Instituto Politécnico Nacional

Escuela Superior de Cómputo

Estructuras de Datos

Tema 09: TAD Árbol binario

M. en C. Edgardo Adrián Franco Martínez
http://www.eafranco.com
edfrancom@ipn.mx
@edfrancom *edgardoadrianfrancom*

Contenido

- *Introducción*
- *El árbol binario*
- *Definición recursiva del árbol binario*
- *Especificación del TAD árbol binario*
  - *Cabecera*
  - *Definición*
  - *Operaciones*
  - *Observaciones*
- *Conversión de arboles generales a binarios*
- *Representación de árboles binarios en memoria*
- *Recorrido en árboles binarios*
  - *Preorden*
  - *Inorden*
  - *postorden*
- *Recursividad en los recorridos*

Introducción

- **Los árboles binarios** representan las **estructuras de datos no lineales y dinámicas más utilizadas cuando de árboles se habla** en computación.

- Un **árbol binario** es una estructura de datos en la cual cada nodo siempre tiene un hijo izquierdo y un hijo derecho.

- No pueden tener más de dos hijos (de ahí el nombre "binario").

El árbol binario

- Un árbol binario es una estructura de datos de tipo árbol en donde cada uno de los nodos del árbol puede tener 0, 1, ó 2 subárboles llamados de acuerdo a su caso como:
  - Si el nodo raíz tiene **0 relaciones** se llama ***hoja***.
  - Si el nodo raíz tiene **1 relación a la izquierda**, el segundo elemento de la relación es el ***subárbol izquierdo***.
  - Si el nodo raíz tiene **1 relación a la derecha**, el segundo elemento de la relación es el ***subárbol derecho***.

- Un **árbol *binario A*** es un conjunto de elementos del mismo tipo tal que:

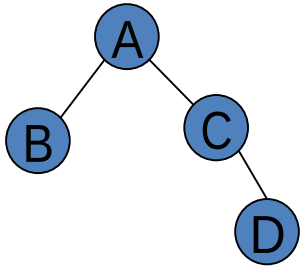
  - O bien es el conjunto vacío, en cuyo caso se le denomina árbol *vacío*.

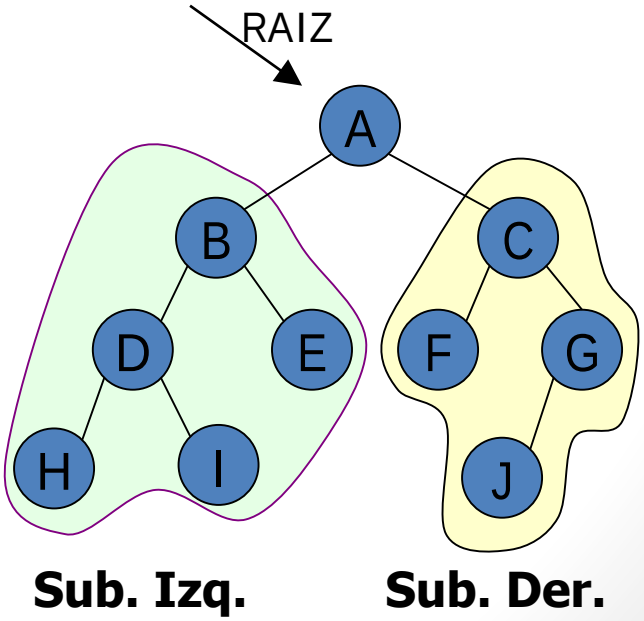
  - O bien no es vacío, en cuyo caso existe un elemento distinguido llamado raíz, y el resto de los elementos se distribuyen en dos subconjuntos disjuntos A1, A2, cada uno de los cuales es un árbol binario, llamados respectivamente subárboles izquierdo y derecho del árbol original

  -

El árbol binario

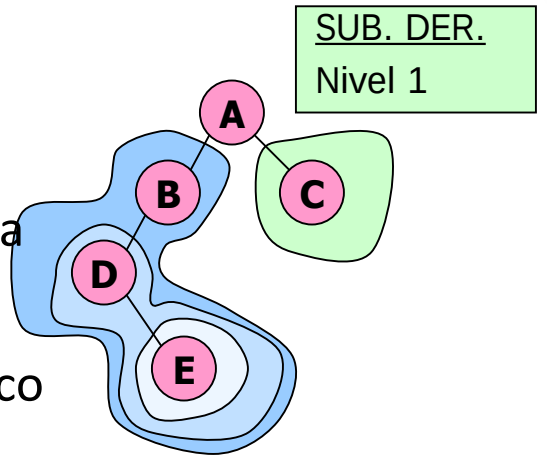
- El árbol binario es un tipo especial de árbol en donde:
  - Cada nodo no puede tener mas de dos hijos
- Un árbol binario puede ser un conjunto
  - **Vacío**, no tiene ningún nodo
  - O constar de tres partes:
    - Un nodo raíz y
    - Dos subárboles binarios: izquierdo y derecho
- La definición de un árbol binario es recursiva
  - La definición global depende de si misma



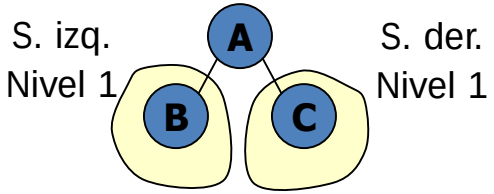


Definición recursiva del árbol binario

- La definición del árbol es recursiva
  - Se basa en si misma
- La terminología de los árboles
  - También puede ser definida en forma recursiva
- **Ejemplo: NIVEL DE UN ÁRBOL**
  - Identificar el caso recursivo y el caso mas básico



nivel 1 → A



Nivel Del Arbol: 2

SUB. IZQ.
Nivel = 1 +
Max(**3**,**Sub.Der.**)

NIVEL : 4

Caso Básico

Un árbol con un solo nodo tiene nivel 1

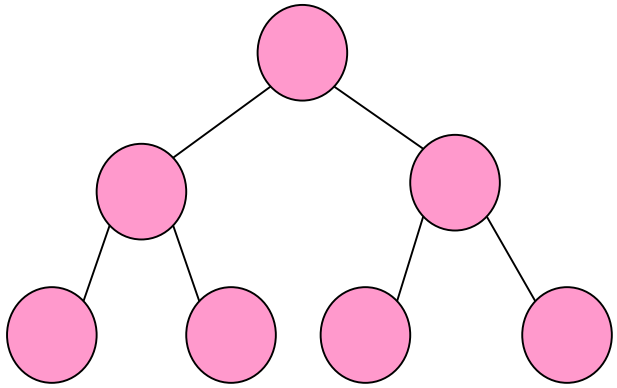
Caso Recursivo

Si tiene mas de un nodo, el nivel es:

1 + MAX(Nivel(SubIzq), Nivel(SubDer))

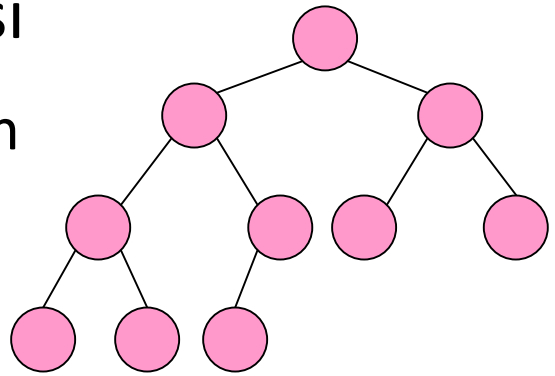
Arboles binarios llenos

- Un árbol de altura **h**, esta lleno si
  - Todas sus hojas están en el nivel h.
  - Los nodos de altura menor a h tienen siempre 2 hijos.
- Recursiva
  - Si T esta vacío
    - Entonces T es un árbol binario lleno de altura 0
  - Si no esta vacío, y tiene h>0
    - Esta lleno si los subárboles de la raíz, son ambos árboles binarios llenos de altura h-1



Arboles binarios completos

- Un árbol de altura **h** esta completo si
  - Todos los nodos hasta el nivel h-2 tienen dos hijos cada uno y
  - En el nivel h-1, si un nodo tiene un hijo derecho, todas las hojas de su subárbol izquierdo están a nivel h
- **Si un árbol esta lleno, también esta completo**



"Los elementos del último nivel están colocados de izquierda a derecha sin dejar huecos entre ellos."

Arboles binarios equilibrados

- Un **árbol binario equilibrado** es cuando
  - La diferencia de altura entre los subárboles de cualquier nodo es máximo 1
- Un árbol binario **equilibrado totalmente**
  - Los subárboles izquierdo y derecho de cada nodo tienen las misma altura: es un árbol lleno
- Un **árbol binario completo** es equilibrado
- Un **árbol binario lleno es totalmente equilibrado**

Especificación del TDA Árbol binario

Cabecera

- **Nombre**: Árbol binario *(Binary Tree)*
- **Lista de operaciones:**
  - ***Operaciones de construcción***
    - ***Inicializar (Initialize):*** Recibe un árbol **A** y lo inicializa para su trabajo normal.
    - ***Eliminar (Destroy):*** Recibe una un árbol **A** y lo libera completamente.
  - ***Operaciones de posicionamiento y búsqueda***
    - ***Raíz (Root):*** *Recibe un árbol **A** y devuelve la posición de la raíz.*
    - ***Padre (Parent):*** Recibe un árbol **A** y una posición **p**, devuelve la posición de padre de **p**.
    - ***Hijo derecho (Right Son):*** Recibe un árbol **A** y una posición **p**, devuelve la posición del hijo derecho de **p**.
    - ***Hijo izquierdo (Left Son):*** Recibe un árbol **A** y una posición **p**, devuelve la posición del hijo izquierdo de **p**.

- ***Buscar (Search):*** Recibe un árbol **A** y un elemento **e**, devuelve la posición del elemento en el árbol **A**.

Operaciones de consulta

- ***Vacía (Empty):*** *Recibe un árbol **A** y devuelve verdadero en caso de que el árbol **A** este vacío.*
- ***Nodo Nulo (Null Node):*** Recibe un árbol **A** y una posición **p**, devuelve verdadero si la posición **p** del árbol **A** es nula o incorrecta.
- ***Leer nodo (Read Node):*** Recibe un árbol **A** y una posición **p**, devuelve el elemento contenido en el nodo con posición **p** del árbol **A.**

Operaciones de modificación

- ***Nuevo Hijo Derecho (New Right Son):*** *Recibe un árbol **A,** una posición **p** y un elemento **e,** se añade a **e** como hijo derecho del nodo con posición **p.***

- ***Nuevo Hijo Izquierdo (New Left Son):*** *Recibe un árbol **A,** una posición **p** y un elemento **e,** se añade a **e** como hijo izquierdo del nodo con posición **p.***

- ***Eliminar Hijo Derecho (Delete Right Son):*** *Recibe un árbol **A** y una posición **p,** se elimina al hijo derecho y todos sus descendientes, del nodo con posición **p.***

- ***Eliminar Hijo Izquierdo (Delete Left Son):*** *Recibe un árbol **A** y una posición **p,** se elimina al hijo izquierdo y todos sus descendientes, del nodo con posición **p.***

- ***Eliminar Nodo(Delete Node):*** *Recibe un árbol **A** y una posición **p,** se elimina al nodo con posición **p** y todos sus descendientes.*

- ***Remplazar Nodo (Replace Node):*** *Recibe un árbol **A,** una posición **p** y un elemento **e,** se remplaza a **e** del nodo con posición **p** en **A.***

Descripción

- Un árbol binario de tipo T es una estructura homogénea, resultado de la concatenación de un elemento de tipo T, llamado raíz, con dos arboles binarios disjuntos, llamados subárbol izquierdo y subárbol derecho. Un árbol binario especial es el árbol binario vacio.

- Los valores del ***TAD ArbolBinario*** son arboles binarios donde cada nodo contiene un dato del tipo **Elemento**. Las posiciones de los nodos en el árbol son del tipo **Posición**. Los árboles son mutables: **Nuevo Hijo Derecho** , **Nuevo Hijo Izquierdo**, **Eliminar Hijo Derecho**, **Eliminar Hijo Izquierdo** y **Eliminar Nodo y Remplazar Nodo** añaden, eliminan y modifican respectivamente elementos y la estructura de un árbol.

Operaciones

- ***Inicializar (Initialize):*** ***recibe<-árbol(A);***
  - ***Initialize (A)***
  - Efecto: Recibe un árbol binario **A** y lo inicializa para su trabajo normal.

- ***Eliminar (Destroy):*** ***recibe<-árbol(A);***
  - ***Destroy (A)***
  - Efecto: Recibe un árbol binario **A** y lo libera completamente.

- ***Raíz (Root):*** ***recibe<-árbol(A); retorna -> posición***
  - ***Root (A)***
  - Efecto: Recibe un árbol binario **A** y retorna la posición de la raíz de **A**, si el árbol es vacío devuelve una posición nula.

- **Padre (Parent):** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***Parent(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P**, devuelve la posición de padre de **p**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida. Si **P** es la raíz se devuelve una posición nula.

- **Hijo derecho (Right Son):** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***RightSon(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P**, devuelve la posición del hijo derecho de **p**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida. Si **P** no tiene hijo derecho devuelve una posición nula.

- ***Hijo izquierdo (LeftSon):*** ***recibe<-árbol(A), posición(P); retorna -> posición***
  - ***LeftSon(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P**, devuelve la posición del hijo izquierdo de **p**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida. Si **P** no tiene hijo izquierdo devuelve una posición nula.

- ***Buscar (Search):*** ***recibe<-árbol(A), elemento (E); retorna -> posición***
  - ***Search(A,E)***
  - Efecto: Recibe un árbol binario **A** y un elemento **E**, devuelve la posición del elemento **E** en el árbol **A**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida. Si **E** no es encontrado devuelve una posición nula.

- ***Vacia (Empty):*** ***recibe<-árbol(A); retorna -> booleano***
  - ***Empty(A)***
  - Efecto: Recibe un árbol binario **A** y devuelve **verdadero** en caso de que el árbol A este vacío, devuelve **falso** en caso contrario.

- ***Nodo Nulo (Null Node):*** ***recibe<-árbol(A), posición (P); retorna -> booleano***
  - ***NullNode(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P**, devuelve verdadero si la posición **P** del árbol **A** es nula o incorrecta y devuelve falso en caso contrario.

- ***Leer Nodo(Read Node):*** ***recibe<-árbol(A), posición (P) retorna -> elemento***
  - ***ReadNode(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P,** devuelve **el elemento** en la posición **P** del árbol **A**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida..

- ***Nuevo Hijo Derecho(New Right Son):*** ***recibe<-árbol(A), posición (P), elemento E;***
  - ***NewRightSon(A,P,E)***
  - Efecto: Recibe un árbol binario **A**, una posición **P** y un elemento **E**, se añade un nodo que contenga **E** como hijo derecho del nodo con posición **P**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida. Si el árbol **A** es vacío se agrega a **un nodo raíz con E.** si **P** ya tiene un hijo derecho, se cancela la operación.

- ***Nuevo Hijo Izquierdo(New Left Son):*** ***recibe<-árbol(A), posición (P), elemento E;***
  - ***NewLeftSon(A,P,E)***
  - Efecto: Recibe un árbol binario **A**, una posición **P** y un elemento **E**, se añade un nodo que contenga **E** como hijo izquierdo del nodo con posición **P**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida. Si el árbol **A** es vacío se agrega a **un nodo raíz con E;** si **P** ya tiene un hijo Izquierdo, se cancela la operación.

- ***Eliminar Hijo Derecho (Delete Right Son):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteRightSon(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición se elimina al hijo derecho y todos sus descendientes del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida y tiene un hijo derecho.

- ***Eliminar Hijo Izquierdo(Delete Left Son):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteLeftSon(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición se elimina al hijo izquierdo y todos sus descendientes del nodo con posición **P**.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida y tiene un hijo izquierdo.

- ***Eliminar Nodo(Delete Node):*** ***recibe<-árbol(A), posición (P);***
  - ***DeleteNode(A,P)***
  - Efecto: Recibe un árbol binario **A** y una posición **P,** se elimina al nodo con posición **P** y todos sus descendientes.
  - Requerimientos: El árbol **A** es no vacío y la posición **P** es una posición valida.

- ***Remplazar Nodo(Replace Node):*** ***recibe<-árbol(A), posición (P), elemento (E);***
  - ***ReplaceNode(A,P)***
    - Efecto: Recibe un árbol binario **A,** una posición **P** y un elemento **E,** se remplaza a **E** del nodo con posición **P** en **A**.
  - Requerimientos: El árbol binario **A** es no vacío y la posición **P** es una posición valida.

Observaciones

- Dos árboles binarios son **distintos** cuando sus estructuras (la distribución de los arcos) son diferentes.

- Dos árboles son **similares** cuando sus estructuras son idénticas, pero la información que contienen sus nodos difiere entre si.

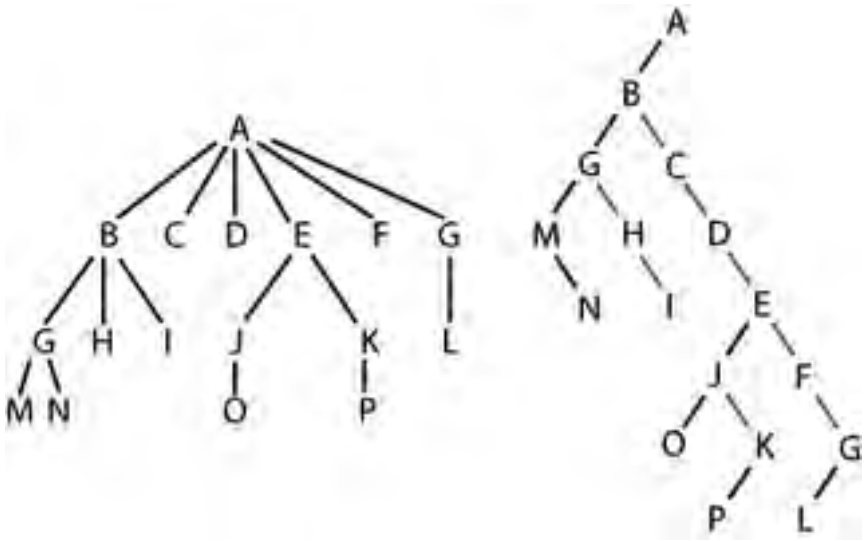
- Dos árboles binarios **equivalente** son aquellos que son similares y además los nodos contienen la misma información.

- Se define un **árbol binario completo** como un árbol en el que todos sus nodos, excepto los del ultimo nivel, tienen dos hijos: el subárbol izquierdo y el subárbol derecho.

- **Recorrer un árbol** significa visitar sus nodos, ya sea para consultar la información de cada uno o buscar alguno de estos para su modificación o eliminación.

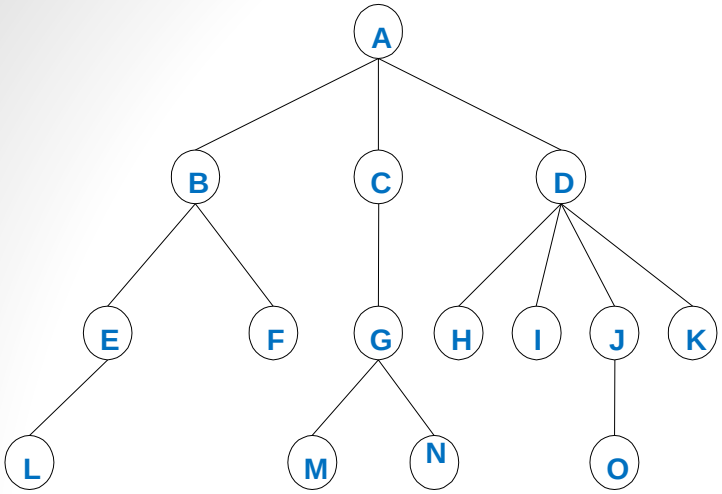
Conversión de arboles generales a binarios

- Los árboles binarios, se aplican en la solución de muchos problemas. Además, su uso se ve favorecido por su dinámica, la no linealidad entre sus elementos y por su sencilla programación. Por lo tanto resulta útil poder convertir árboles generales, con 0 a *n* hijos, en árboles binarios.

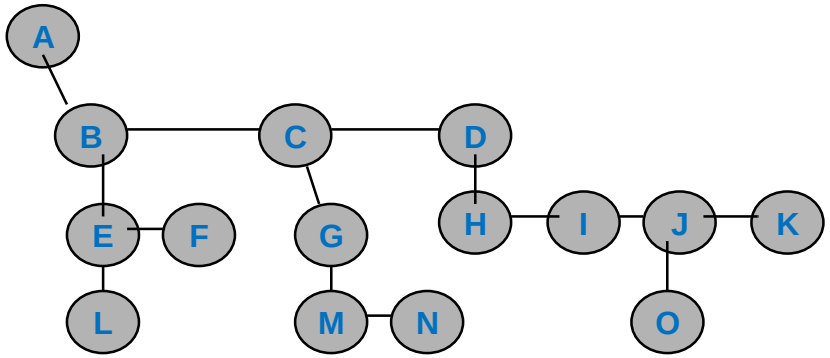


Conversión de arboles generales a binarios

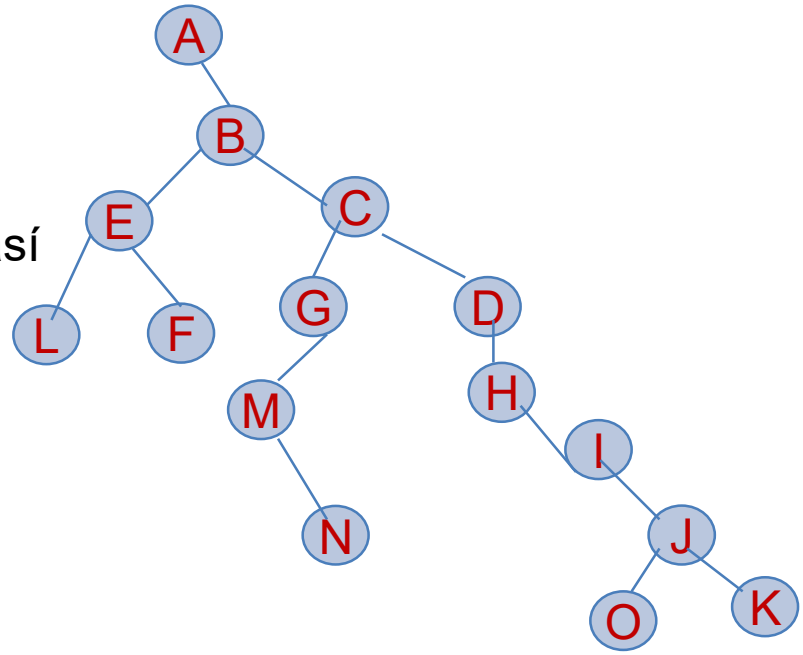
1. **Enlazar los hijos** del nodo en **forma horizontal** (Los hermanos)
2. **Relacionar en forma vertical** el nodo padre con el hijo que se encuentra *mas a la izquierda*. Además se debe **eliminar el vinculo** de ese padre con el resto de sus hijos.
3. **Rotar el diagrama** resultante, aproximadamente 45 grados en sentido horario, y así se obtendrá el árbol binario correspondiente.

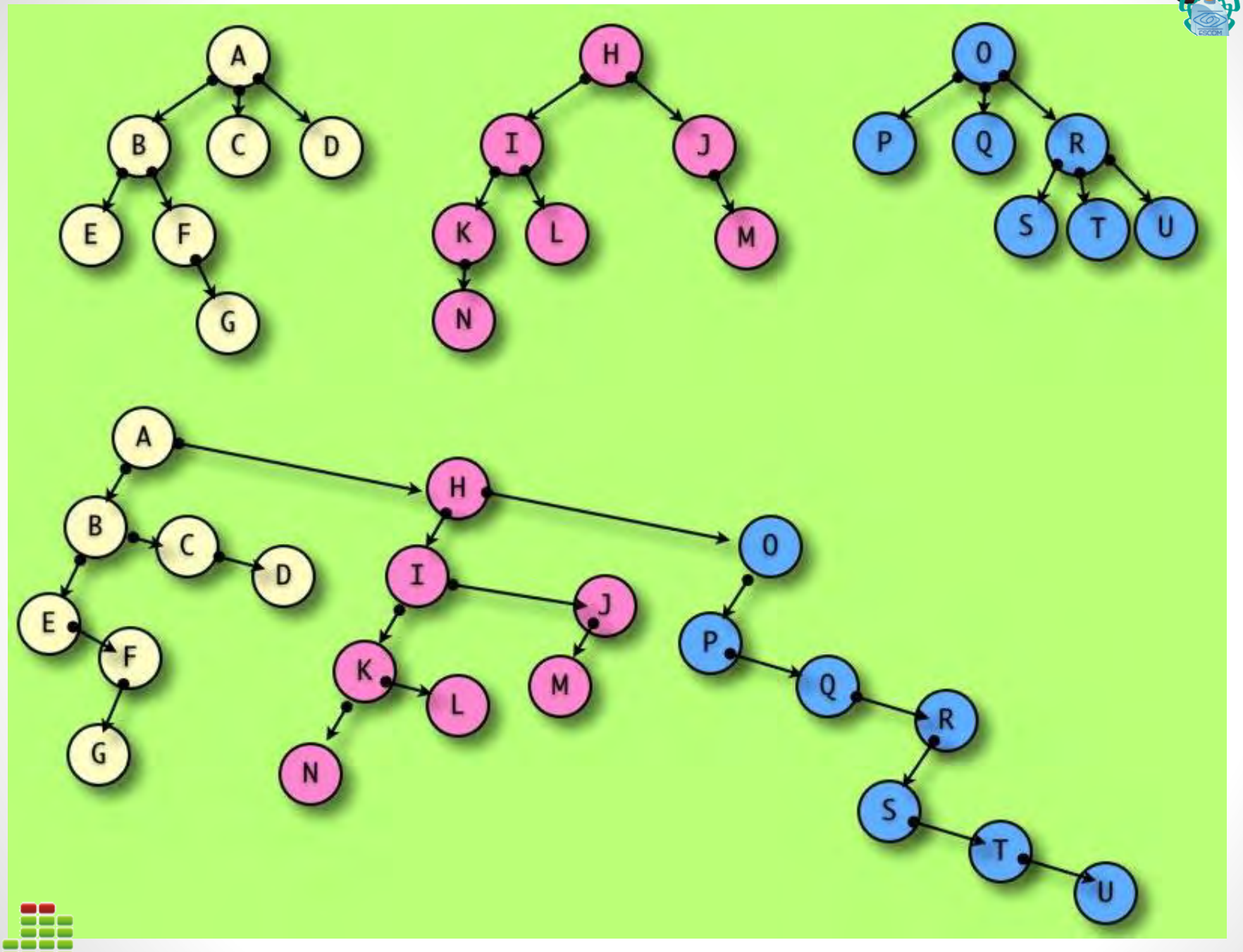


=



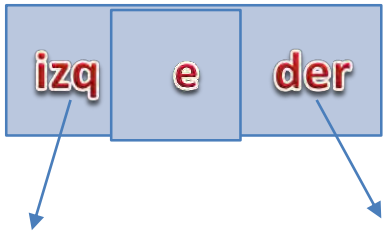
que da así




A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
A
H
O
B
C
D
I
J
E
F
P
Q
R
K
L
M
S
G
N
T
U

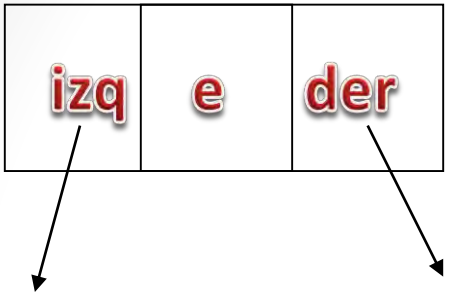
- Las dos maneras más comunes de representar un árbol binario en memoria son:

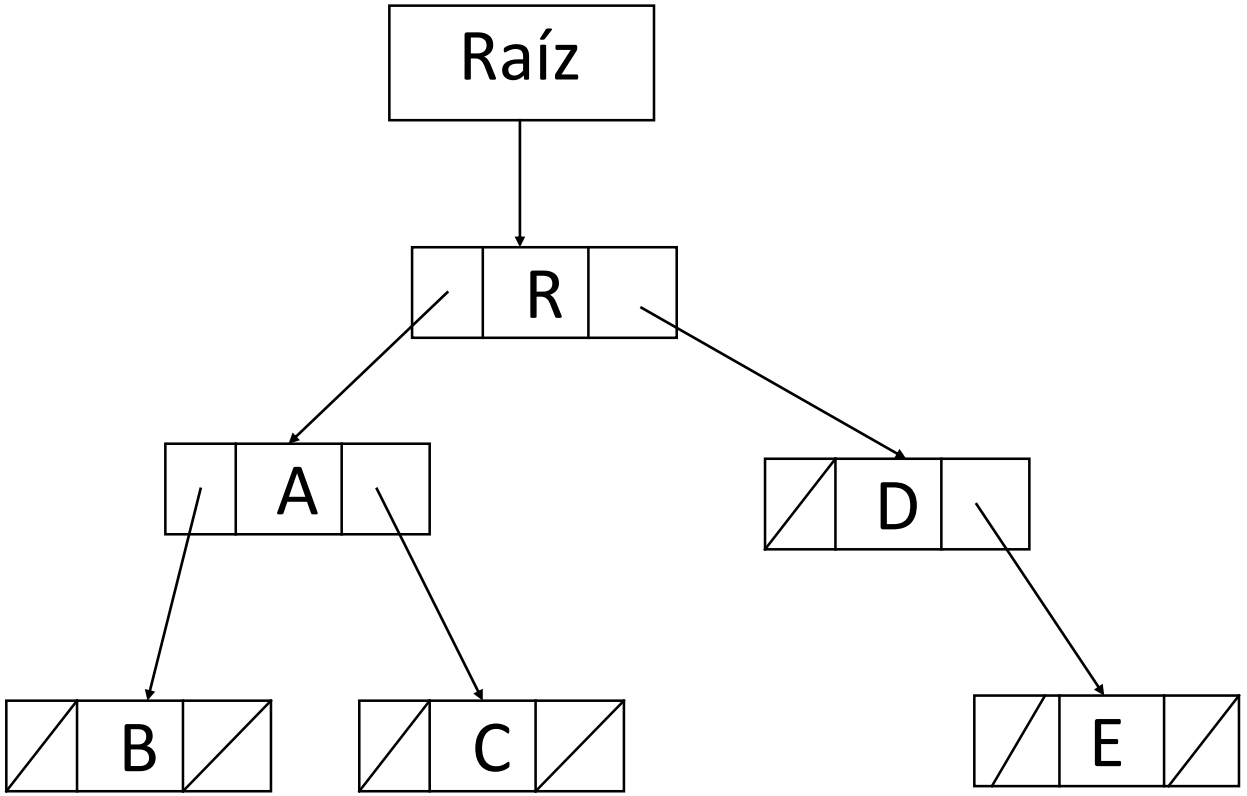
1. Por medio de datos tipo puntero
2. Por medio de arreglos *(Dinámicos)*



- Los nodos del árbol binario se representan como bloques de memoria, cada uno de ellos con tres campos. En uno se almacenará la información del nodo. Los dos restantes se utilizarán para apuntar los subárboles izquierdo y derecho, respectivamente, del nodo en cuestión.

- Cada nodo tiene la siguiente forma:

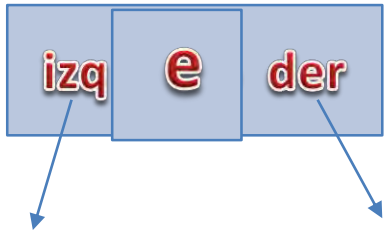




Representación de árboles binarios en memoria

- **Izq:** Es el campo donde se almacena la dirección del subárbol izquierdo del nodo.
- **e:** Representa el campo donde se almacena el elemento que almacena el nodo (información).
- **Der:** Es el campo donde se almacena la dirección del subárbol derecho del nodo.

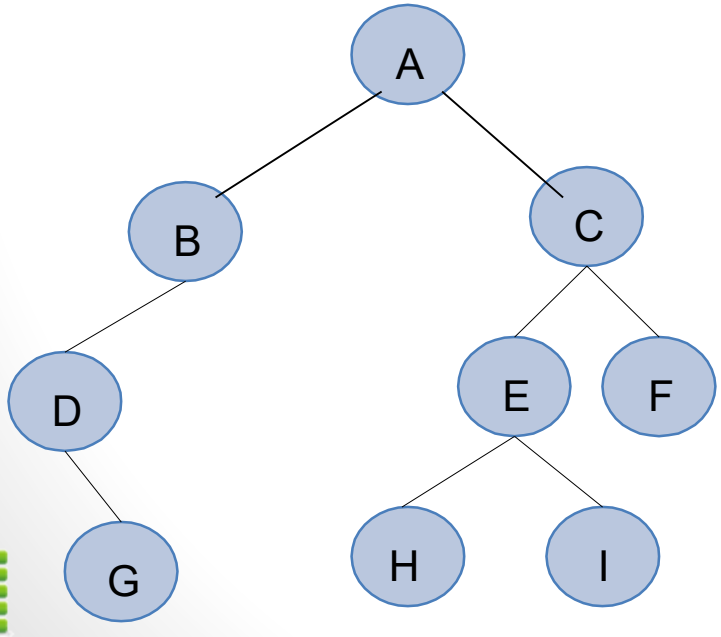
```
//Estructura en C para representar un nodo de un árbol binario
typedef struct nodo
{
        //Apuntador al hijo izquierdo
        struct nodo *izq;
        //Información (elemento)
        elemento e;
        //Apuntador al hijo derecho
        struct nodo *der;
}nodo;
```



Recorrido en árboles binarios

- **Recorrer un árbol** significa visitar los nodos del árbol en forma ordenada, de tal manera que **todos los nodos del mismo sean visitados una sola vez**.

- Para árboles binarios, se definen los recorridos en ***preorden, inorden y postorden.***



Preorden = A B D G C E H I F

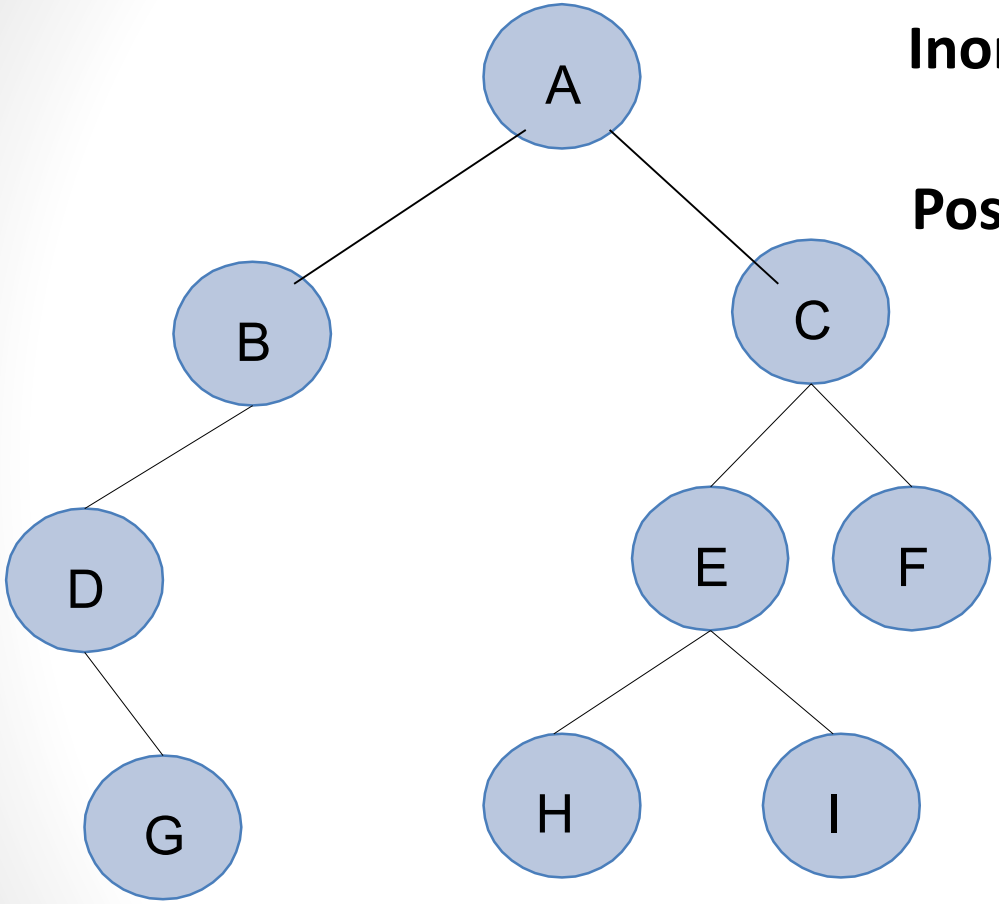
Inorden= D G B A H E I C F

Postorden= G D B H I E F C A

- Recorrido en **preorden**
  - Visitar la raíz
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho

- Recorrido en **inorden**
  - Recorrer el subárbol izquierdo
  - Visitar la raíz
  - Recorrer el subárbol derecho

- Recorrido en **postorden**
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
  - Visitar la raíz

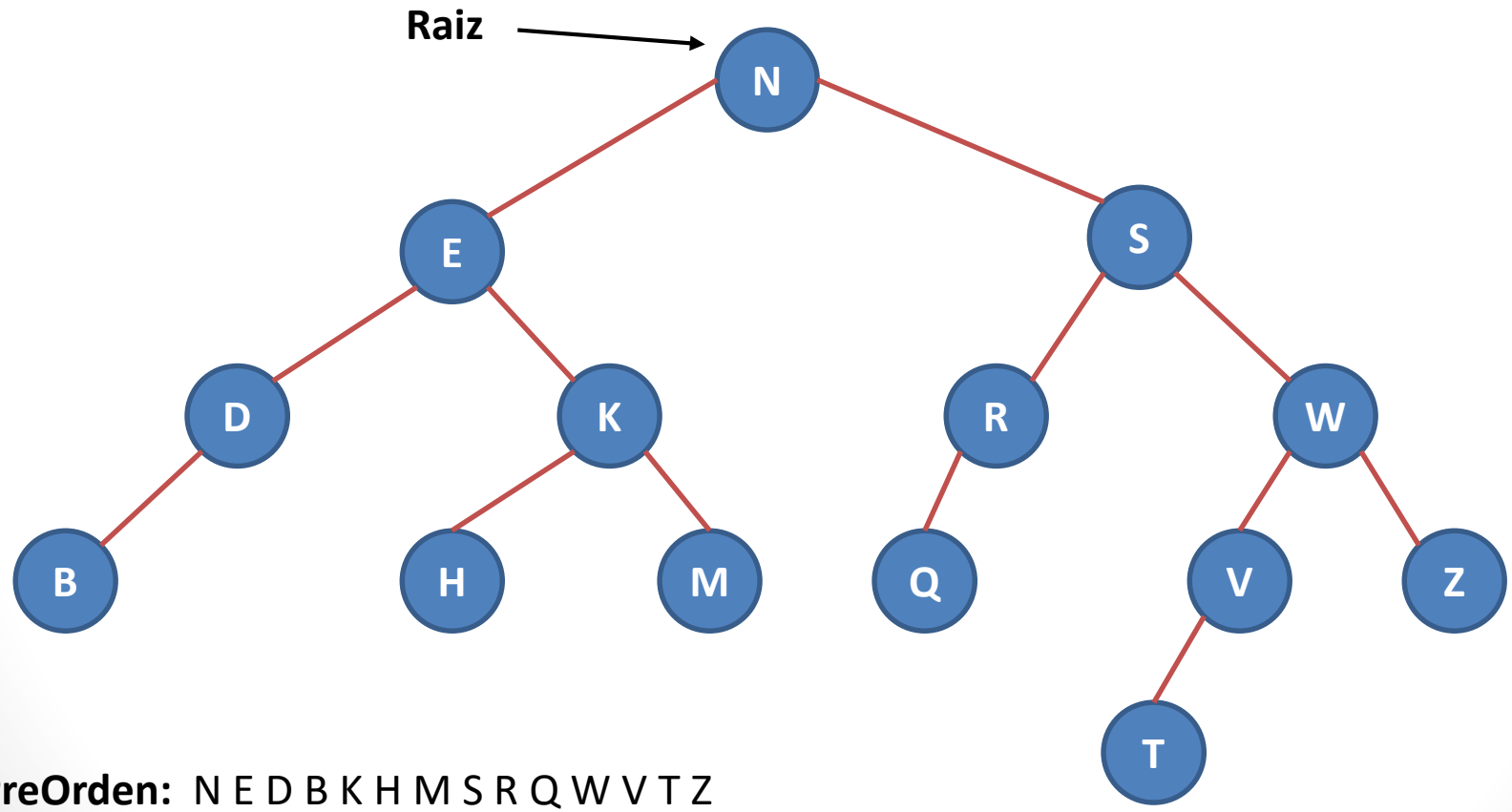


Preorden = A B D G C E H I F

Inorden= D G B A H E I C F

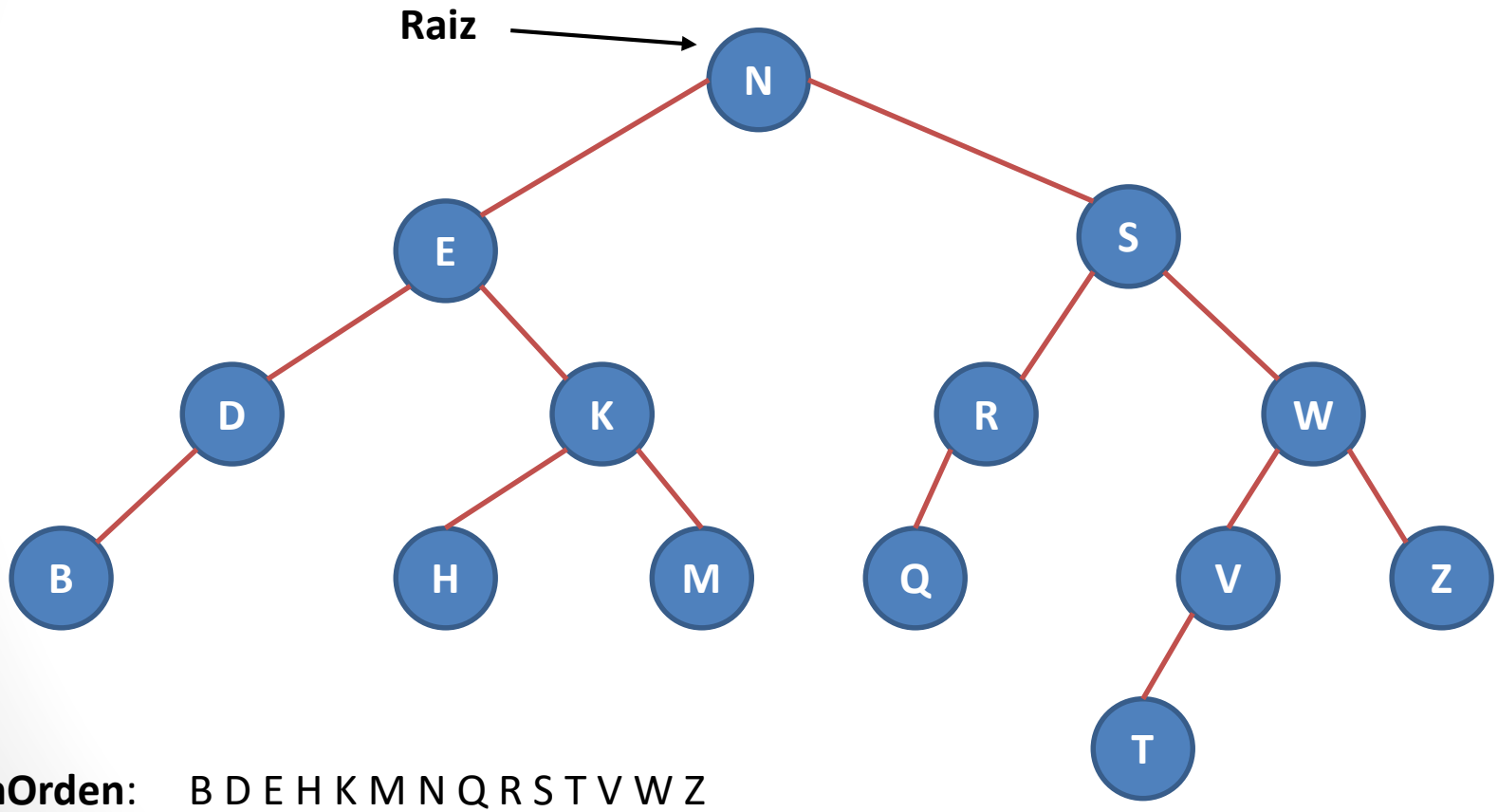
Postorden= G D B H I E F C A

Recorrido Preorden



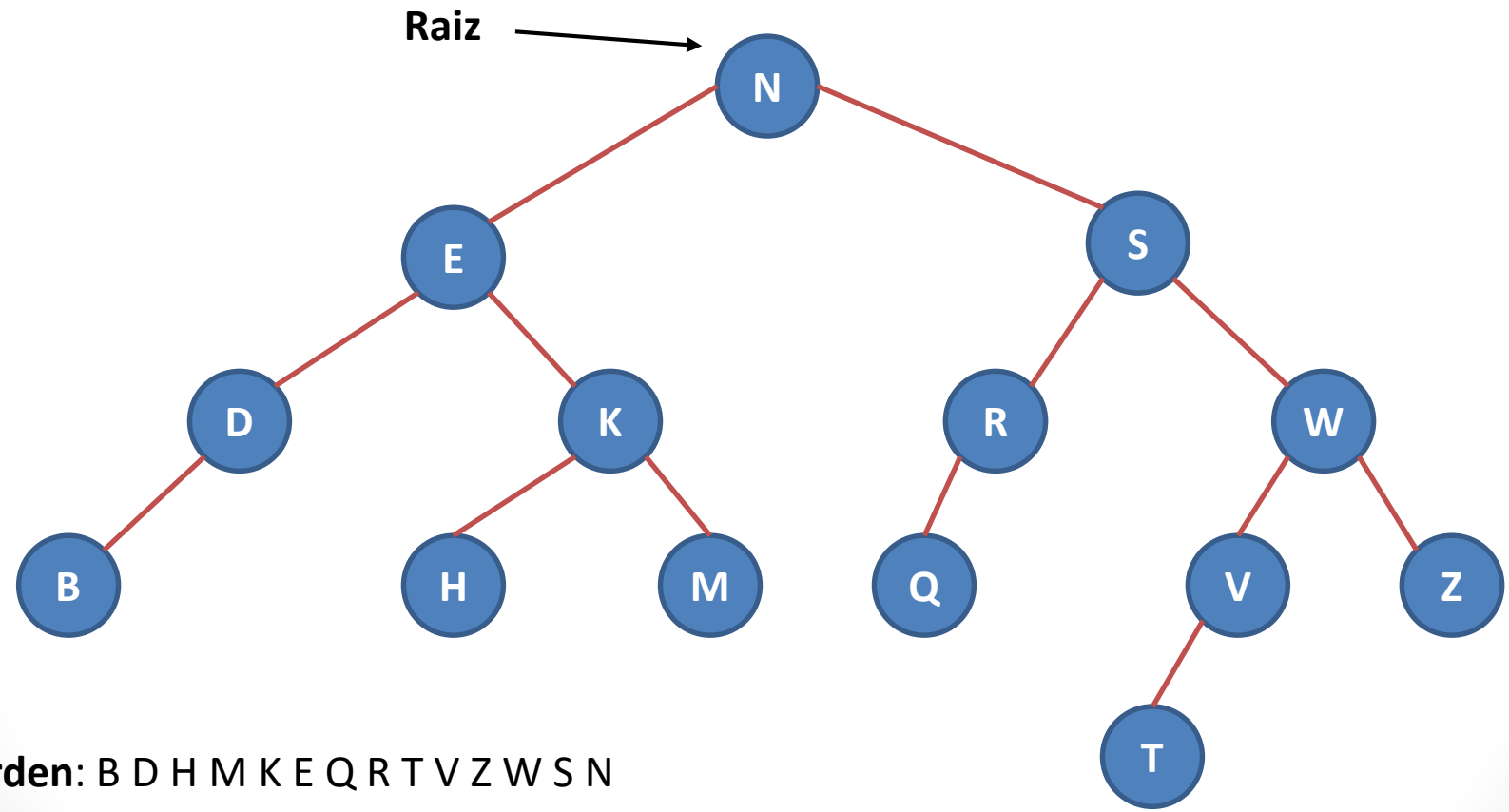
PreOrden: N E D B K H M S R Q W V T Z

Recorrido Inorden



InOrden: B D E H K M N Q R S T V W Z

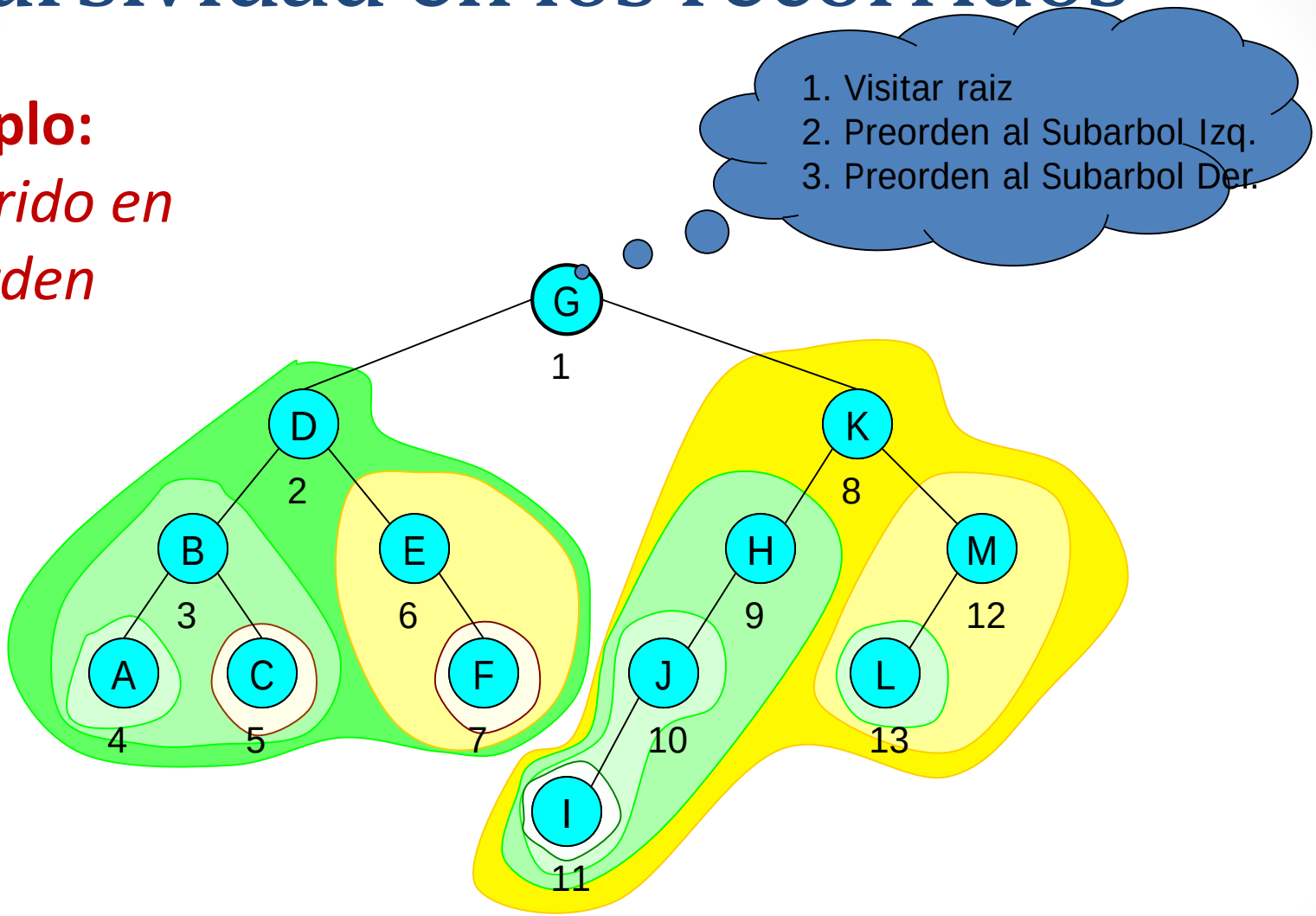
Recorrido en Postorden



Postorden: B D H M K E Q R T V Z W S N

Recursividad en los recorridos

Ejemplo:
recorrido en preorden



G-D-B-A-C-E-F-K-H-J-I-M-L