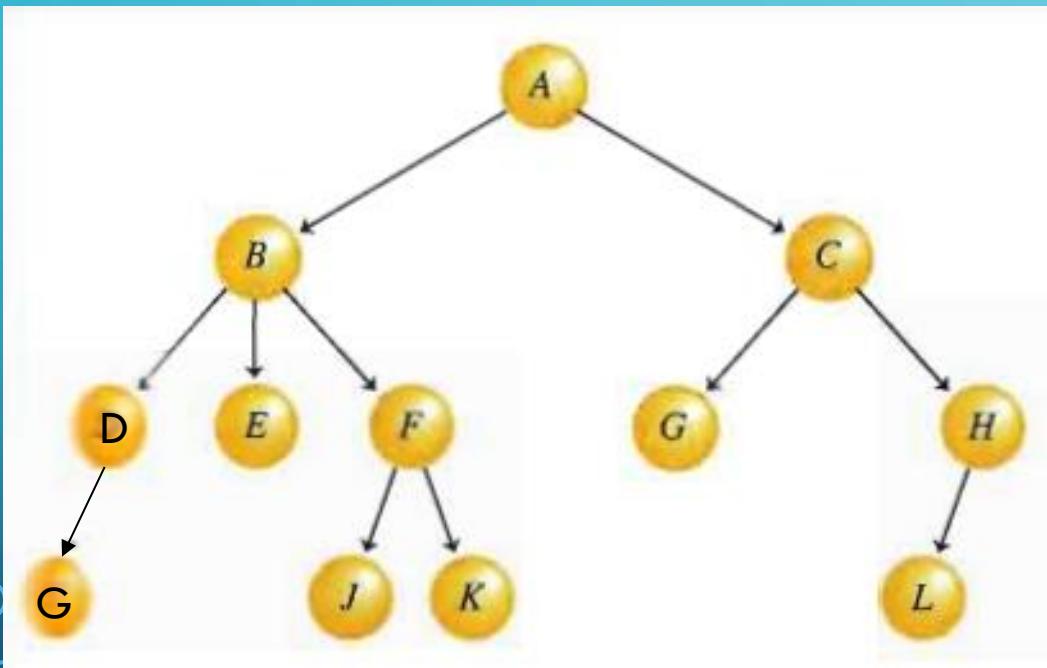
La transformación de un **árbol general** a un **árbol binario** se hace Aplicando los siguientes datos:

Pasos para la transformación

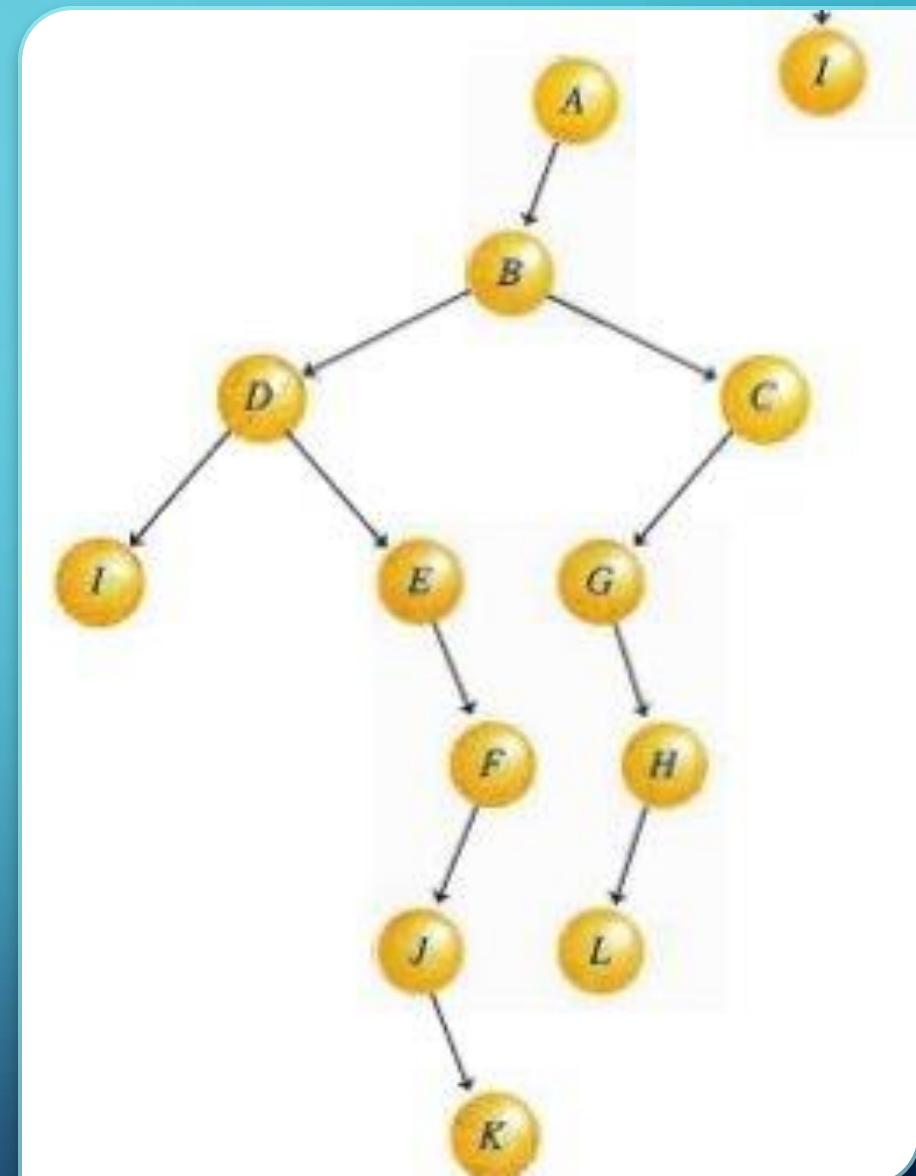
1. **Enlazar los hijos de cada nodo en forma horizontal –los hermanos–.**
2. **Relacionar en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda.**
Además, se debe eliminar el vínculo de ese padre con el resto de sus hijos.
3. **Rotar el diagrama resultante.**

Rotarlo aproximadamente 45 grados hacia la izquierda, y así se obtendrá el árbol binario correspondiente.

EJEMPLO

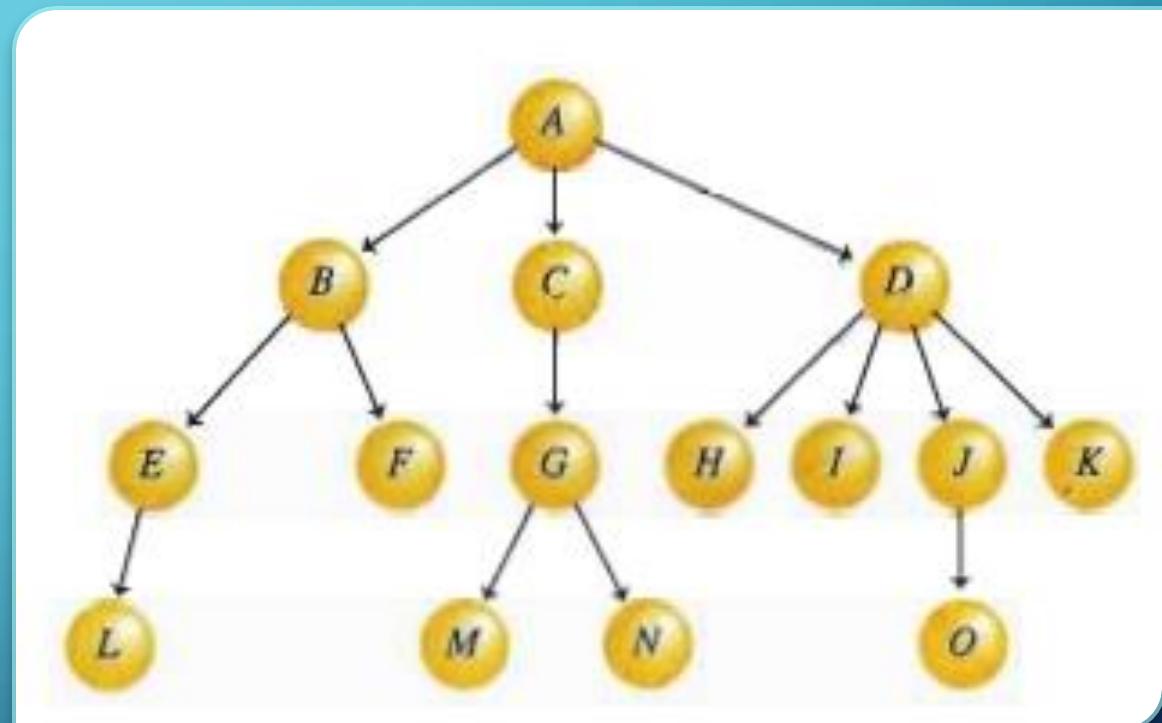


EJEMPLO

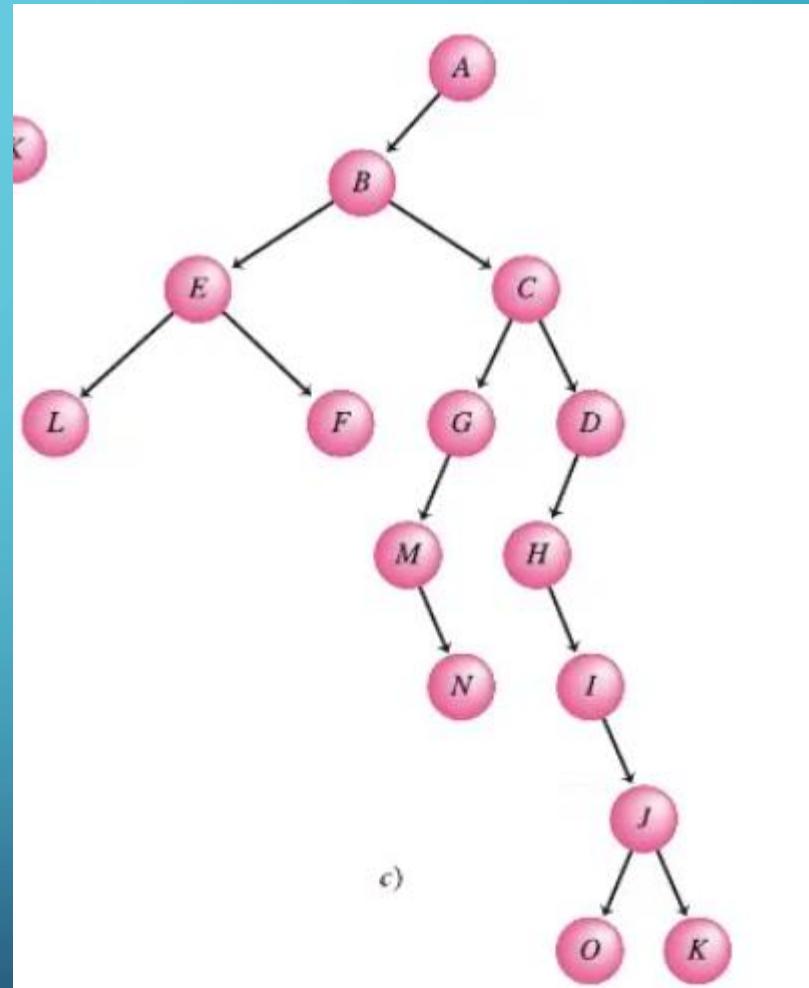
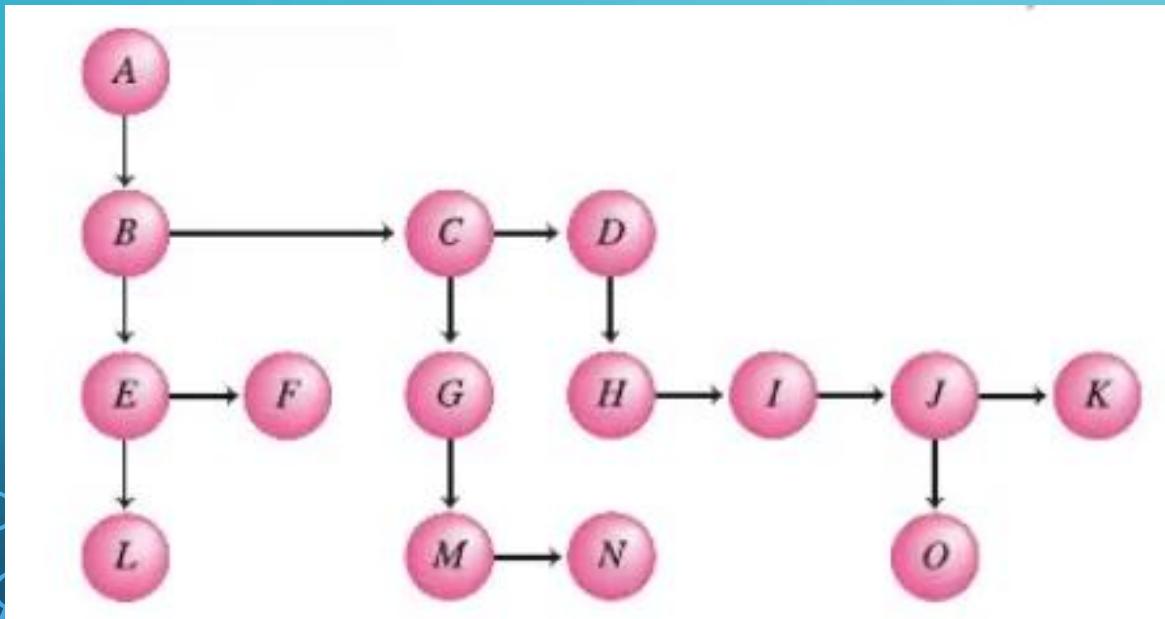


EJERCICIO

- Convertir el siguiente árbol general a árbol binario siguiendo los pasos para transformación.
- **Pasos para la transformación**
 - 1. Enlazar los hijos de cada nodo en forma horizontal –los hermanos-.**
 - 2. Relacionar en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda.**
 - 3. Rotar el diagrama resultante.**



EJERCICIO (RESULTADO)

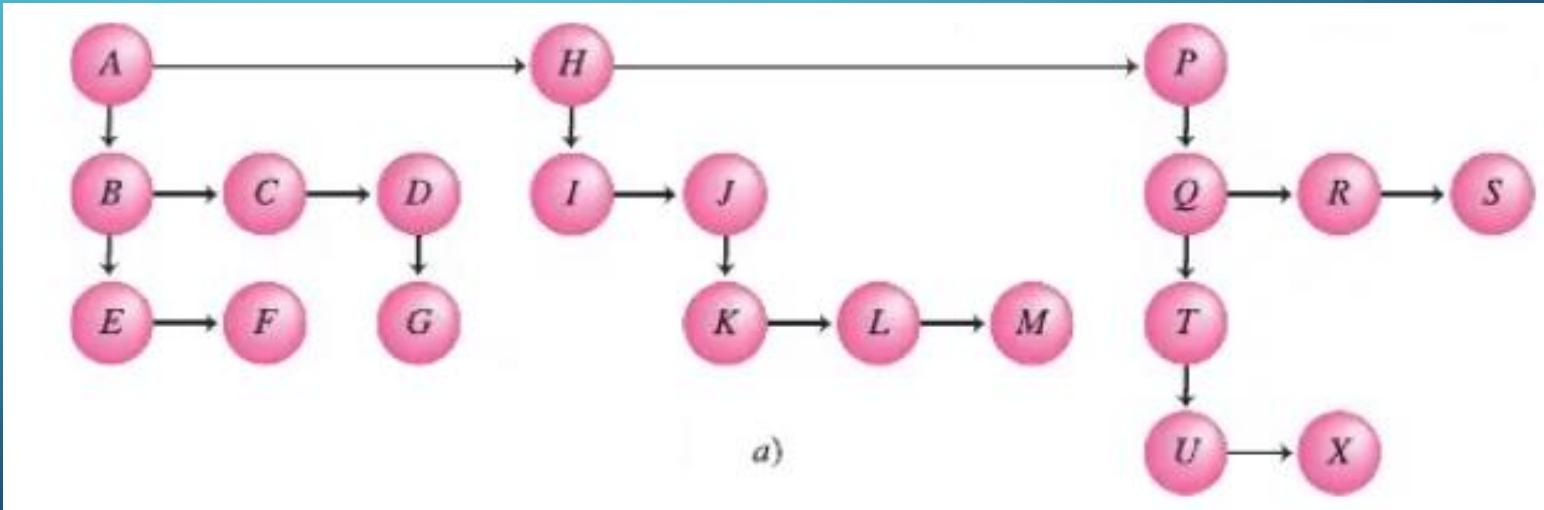
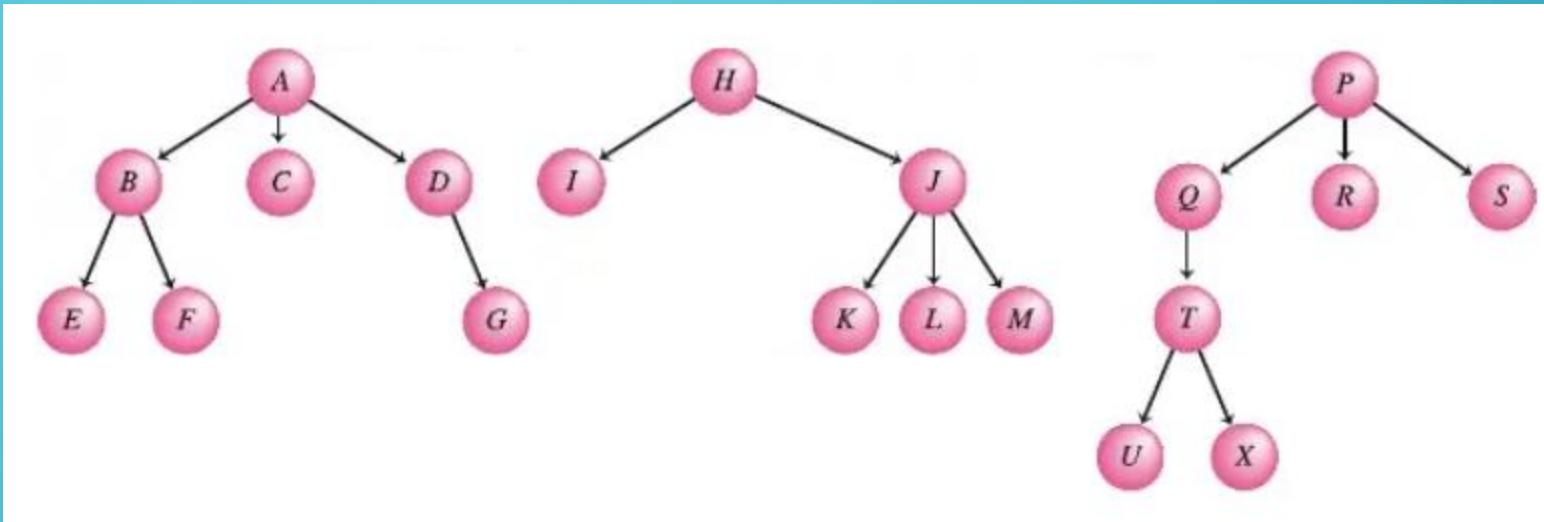


TRANSFORMACIÓN DE UN BOSQUE A UN ÁRBOL BINARIO

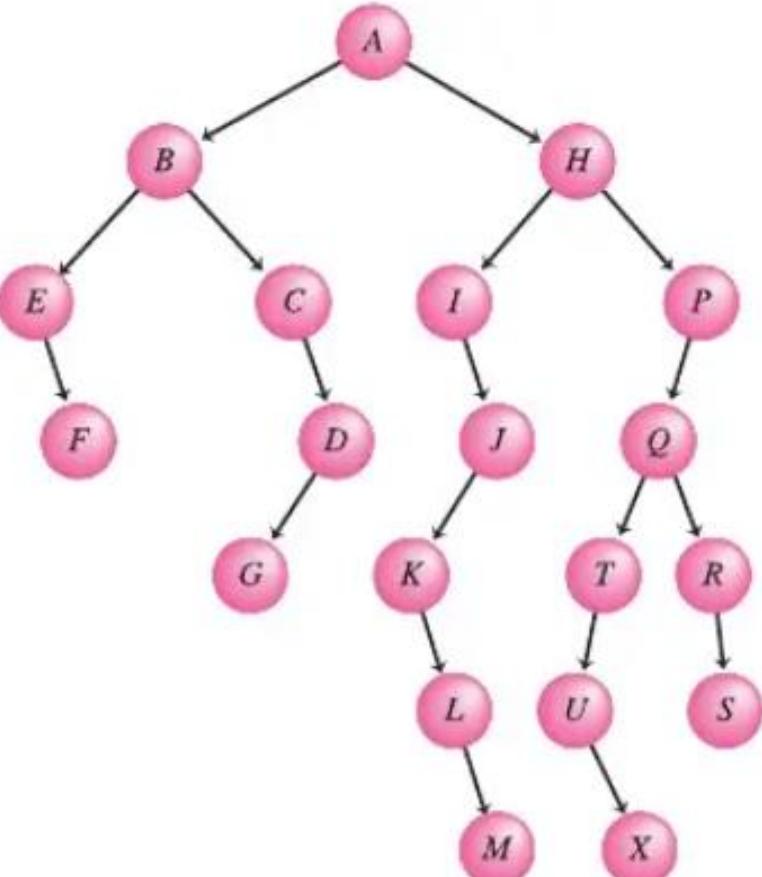
Un bosque representa un conjunto normalmente ordenado de uno o más árboles generales. Es posible utilizar el algoritmo de conversión analizado anteriormente, con algunas modificaciones, para generar un árbol binario a partir de un bosque.

1. **Enlazar en forma horizontal las raíces de los distintos árboles generales.**
2. **Enlazar los hijos de cada nodo en forma horizontal –los hermanos–.**
3. **Relacionar en forma vertical el nodo padre con el hijo que se encuentra más a la izquierda.**
 - Además, se debe eliminar el vínculo de ese padre con el resto de sus hijos.
4. **Rotar el diagrama resultante.**
 - Rotarlo aproximadamente 45 grados hacia la izquierda, y así se obtendrá el árbol binario correspondiente.

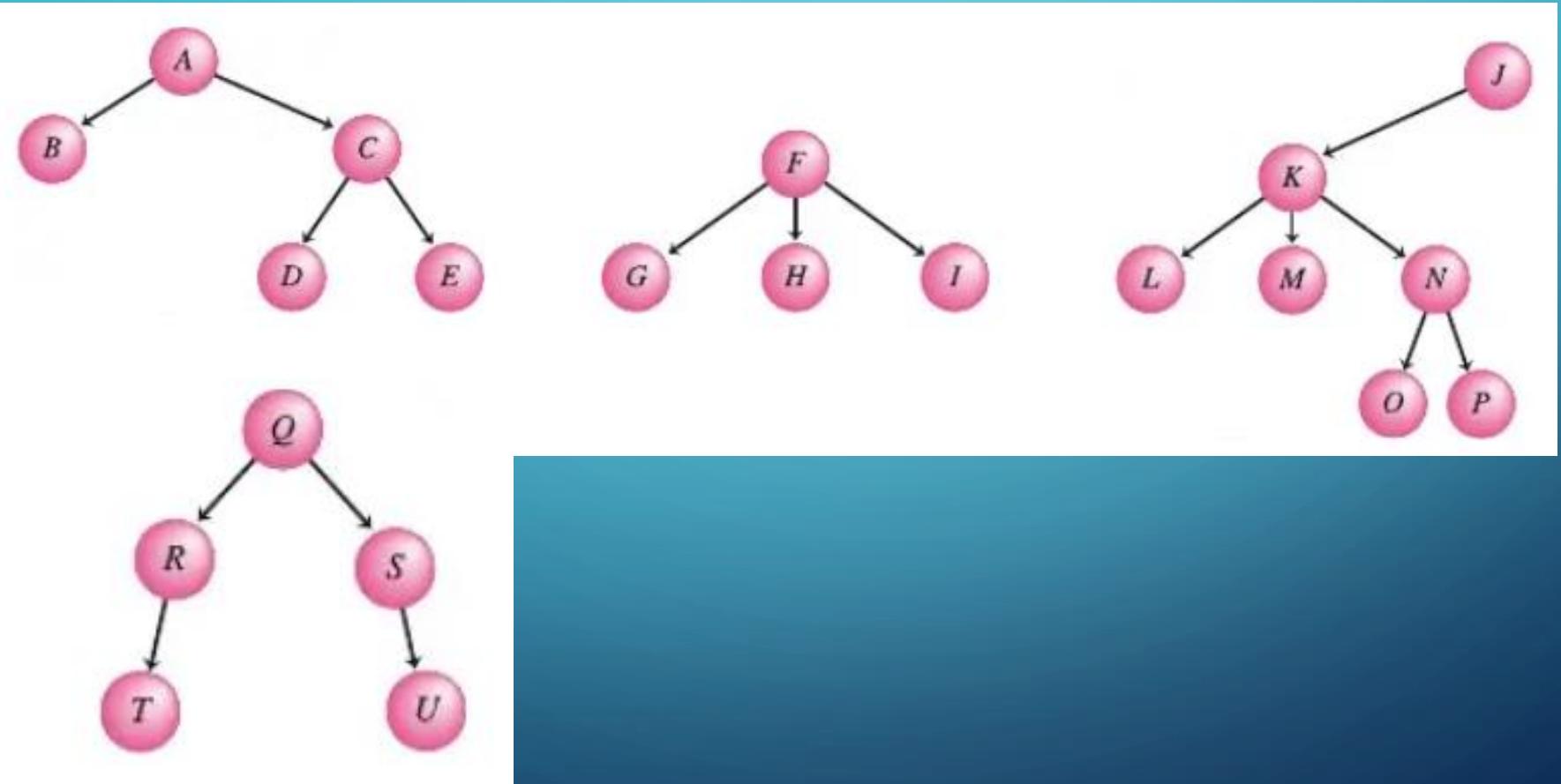
EJEMPLO



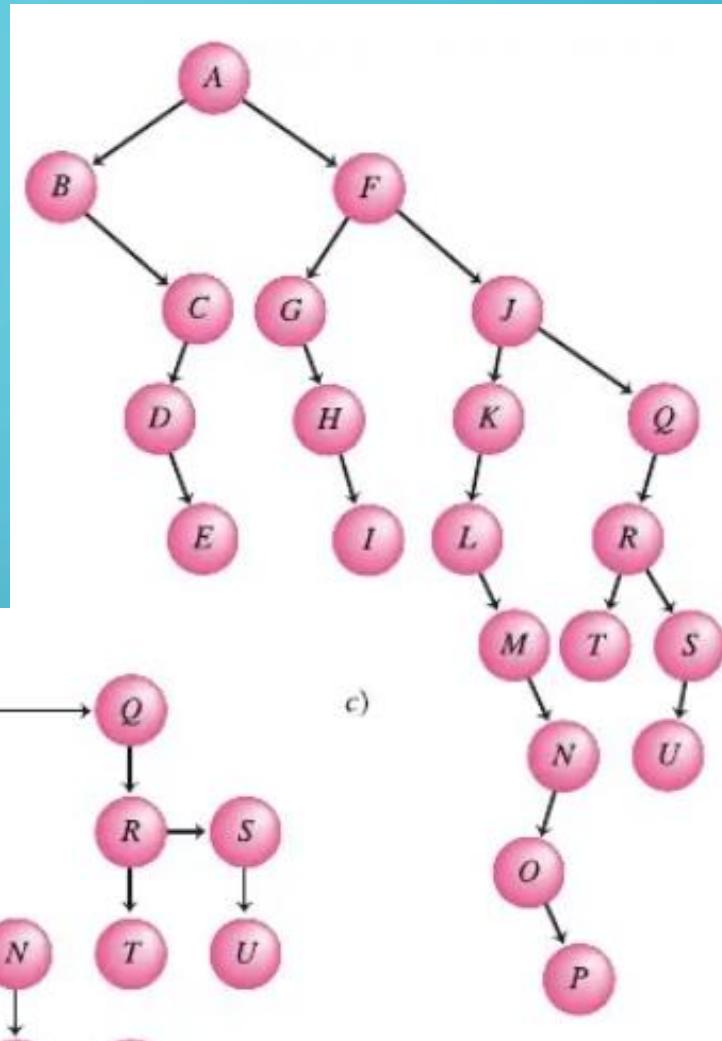
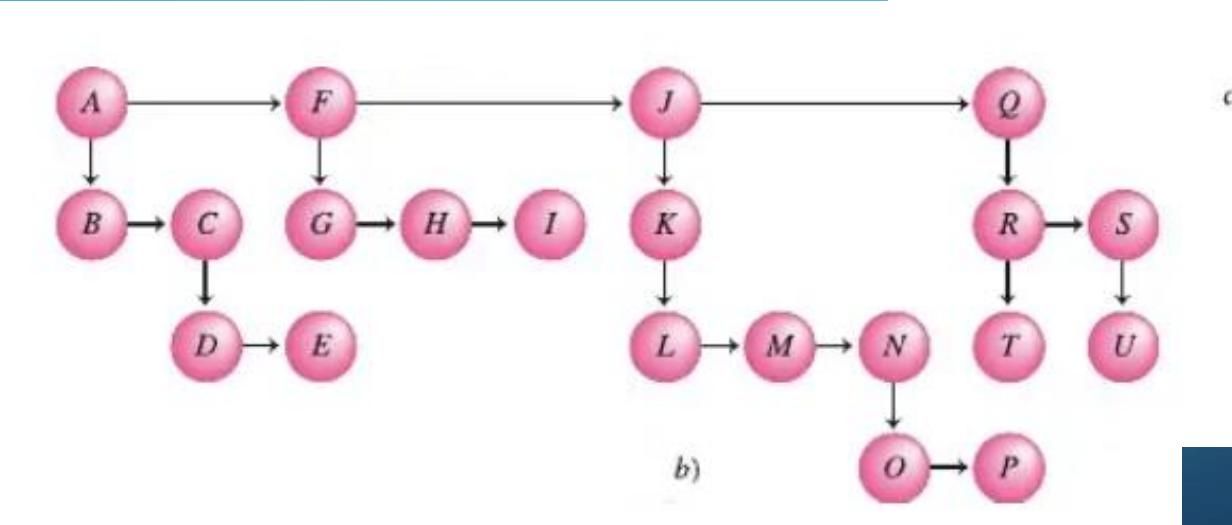
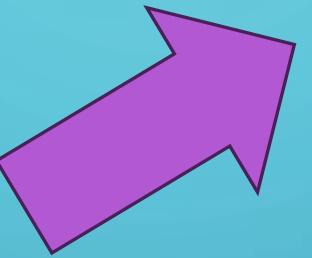
EJEMPLO



EJERCICIO (BOSQUE A ÁRBOL BINARIO)



EJERCICIO (RESULTADOS)



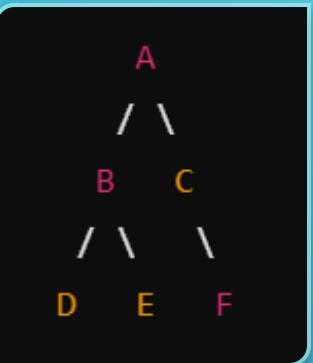
REPRESENTACIÓN DE ÁRBOLES BINARIOS

1. Representación mediante estructuras enlazadas (punteros o nodos)

- Esta es la representación más común en la implementación de árboles binarios en programación. Cada nodo contiene:
 - Un **valor** o dato.
 - Un puntero al **hijo izquierdo**.
 - Un puntero al **hijo derecho**.

```
struct Nodo {  
    int dato;  
    Nodo* izquierdo;  
    Nodo* derecho;  
};
```

REPRESENTACIÓN DE ÁRBOLES BINARIOS



[A, B, C, D, E, None, F]

2. Representación en un arreglo (Vector o Lista)

- Otra manera de representar un árbol binario es usando un **arreglo**, donde:
 - El nodo en la posición i tiene:
 - Hijo izquierdo en la posición $2i + 1$
 - Hijo derecho en la posición $2i + 2$
 - El padre está en $(i - 1) / 2$ (si $i > 0$)

REPRESENTACIÓN DE ÁRBOLES BINARIOS

3. Representación en tablas (Base de datos)

- En bases de datos o estructuras tabulares, un árbol binario puede representarse como una tabla con columnas como:

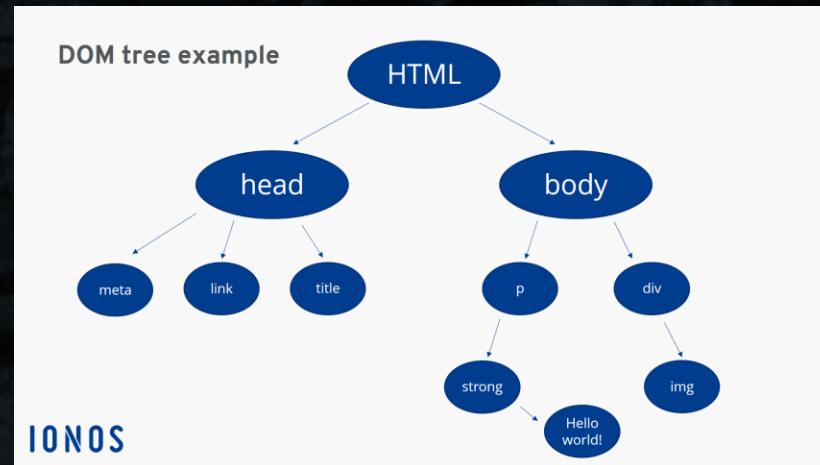
ID	Valor	ID_Padre	ID_Hijo_Izq	ID_Hijo_Der
1	A	NULL	2	3
2	B	1	4	5
3	C	1	NULL	6
4	D	2	NULL	NULL
5	E	2	NULL	NULL
6	F	3	NULL	NULL

APLICACIONES

Los árboles son una estructura de datos fundamental en programación y tienen muchas aplicaciones en diferentes áreas. Algunas de las más importantes incluyen:

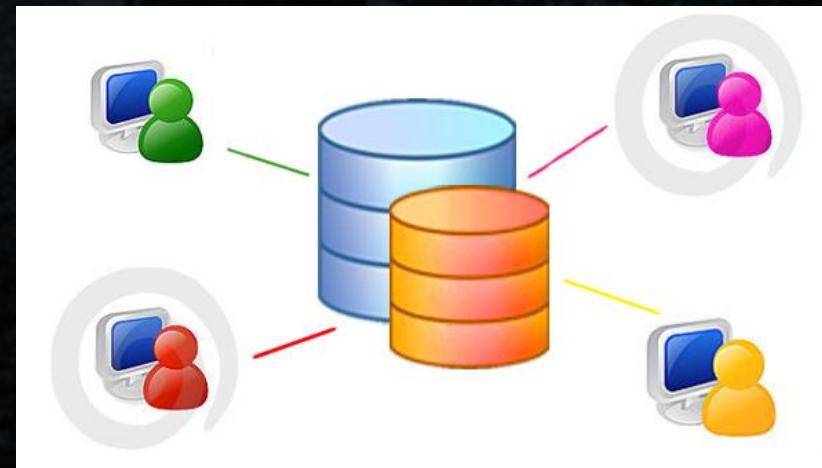
1. Manejo de Datos Jerárquicos.

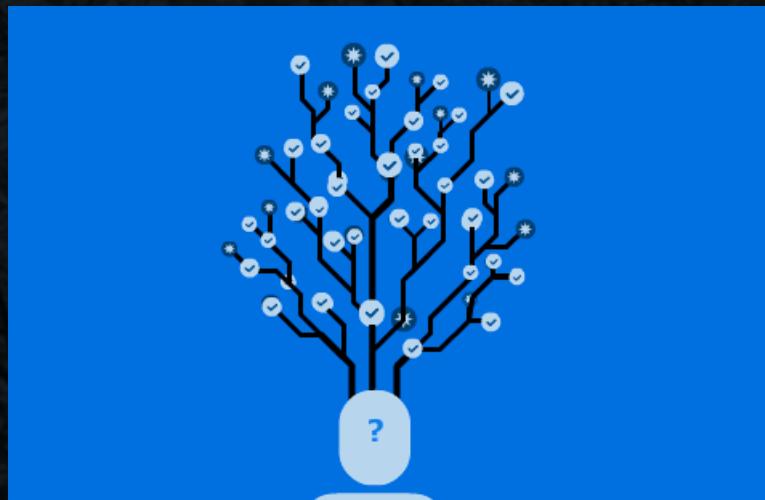
Los sistemas de archivos organizan las carpetas y archivos en una estructura de árbol. De manera similar, los documentos XML y HTML utilizan el DOM (Document Object Model) para representar sus elementos como un árbol.



2. Búsqueda y Ordenación.

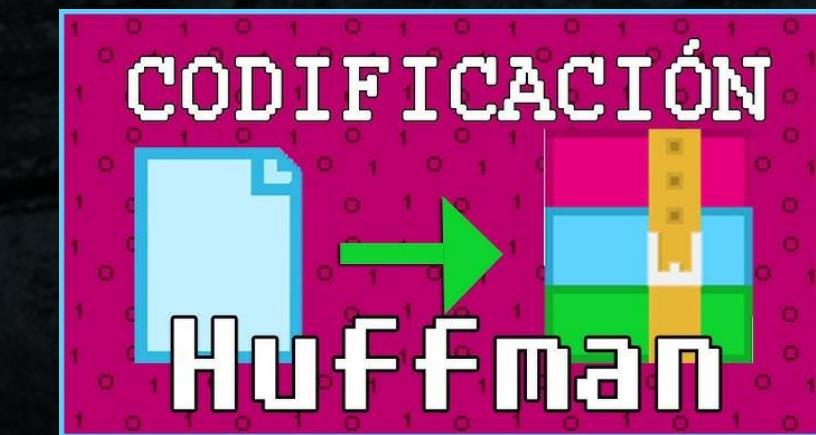
Los árboles binarios de búsqueda (BST) permiten realizar búsquedas, inserciones y eliminaciones de manera eficiente. Además, los árboles balanceados, como los AVL, se emplean en bases de datos y sistemas de archivos para mejorar el rendimiento.





3. Inteligencia Artificial y Juegos.

Los árboles de decisión se utilizan en el aprendizaje automático para clasificar datos. Por otro lado, son fundamentales en juegos como el ajedrez para determinar el mejor movimiento posible.



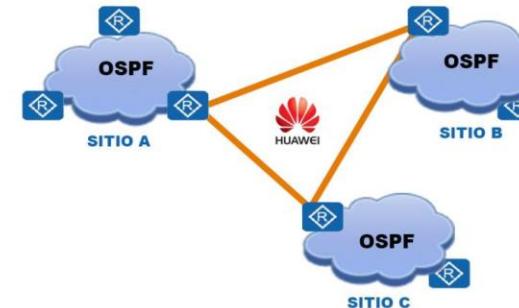
4. Compresión y Codificación.

El árbol de Huffman es un algoritmo empleado para la compresión de datos en formatos como ZIP y JPEG, permitiendo reducir el tamaño de los archivos de manera eficiente.

5. Redes y Sistemas Distribuidos.

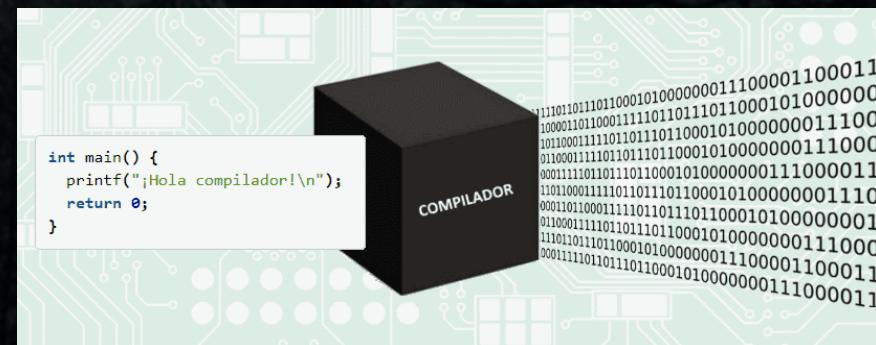
En el enrutamiento de redes, protocolos como OSPF utilizan estructuras de árbol para encontrar las rutas más eficientes. Asimismo, facilitan el almacenamiento y la recuperación eficiente de información en sistemas de bases de datos.

Open Shortest Path First (OSPF)



6. Procesamiento de Lenguaje y Compiladores.

El árbol de sintaxis abstracta (AST) representa la estructura lógica de un programa, siendo fundamental en los procesos de compilación e interpretación.



CONCLUSIÓN

Los árboles son una estructura de datos esencial en la programación, con aplicaciones que van desde la organización eficiente de la información hasta la optimización de procesos de búsqueda y almacenamiento. Su flexibilidad y variedad de implementaciones los convierten en una herramienta clave en el desarrollo de algoritmos y sistemas complejos. Comprender su funcionamiento y las diferentes variaciones permite mejorar el rendimiento y la eficiencia en múltiples áreas de la programación.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

struct Nodo
{
    int dato;
    struct Nodo *izq, *der;
};

typedef struct Nodo NODO;
typedef NODO *NODOSIG;

NODOSIG crearNodo(int dato)
{
    NODOSIG nuevo = (NODOSIG)malloc(sizeof(NODO));
    if (!nuevo)
    {
        printf("Error al asignar memoria.\n");
        return NULL;
    }
    nuevo->dato = dato;
    nuevo->izq = nuevo->der = NULL;
    return nuevo;
}
```

Se define una estructura llamada Nodo.

Esto es la base para un árbol binario, ya que cada nodo tiene dos punteros (izq y der) que apuntan a otros nodos.

Función que crea un nodo para el árbol. Se asigna memoria dinámica. Se asigna el valor dato al nodo y se establecen los punteros izq y der en NULL, ya que es un nuevo nodo sin hijos.

Caso base:

- Si `raiz == NULL`, significa que no está en el árbol.
- Si `raiz->dato == dato`, lo encontramos y devolvemos el nodo.

Si dato es menor, buscamos en la izquierda. Si dato es mayor, buscamos en la derecha.

```
NODOSIG insertar(NODOSIG raiz, int dato)
{
    if (raiz == NULL)
        return crearNodo(dato);

    if (dato <= raiz->dato) {
        raiz->izq = insertar(raiz->izq, dato);
    } else if (dato > raiz->dato)
        raiz->der = insertar(raiz->der, dato);

    return raiz;
}
```

```
NODOSIG buscar(NODOSIG raiz, int dato)
{
    if (raiz == NULL || raiz->dato == dato)
        return raiz;

    if (dato < raiz->dato)
        return buscar(raiz->izq, dato);

    return buscar(raiz->der, dato);
}
```

```
void liberarArbol(NODOSIG raiz)
{
    if (raiz != NULL) {
        liberarArbol(raiz->izq);
        liberarArbol(raiz->der);
        free(raiz);
    }
}
```

Si el árbol está vacío (`raiz == NULL`), creamos un nuevo nodo con el dato y lo devolvemos. Si dato es menor o igual al valor del nodo actual, lo insertamos a la izquierda y si es mayor, lo insertamos a la derecha (recursivamente).

Recorre primero la izquierda (`liberarArbol(raiz->izq);`), después la derecha (`liberarArbol(raiz->der);`), recursivamente y finalmente libera el nodo actual con (`free(raiz);`).

```
void inorder(NODOSIG raiz)
{
    if (raiz != NULL)
    {
        inorder(raiz->izq);
        printf("%d -> ", raiz->dato);
        inorder(raiz->der);
    }
}

void preorder(NODOSIG raiz)
{
    if(raiz != NULL)
    {
        printf("%d -> ", raiz->dato );
        preorder(raiz->izq);
        preorder(raiz->der);
    }
}

void postorden( NODOSIG raiz)
{
    if(raiz != NULL)
    {
        postorden(raiz->izq);
        postorden(raiz->der);
        printf("%d -> ", raiz->dato );
    }
}
```

Recorre la izquierda para después imprimir el dato y después recorre hacia el nodo derecho, todo recursivamente.

Imprime primero (empieza con el nodo raíz) y después hace el recorrido hacia la izquierda y va imprimiendo cada dato hasta que llegue a NULL del lado izquierdo y recorre al lado derecho y va imprimiendo.

Recorre el subárbol izquierdo, lo imprime hasta llegar a las hojas, las imprime de izquierda a derecha y va subiendo para después seguir con el subárbol derecho y terminar con el nodo raíz

```
bool numeroValido(char *cadena, int *numero)
{
    char *finalPtr;

    *numero = strtol(cadena, &finalPtr, 10);

    if (*finalPtr != '\0')
        return false;

    return true;
}
```

Verifica si una cadena representa un número entero válido y convierte la cadena a un número entero almacenándolo en una variable, utilizando `strtol`.
Donde:

- `&finalPtr`: Puntero que `strtol` usará para almacenar la posición del primer carácter no convertido.
- 10: base decimal
- `(*finalPtr != '\0')`: Verifica si hay caracteres no numéricos después del número

```
void imprimirArbol(NODOSIG raiz, int espacio)
{
    if (raiz == NULL)
        return;

    espacio += 5;

    imprimirArbol(raiz->der, espacio);

    printf("\n");
    for (int i = 5; i < espacio; i++)
        printf(" ");
    printf("%d\n", raiz->dato);

    imprimirArbol(raiz->izq, espacio);
}
```

Incrementa espacio para empujar los niveles más profundos hacia la derecha.

Imprime primero el subárbol derecho, haciendo que los nodos derechos aparezcan arriba.

Imprime el nodo actual con un número de espacios proporcional a su profundidad.

Imprime el subárbol izquierdo, colocando los nodos izquierdos más abajo.

FUNCION PRINCIPAL

```
int main()
{
    NODOSIG raiz = NULL;
    int opcion, dato;
    char cadena[32];

    do {
        do {
            printf("\n1. Insertar\n2. Buscar\n3. Mostrar \n4. Salir\nSeleccione: ");
            scanf("%s", cadena);
            if( !numeroValido(cadena, &opcion) || opcion < 1 || opcion > 4 )
                printf("Opcion invalida\n");
        } while( !numeroValido(cadena, &opcion) || opcion < 1 || opcion > 4 );

        switch (opcion)
        {
            case 1:
                do {
                    printf("Ingrese un numero: ");
                    scanf("%s", cadena);
                    if( !numeroValido(cadena, &dato) || dato < 1 )
                        printf("Numero invalido\n");
                } while( !numeroValido(cadena, &dato) || dato < 1 );
                raiz = insertar(raiz, dato);
                break;

            case 2:
                do {
                    printf("Ingrese numero a buscar: ");
                    scanf("%s", cadena);
                    if( !numeroValido(cadena, &dato) || dato < 1 )
                        printf("Numero invalido\n");
                } while( !numeroValido(cadena, &dato) || dato < 1 );
                if (buscar(raiz, dato))
                    printf("El numero %d esta en el arbol.\n", dato);
                else
                    printf("El numero %d no esta en el arbol.\n", dato);
                break;
        }
    }
}
```

Validación de lectura de variable para el menú.

Caso 1, se agrega un numero validado al árbol utilizando la función insertar.

Caso 2, se hace una búsqueda dado un número y se verifica si es que está en el árbol.

```
case 3:  
    do {  
        printf("\nMostrar arbol en:\n1. Preorden\n2. Inorden\n3. Postorden\n: ");  
        scanf("%s", cadena);  
        if(!numeroValido(cadena, &dato) || dato < 1 || dato > 3 )  
            printf("Respuesta invalida\n");  
    } while( !numeroValido(cadena, &dato) || dato < 1 || dato > 3 );  
    if( raiz != NULL )  
    {  
        if( dato == 1 )  
        {  
            preorden(raiz);  
            printf("NULL\n\n");  
            imprimirArbol(raiz, 0);  
        }  
        else if( dato == 2 )  
        {  
            inorder(raiz);  
            printf("NULL\n\n");  
            imprimirArbol(raiz, 0);  
        }  
        else  
        {  
            postorden(raiz);  
            printf("NULL\n\n");  
            imprimirArbol(raiz, 0);  
        }  
    }  
    else  
        printf("No se ha creado el arbol\n");  
    break;  
  
case 4:  
    if( raiz != NULL )  
    {  
        liberarArbol(raiz);  
        printf("Memoria liberada. Saliendo...\n");  
    }  
    else  
        printf("No se ha creado el arbol\n");  
    break;  
}  
} while (opcion != 4);  
  
return 0;
```

Caso 3, se le pide al usuario el tipo de recorrido, se valida y después se llama a la función para imprimir los datos del árbol.

Caso 4, se llama a la función liberarArbol para liberar la memoria nodo por nodo.

BIBLIOGRAFIAS

Instituto Consorcio Clavijero. (s. f.). *6.1.1 Concepto de arbol - Matematicas Discretas*. Recuperado el 6 de marzo de 2025, de https://cursos.clavijero.edu.mx/cursos/006_md/modulo6/contenidos/tema6.1.1.html

Fagúndez, M. (S/f). *Teoría de Grafos y Árboles*. Recuperado el 6 de marzo de 2025, de <http://www.geocities.ws/mfagundez24/UJAP/Discreta/TeoriadeGrafos3.pdf>

Arriondo, R., Bottazzi, C., Costarelli, S., D'Elía, J., Dalcin, L., ... (2024, 3 Diciembre). *Algoritmos y Estructuras de Datos*. Recuperado el 6 de marzo de 2025, de <https://cimec.org.ar/~mstorti/aed/aednotes.pdf>

Charly's Codes. (2020, 7 Junio). *Árbol general - recorrido por nivel* [Vídeo]. YouTube. Recuperado el 6 de marzo de 2025, de <https://www.youtube.com/watch?v=UCumF3yXYoU>

Horowitz, E., Sahni, S., & Anderson-Freed, S. (2008). *Fundamentals of Data Structures in C++*. Recuperado el 6 de marzo de 2025, de https://www.researchgate.net/publication/220693653_Fundamentals_of_Data_Structure_in_C