Estructura de Datos y Algoritmos 1

Teórico #9:

Estructuras arborescentes

Árboles Binarios.

Árboles Generales

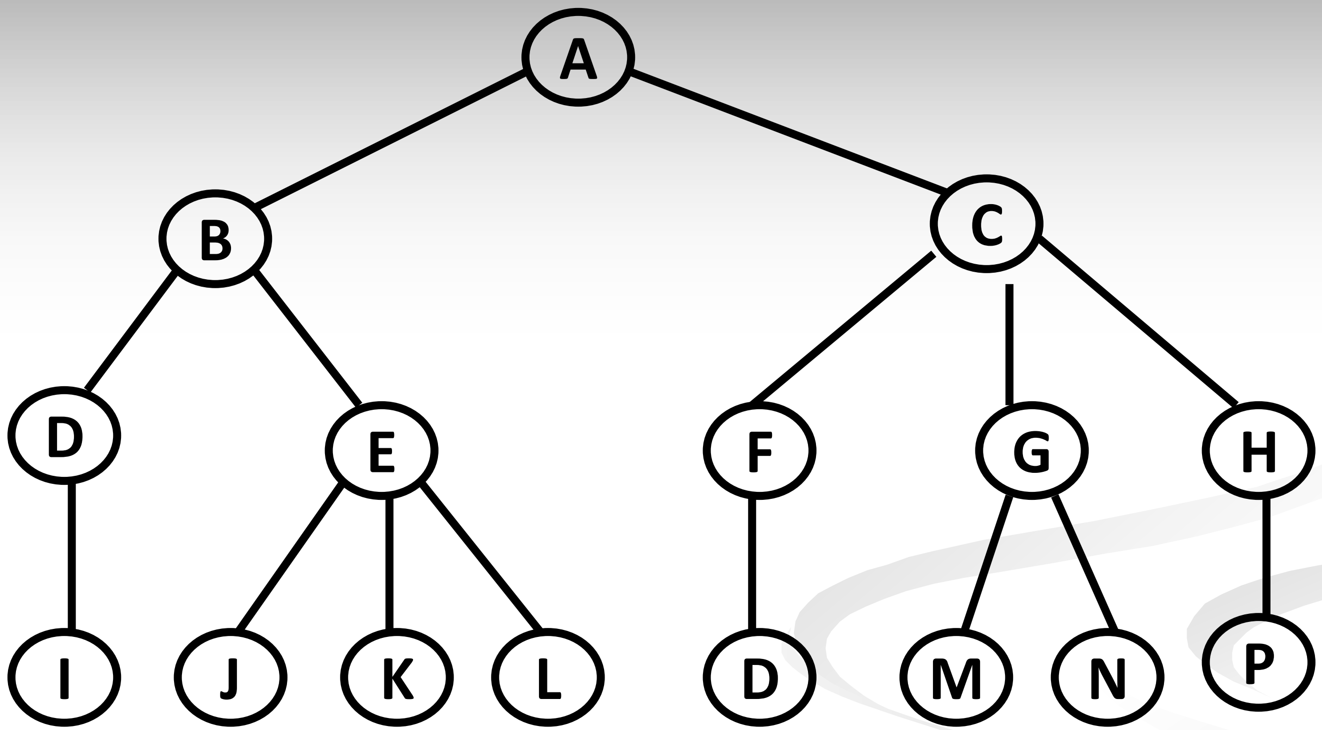
Introducción

- Una lista es una sucesión de cero o más elementos de un tipo dado. En este sentido, es una estructura lineal.

- La estructura de árbol expresa una relación jerárquica entre los elementos de un conjunto dado. En este sentido, es una estructura no lineal.

- En algunos problemas de las ciencias, y en particular de las ciencias de la informática, la estructura de árbol es la forma más natural y útil de representación de las instancias del problema.
    - Búsqueda y ordenamiento
    - Evaluación de expresiones.
    - Codificación de la información
    - Análisis de la sintaxis de los lenguajes de programación

Representación Gráfica más usual de árbol



Definición recursiva de árbol

Un **árbol** $\mathbf{T}$ es un conjunto finito de elementos (que denominaremos nodos) con las siguientes propiedades:

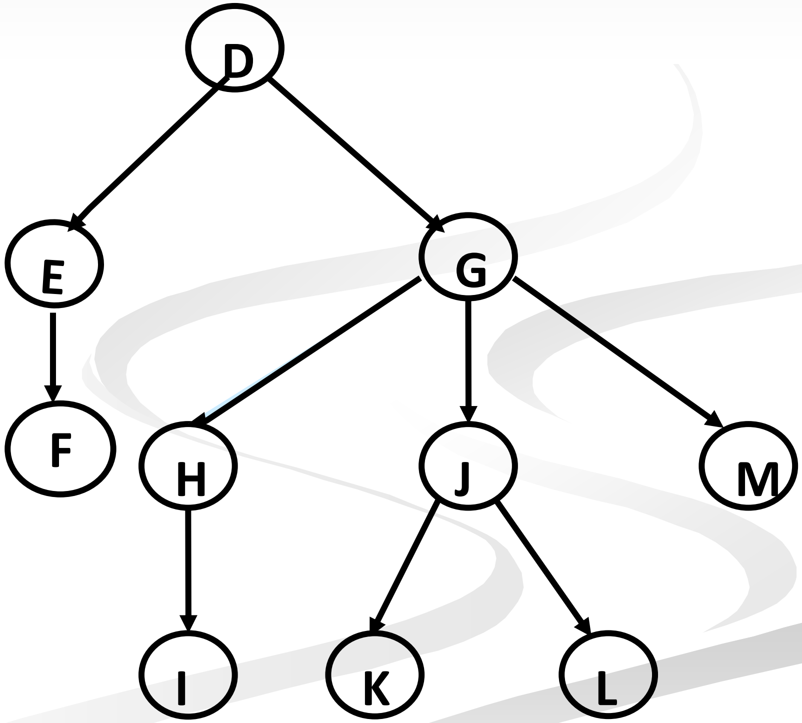
1. La estructura $\mathbf{T}$ puede no tener elemento, lo que se denomina **árbol vacío**.
2. Si el conjunto no es vacío, tiene la siguiente estructura:
   - Existe un nodo especial $\mathbf{r}$ denominado **raíz del árbol**
   - Los restantes nodos están particionados en n (n≥0) **subconjuntos disjuntos** $\mathbf{T}_1$, $\mathbf{T}_2$, …, $\mathbf{T}_n$, siendo cada uno de ellos un árbol

Terminologías

El **grado de un nodo** es el número de subárboles asociados con el nodo. El **grado del árbol** es el grado de la raíz.

Por ejemplo, el grado de:

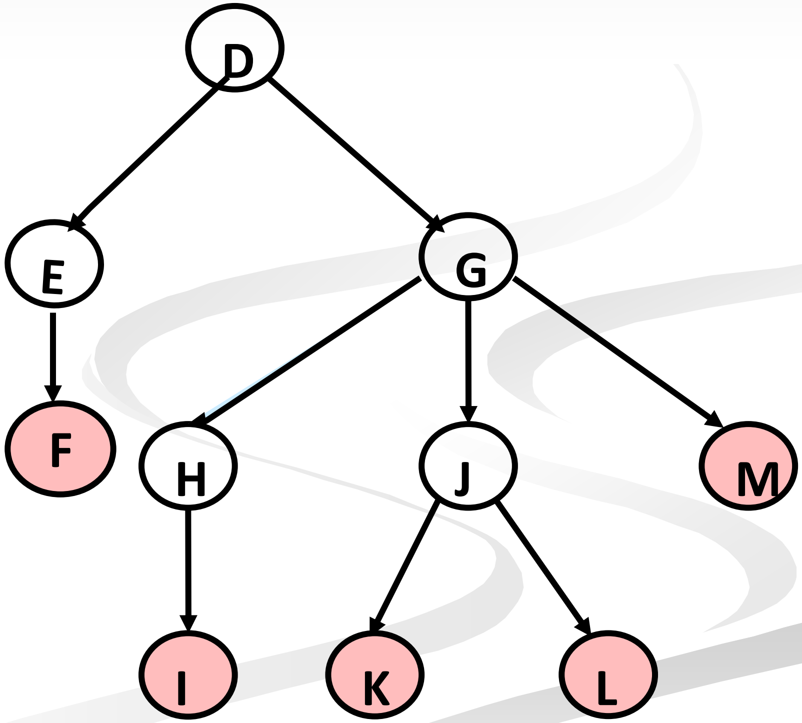
- la raíz D del árbol es 2
- del nodo G es 3
- grado del árbol es 2



Terminologías

Los nodos de grado cero (no tienen subárboles asociados) son denominados **“hojas”**.

Por ejemplo son hojas los nodos F, I, K, L y M

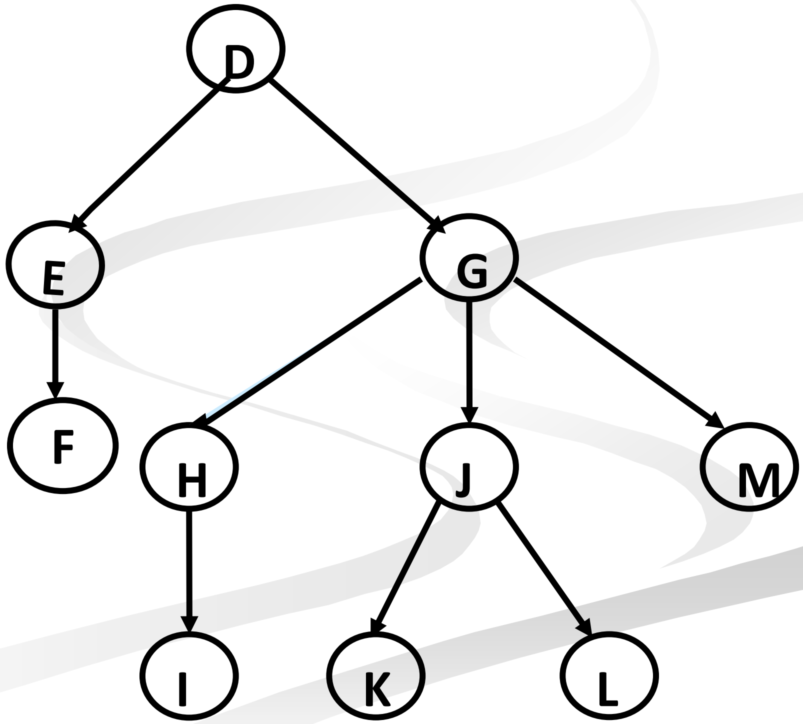


Terminologías

A la raíz $\mathbf{r}$ del árbol $\mathbf{T}$ se le denomina **“padre”** de todas las raíces $\mathbf{r}_i$ de los subárboles $\mathbf{T}_i$ de $\mathbf{T}$.

Cada raíz $\mathbf{r}_i$ del subárbol $\mathbf{T}_i$ del árbol $\mathbf{T}$ de raíz $\mathbf{r}$ es denominado **“hijo”** de $\mathbf{r}$.

- D es padre de E y G
- G es padre de H, J y M
- K no es padre de ningún nodo
- E y G son hijos de D
- H, J y M son hijos de G
- D no es hijo de ningún nodo



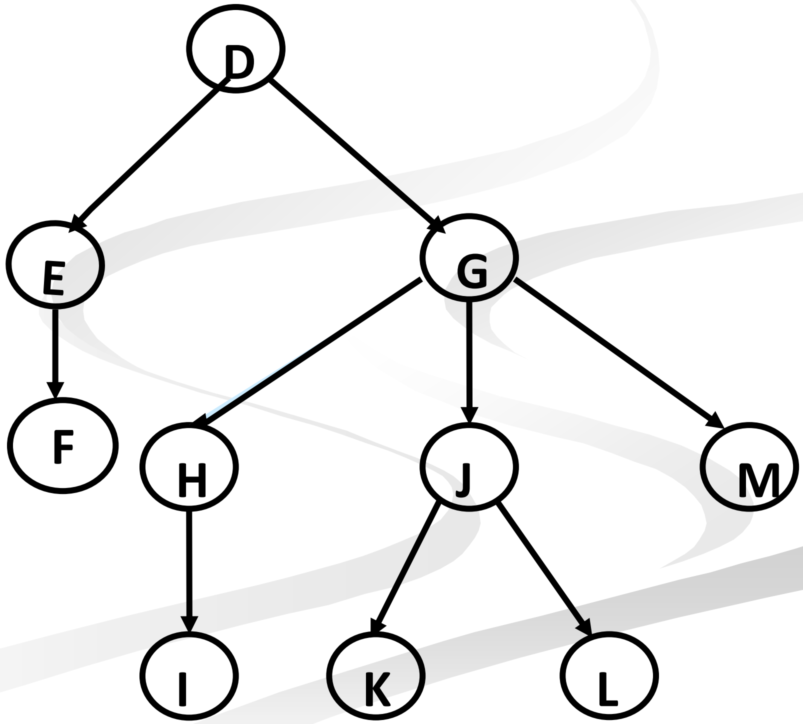
Terminologías

Dos raíces $\mathbf{r}_i$ y $\mathbf{r}_j$ de dos subárboles distintos $\mathbf{T}_i$ y $\mathbf{T}_j$ del árbol $\mathbf{T}$ son denominados **“hermanos”**.

E y G son hermanos
(pues son las raíces de hijos de D)

H, J y M son hermanos
(pues son las raíces de hijos de G)

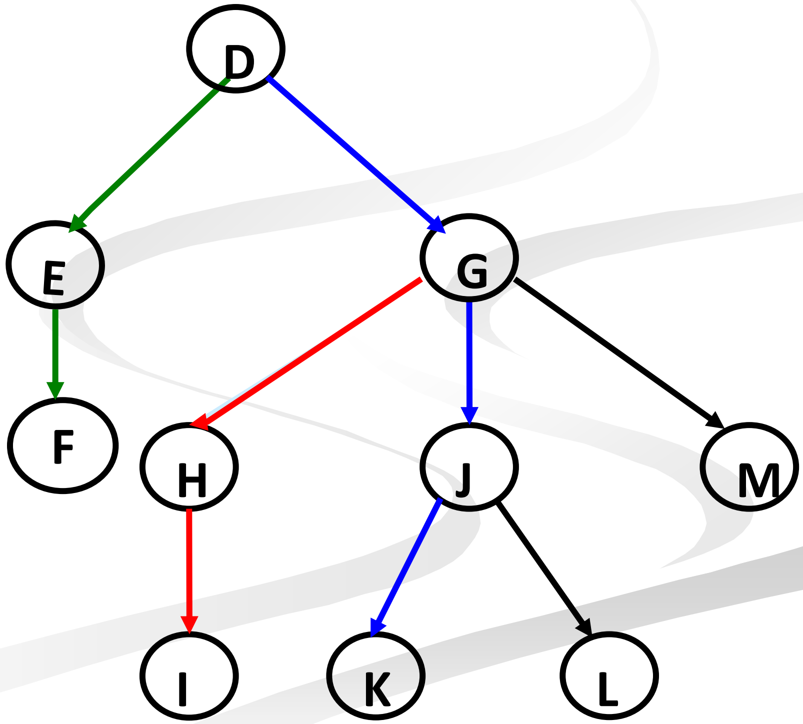
F no tiene hermanos



Terminologías

Dado un árbol $\mathbf{T}$, un "**camino**" es una sucesión no vacía $\mathbf{r}_1, \ldots, \mathbf{r}_n$ de nodos de $\mathbf{T}$, que cumple que para $1 \le i < n$ se tiene que $\mathbf{r}_i$ es padre de $\mathbf{r}_{i+1}$.

La "**longitud de un camino**" es el número de nodos del camino menos 1

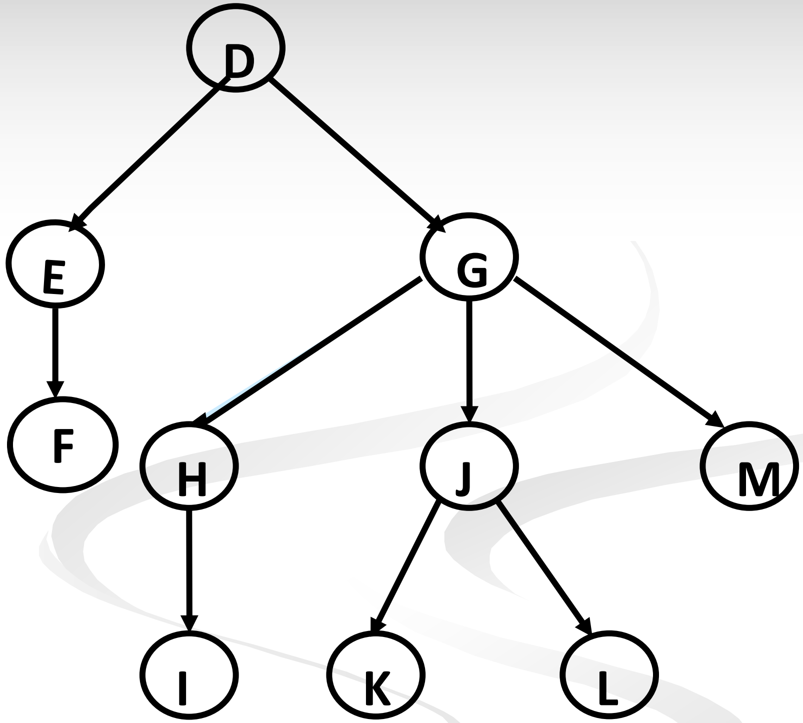


G,H,I (longitud 2)
D,E,F (longitud 2)
D,G,J,K (longitud 3)
son caminos

D,E,H no es camino

Terminologías

El "**nivel o profundidad**" de un nodo $\mathbf{r_i}$ de $\mathbf{T}$ es la longitud incrementada en 1 del único camino de la raíz $\mathbf{r}$ al nodo $\mathbf{r_i}$



Por ejemplo:

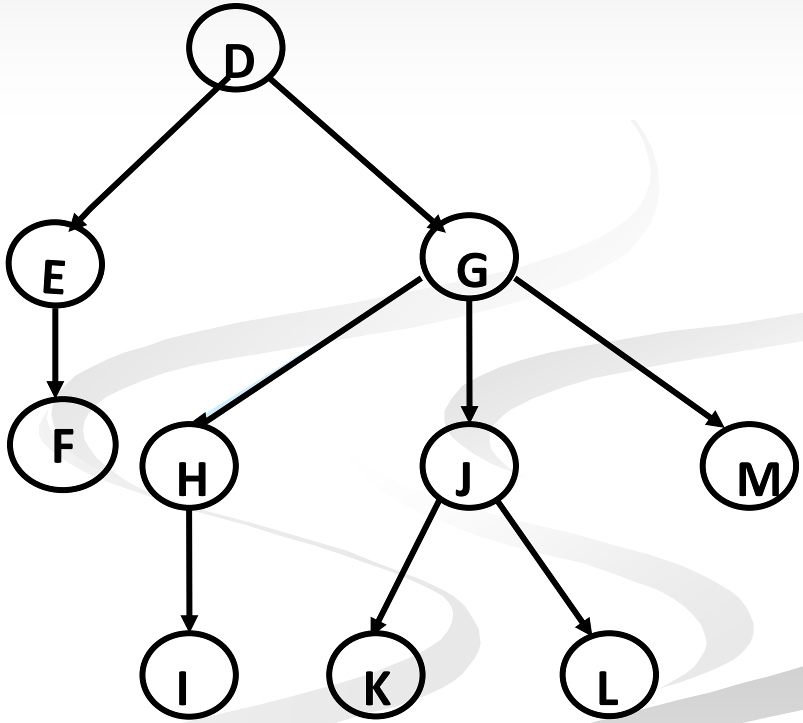
- el nivel de D es 1
- el nivel de J es 3
- el nivel de K es 4

$$Nivel(r_i) = \begin{cases} 1 & \text{Si } r_i \text{ es raiz} \\ 1 + Nivel(r_p) & \text{Si } r_i \text{ es hijo de } r_p \end{cases}$$

Terminologías

La "**altura o peso de un nodo**" r_i de T es la longitud del mayor camino de r_i a las hojas.

La **altura de un árbol** es la altura de la raíz.



Por ejemplo, la altura de:

G es 2

M es 0

D es 3

Y la altura del árbol es 3

ya que la altura de la raíz es 3

Recorridos

La mayoría de los algoritmos sobre árboles tienen en común que visitan sistemáticamente los nodos del árbol realizando alguna operación sobre ellos.
Este proceso es denominado **recorrido** del árbol.

Esencialmente existen dos enfoques diferentes para visitar sistemáticamente todos los nodos de un árbol:

recorrido en profundidad

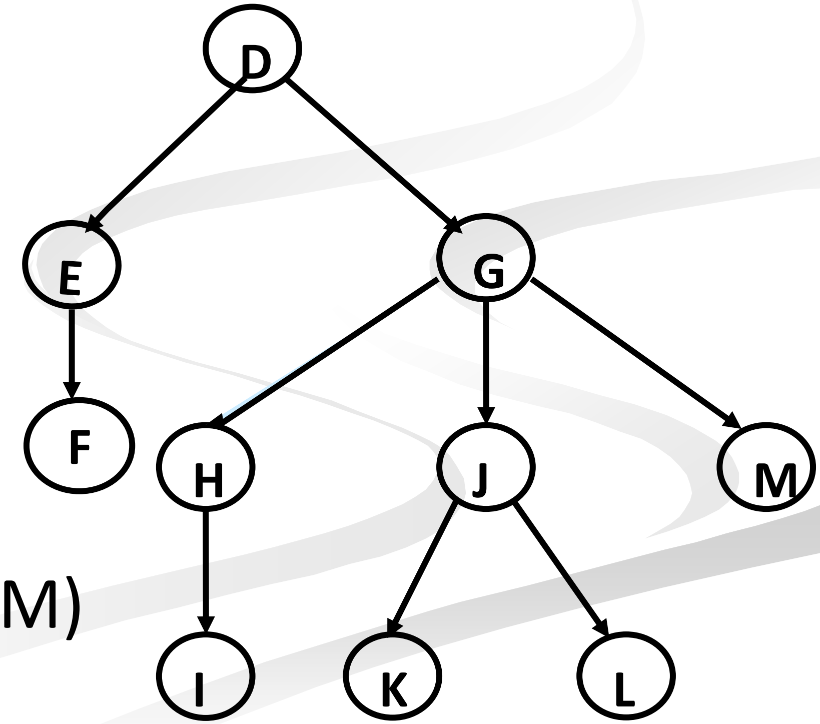
- preorden
- entreorden
- posorden

recorrido a lo ancho.

Recorrido en preorden

El recorrido en **preorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Visitando la raíz del árbol
2. Recorriendo en **preorden** cada uno de los subárboles de la raíz.

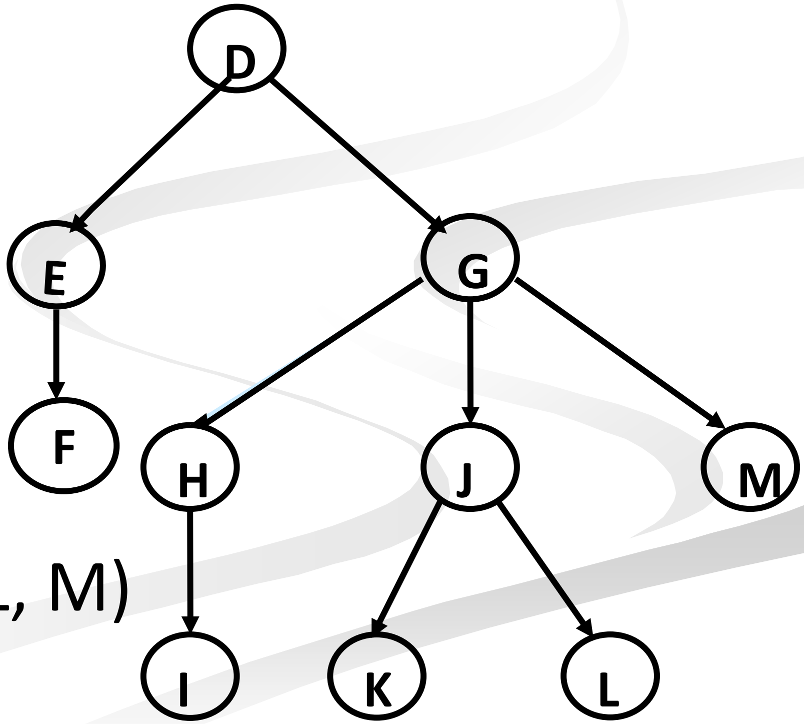


preorden : (D, E, F, G, H, I, J, K, L, M)

Recorrido en entreorden

El recorrido en **entreorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Recorriendo en **entreorden** uno de los subárboles de la raíz
2. Visitando la raíz del árbol
3. Recorriendo en **entreorden** cada uno de los restantes subárboles de la raíz.

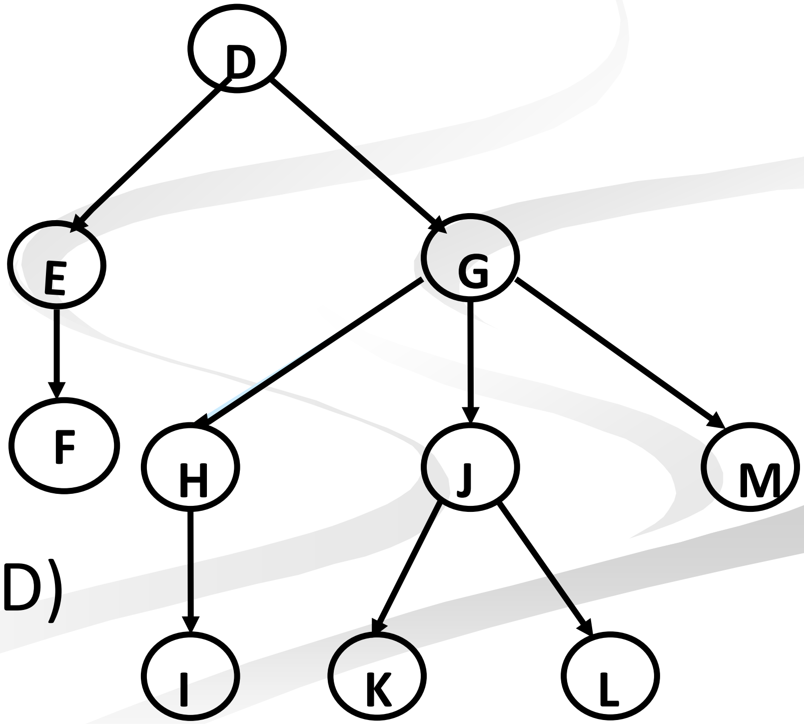


entreorden : (F, E, D, I, H, G, K, J, L, M)

Recorrido en posorden

El recorrido en **posorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Recorriendo en **posorden** cada uno de los subárboles de la raíz
2. Visitando la raíz del árbol

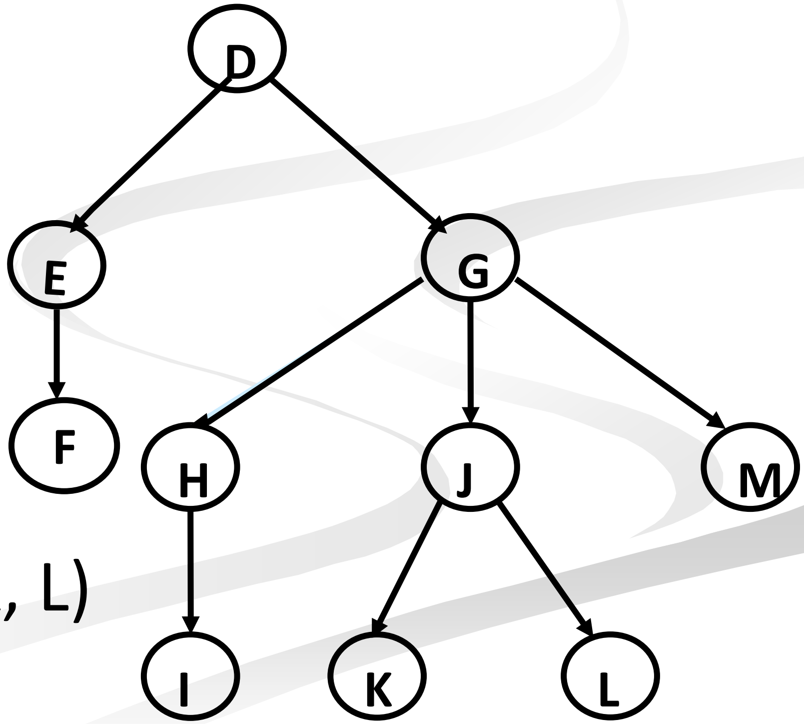


posorden : (F E, I, H, K, L, J, M, G, D)

Recorrido a lo ancho

En el recorrido **a lo ancho** de un árbol se visitan los nodos en el orden de los niveles en el árbol.

En este recorrido se visita el nodo del nivel 0 (esto es la raíz), a continuación todos los nodos del nivel 1, después todos los nodos del nivel 2 y así sucesivamente.



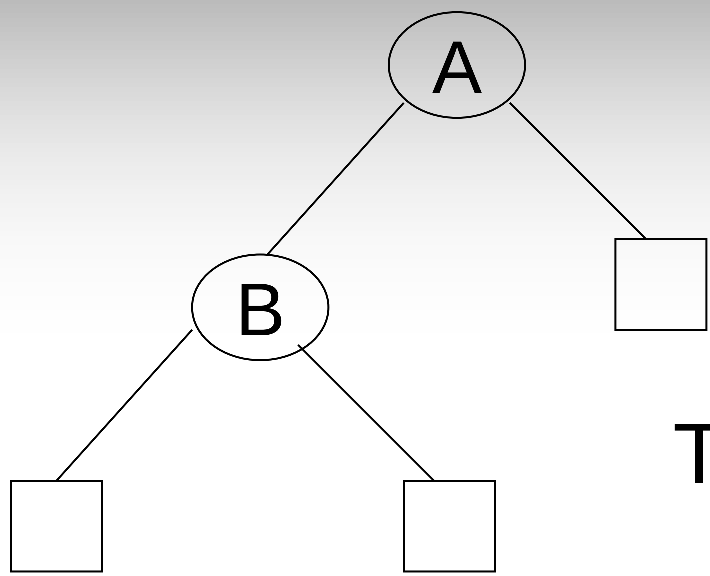
A lo ancho: (D, E, G, F, H, J, M, I, K, L)

Definición de árbol binario

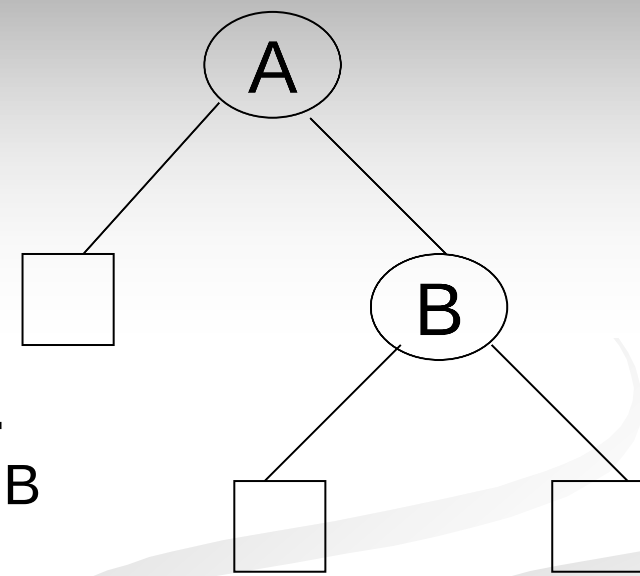
Un **árbol** $\mathbf{T}$ es un conjunto finito de elementos (que denominaremos nodos) con las siguientes propiedades:

1. La estructura $\mathbf{T}$ puede no tener elemento, lo que se denomina **árbol vacío**.
2. Si el conjunto no es vacío, El conjunto consiste de una **raíz** $\mathbf{r}$ y exactamente dos árboles binarios distintos $\mathbf{T_{Izq}}, \mathbf{T_{Der}}$ $\mathbf{T = \{r,\ T_{Izq}\ ,\ T_{Der}\}}$
   El árbol $\mathbf{T_{Izq}}$ es llamado **subárbol izquierdo** de $\mathbf{T}$ y el árbol $\mathbf{T_{Der}}$ es denominado **subárbol derecho** de $\mathbf{T}$.

Ejemplos de árboles binarios



$T_A \neq T_B$



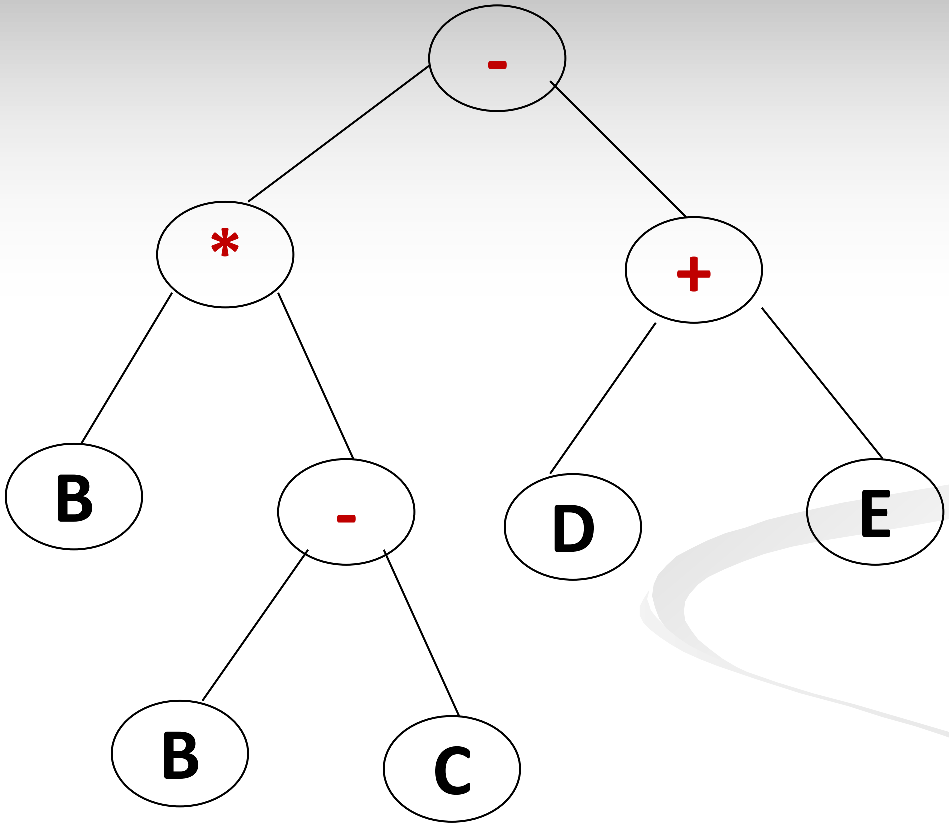
$T_A=\{A, T_{izq} , T_{der} \}$

T_{izq} : $\{B, Ø, Ø\}$
T_{der} : $Ø$

$T_B=\{A, T_{izq} , T_{der} \}$

T_{izq} : $Ø$
T_{der} : $\{B, Ø, Ø\}$

Ejemplos de árboles binarios



Por comodidad no dibujamos los árboles vacíos

Redefinición de Terminologías en árboles binarios

El **grado** de todos los nodos de un árbol binario es **dos**.

Los nodos cuyos ambos hijos son vacíos, son denominados **“hojas”**.

Altura de árbol binario: Definición recursiva

$$altura\text{: ArbolBinario} \rightarrow \mathbb{N}$$

$$altura(\,[\,]\,) = 0$$

$$altura([r, T_{izq}, T_{der}]) =$$
$$1 + \max(altura(T_{izq}),\ altura(T_{der}))$$

Cantidad de nodos de árbol binario

$$cantNodos: \text{ArbolBinario} \rightarrow \mathbb{N}$$

$$cantNodos(\,[\,]\,) = 0$$

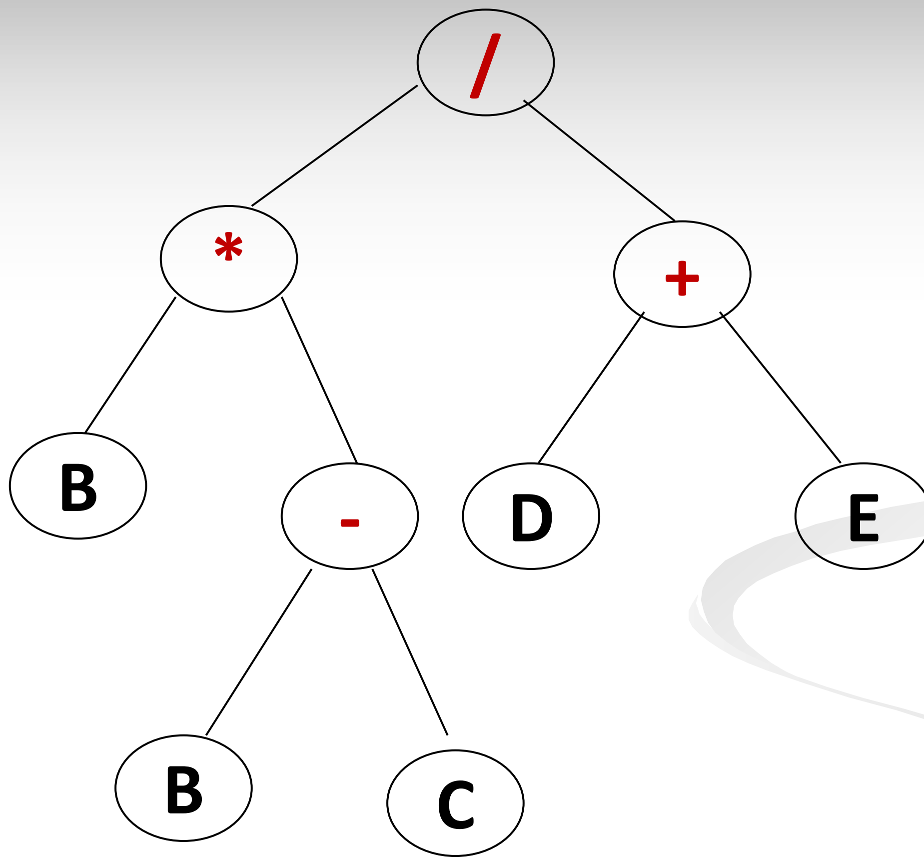
$$cantNodos([r, T_{izq}, T_{der}]) =$$
$$1 + cantNodos(T_{izq}) + cantNodos(T_{der})$$

Árbol Binario: Recorrido en preorden

Recorrido en preorden de un árbol binario diferente del vacío $\mathtt{T} = \{\mathtt{r},\ \mathtt{T}_{\mathtt{Izq}}\ ,\ \mathtt{T}_{\mathtt{Der}}\}$

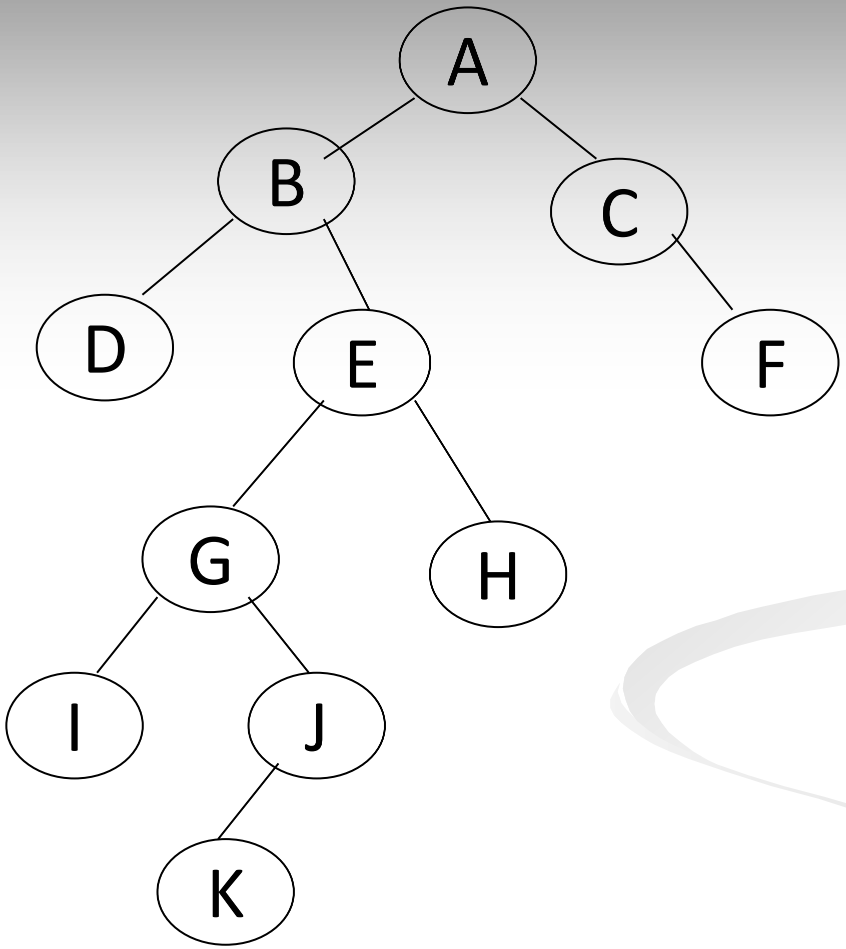
1. Se visita la raíz del árbol
2. Se recorre en preorden el subárbol izquierdo $\mathtt{T}_{\mathtt{Izq}}$.
3. Se recorre en preorden el subárbol derecho $\mathtt{T}_{\mathtt{Der}}$.

Árbol Binario: Recorrido en preorden



/, *, B, -, B, C, +, D, E

Árbol Binario: Recorrido en preorden



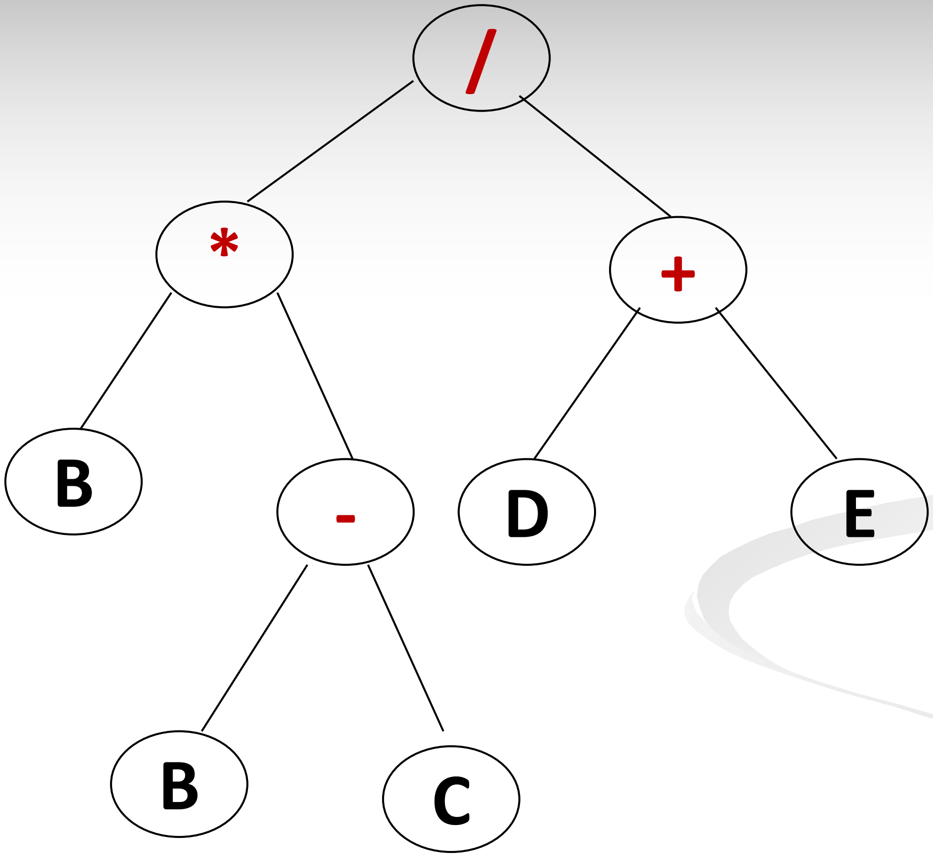
A, B, D, E, G, I, J, K, H, C, F

Árbol Binario: Recorrido en entreorden

Recorrido en entreorden de un árbol binario diferente del vacío $\mathbf{T = \{r,\ T_{Izq}\ ,\ T_{Der}\}}$

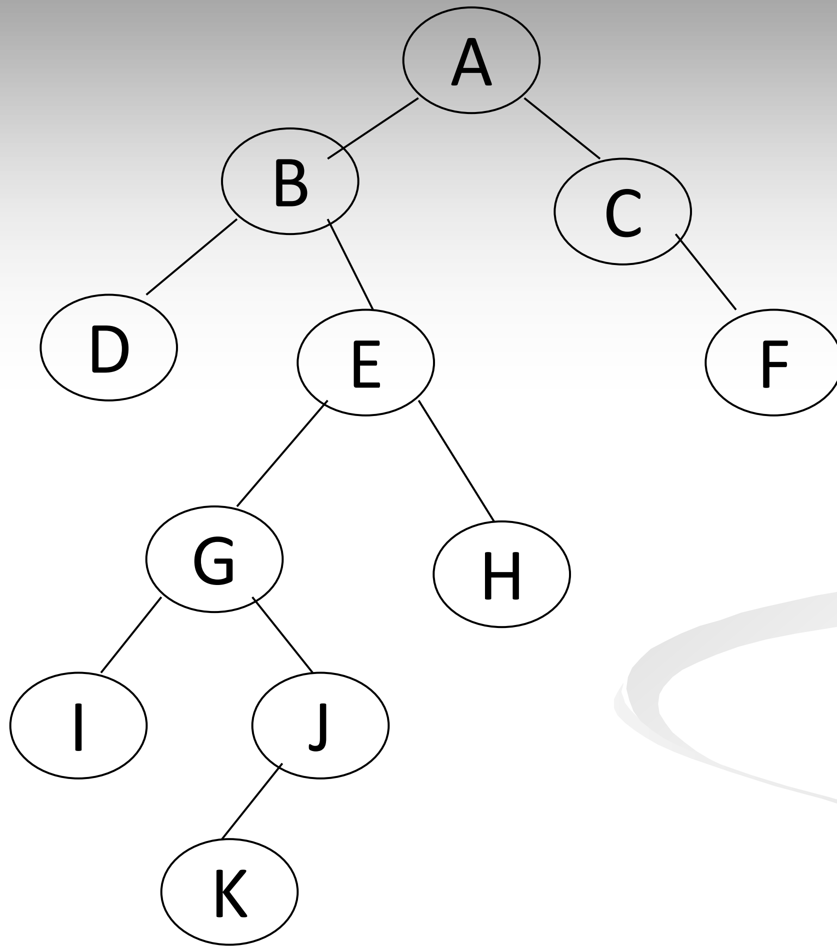
1. Se recorre en entreorden el subárbol izquierdo $\mathbf{T_{Izq}}$.
2. Se visita la raíz del árbol
3. Se recorre en entreorden el subárbol derecho $\mathbf{T_{Der}}$.

Árbol Binario: Recorrido en entreorden



B, *, B, -, C, /, D, +, E

Árbol Binario: Recorrido en entreorden



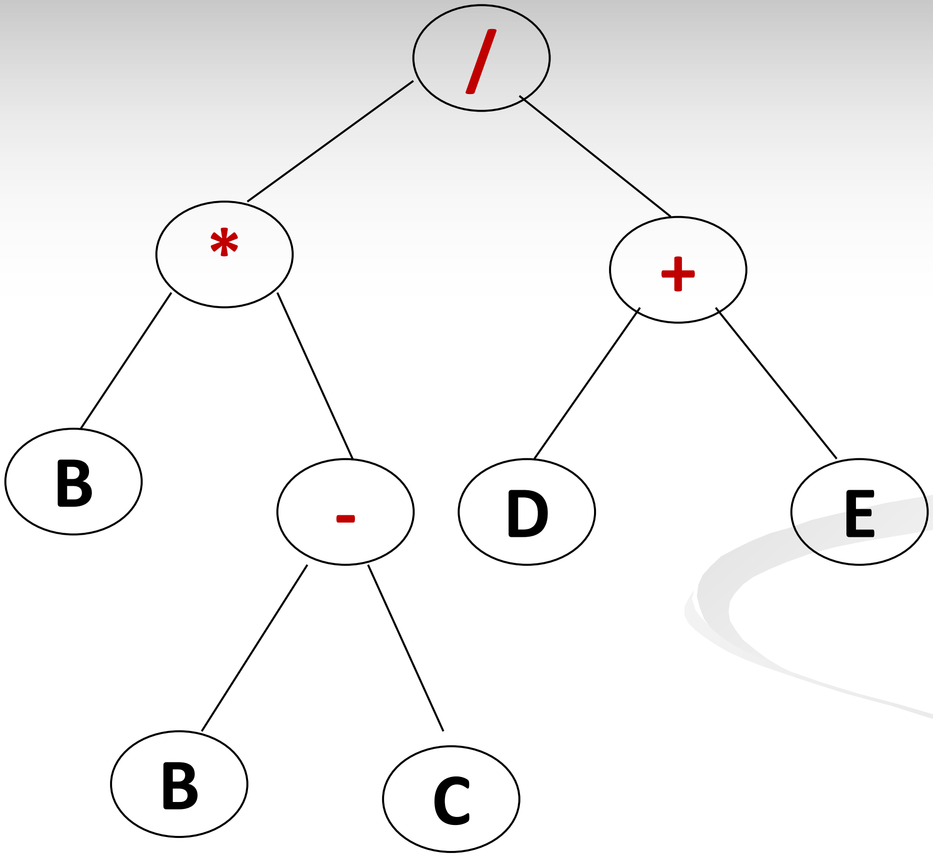
D, B, I, G, K, J, E, H, A, C, F

Árbol Binario: Recorrido en posorden

Recorrido en posorden de un árbol binario diferente del vacío $\mathbf{T = \{r,\ T_{Izq}\ ,\ T_{Der}\}}$

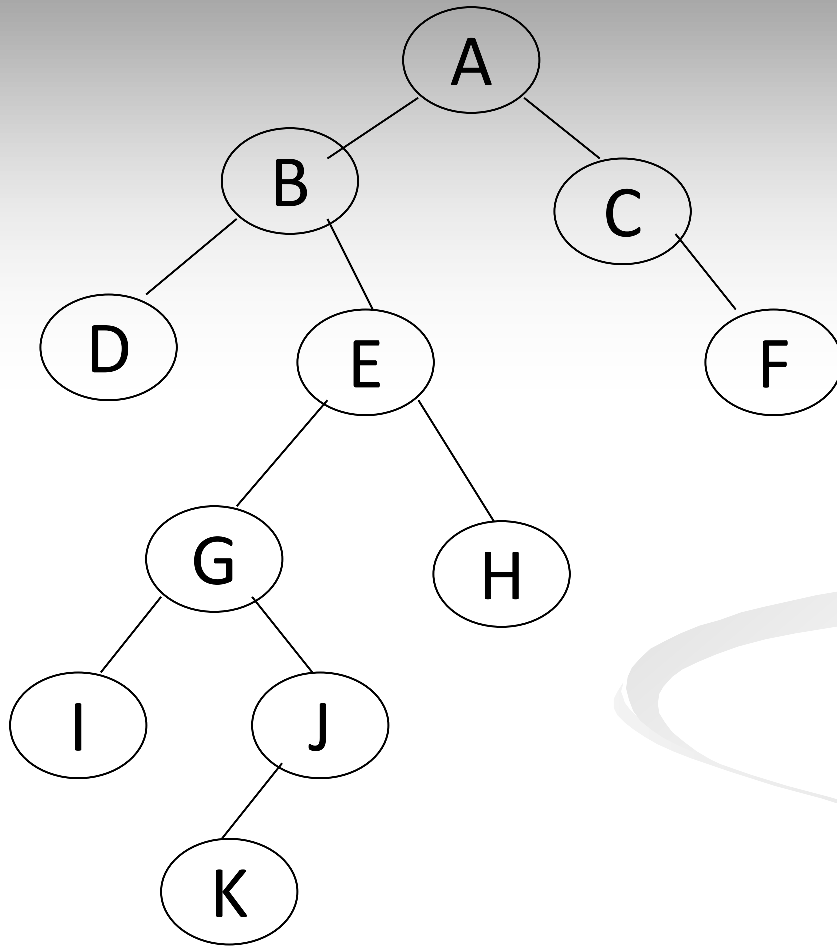
1. Se recorre en posorden el subárbol izquierdo $\mathbf{T_{Izq}}$.
2. Se recorre en posorden el subárbol derecho $\mathbf{T_{Der}}$.
3. Se visita la raíz del árbol

Árbol Binario: Recorrido en posorden



B, B, C, -, *, D, E, +, /

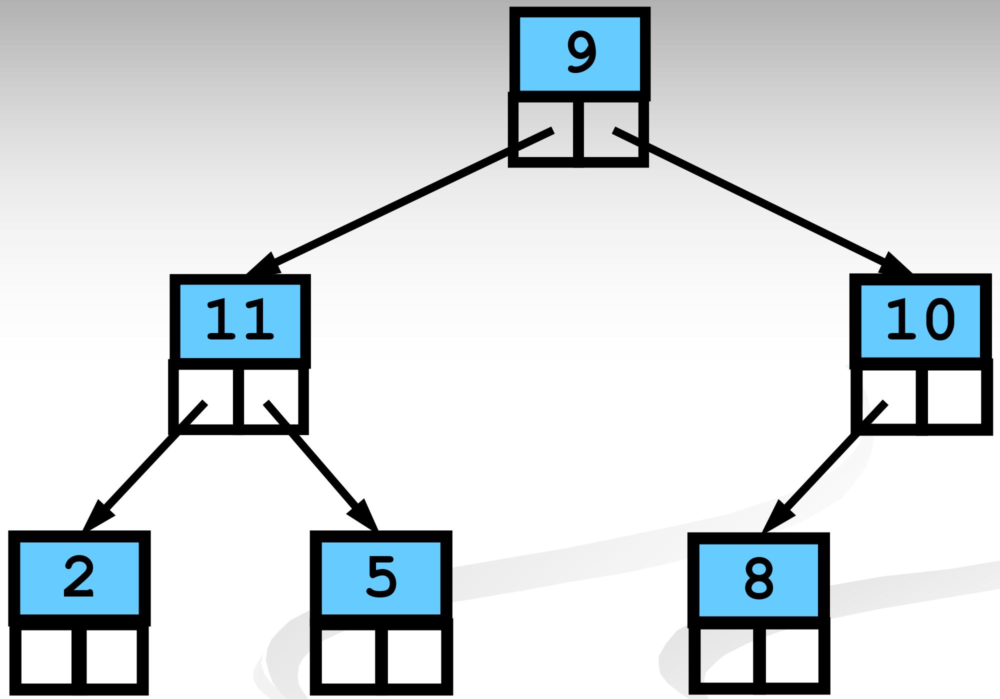
Árbol Binario: Recorrido en posorden



D, I, K, J, G, H, E, B, F, C, A

Implementación de Árbol Binario en C/C++

```
struct NodoAB {
  int dato;
  NodoAB *izq;
  NodoAB *der;
  NodoAB() :
   dato(0), izq(NULL),
            der(NULL)
  {}
  NodoAB(int d) :
   dato(d), izq(NULL),
            der(NULL)
  {}
};
```



Operaciones esHoja y esVacio

```
bool esVacio(NodoAB* A){
  return (A == NULL);
}
```

```
bool esHoja(NodoAB* A){
  return (!esVacio(A))&&(A->izq == NULL)
                     &&(A->der == NULL);
}
```

Implementación Recorrido Preorden

```
void preorden(NodoAB* A){
  if (!esVacio(A)){
    visitar(A->dato);
    preorden(A->izq);
    preorden(A->der);
  }
}
```

En el método **`visitar`** se implementaría la acción que se realiza sobre el dato del nodo visitado.

Recorridos EntreOrden y PosOrden

```
void entreorden(NodoAB* A){
  if (!esVacio(A)){
    entreorden(A->izq);
    visitar(A->dato);
    entreorden(A->der);
  }
}
```

```
void posorden(NodoAB* A){
  if (!esVacio(A)){
    posorden(A->izq);
    posorden(A->der);
    visitar(A->dato);
  }
}
```

Altura de un árbol binario

$$altura: \text{ArbolBinario} \rightarrow \mathbb{N}$$

$$altura(\,[\,]\,) = 0$$

$$altura([r, T_{izq}, T_{der}]) = 1 + \max(altura(T_{izq}), altura(T_{der}))$$

```
int altura(NodoAB* A){
  if (esVacio(A))
    return 0;
  return 1 + max(altura(A->izq), altura(A->der));
}
```

Cantidad de nodos de árbol binario

$$cantNodos: \text{ArbolBinario} \rightarrow \mathbb{N}$$

$$cantNodos(\,[\,]\,) = 0$$

$$cantNodos([r, T_{izq}, T_{der}]) =$$
$$1 + cantNodos(T_{izq}) + cantNodos(T_{der})$$

```
int cantNodos(NodoAB* A){
  if (esVacio(A))
    return 0;
  return 1 + cantNodos(A->izq) + cantNodos(A->der);
}
```

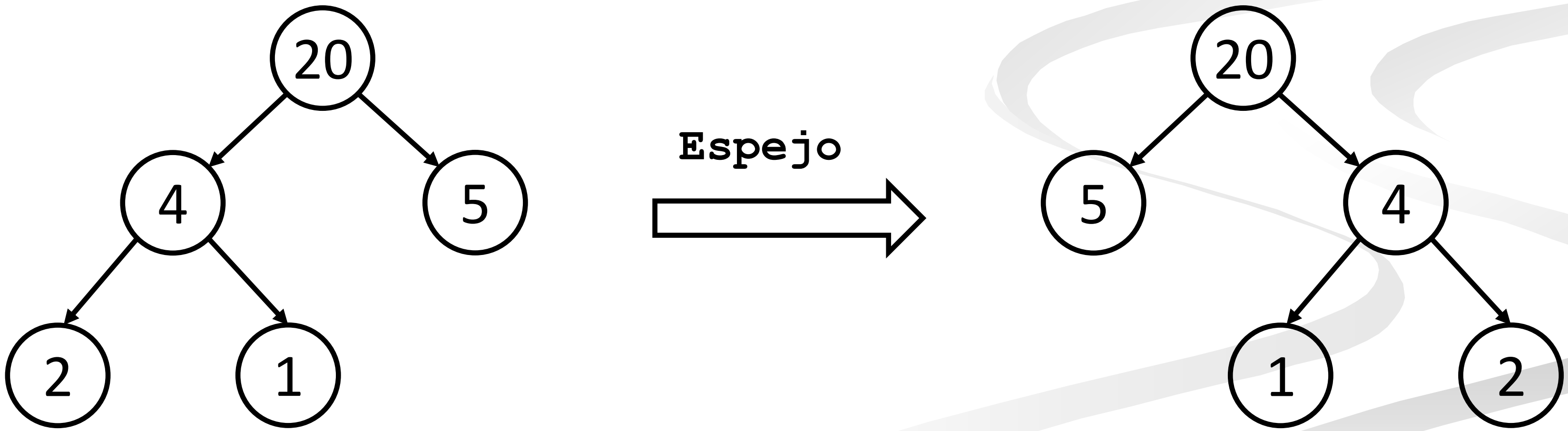
Árbol Binario: Obtener árbol Espejo

Dado un árbol binario A se desea construir un árbol binario, que no comparta memoria con A, y que sea "espejo" de A.

$$\boldsymbol{espejo}: \text{ArbolBinario} \rightarrow \text{ArbolBinario}$$

$$\boldsymbol{espejo}(\,[\,]\,) = [\,]$$

$$\boldsymbol{espejo}([r, T_{izq}, T_{der}]) = [r, \boldsymbol{espejo}(T_{der}), \boldsymbol{espejo}(T_{izq})]$$



Árbol Binario: Obtener árbol Espejo

$$espejo : \text{ArbolBinario} \rightarrow \text{ArbolBinario}$$

$$espejo(\,[\,]\,) = [\,]$$

$$espejo\left([r, T_{izq}, T_{der}]\right) = [r, espejo(T_{der}), espejo\left(T_{izq}\right)]$$

```
NodoAB* espejo(NodoAB* A){
  if (esVacio(A))
    return NULL;
  NodoAB* result = new NodoAB(A->dato);
  result->izq = espejo(A->der);
  result->der = espejo(A->izq);
  return result;
}
```

Árbol Binario: Obtener Hojas

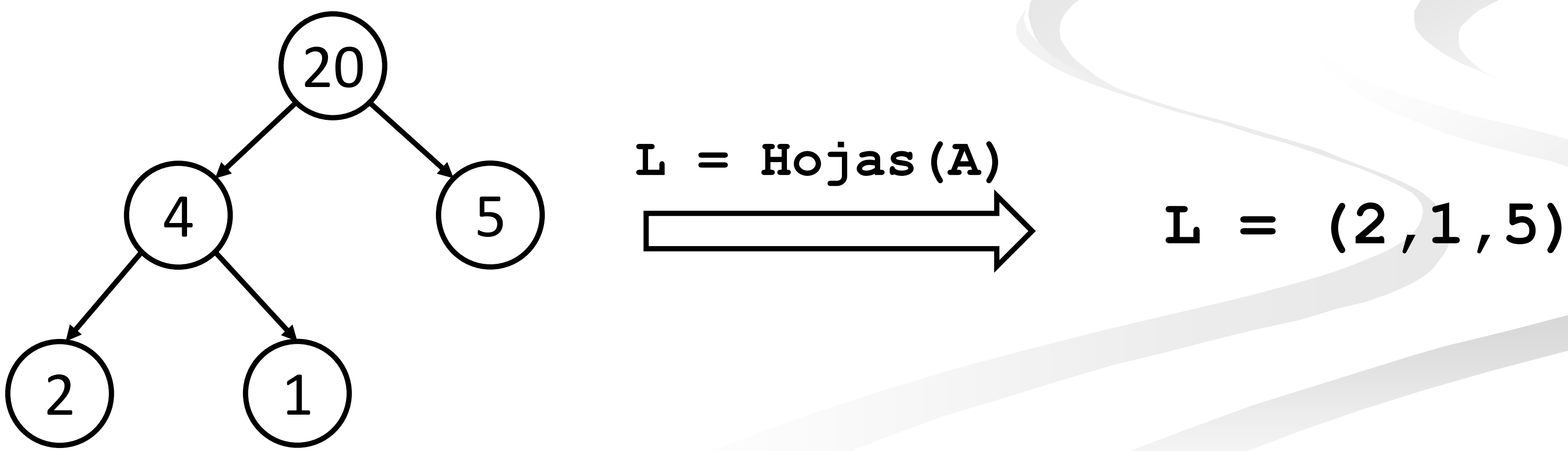
Dado un árbol binario A se desea obtener una lista con los datos de las hojas del árbol.

$$\boldsymbol{hojas}: \text{ArbolBinario} \rightarrow \text{Lista}$$

$$\boldsymbol{hojas}(\,[\,]\,) = [\,]$$

$$\boldsymbol{hojas}(\,[\boldsymbol{r}, [\,], [\,]]\,) = [\boldsymbol{r}]$$

$$\boldsymbol{hojas}([\boldsymbol{r}, \boldsymbol{T_{izq}}, \boldsymbol{T_{der}}]) = \boldsymbol{hojas}(\boldsymbol{T_{izq}}).\boldsymbol{hojas}(\boldsymbol{T_{der}})$$



Árbol Binario: Obtener Hojas

$$hojas: \text{ArbolBinario} \rightarrow \text{Lista}$$

$$hojas(\,[\,]\,) = [\,] \qquad hojas(\,[r, [\,], [\,]]\,) = [r]$$

$$hojas([r, T_{izq}, T_{der}]) = hojas(T_{izq}).hojas(T_{der})$$

```
NodoLista* hojas(NodoAB* A){
  if (esVacio(A))
    return NULL;
  NodoLista* result = NULL;
  if (esHoja(A))
    result = new NodoLista(A->dato);
  else
    result = concatListas(hojas(A->izq), hojas(A->der));
  return result;
}
```

Árbol Binario: Obtener Nodos Nivel k

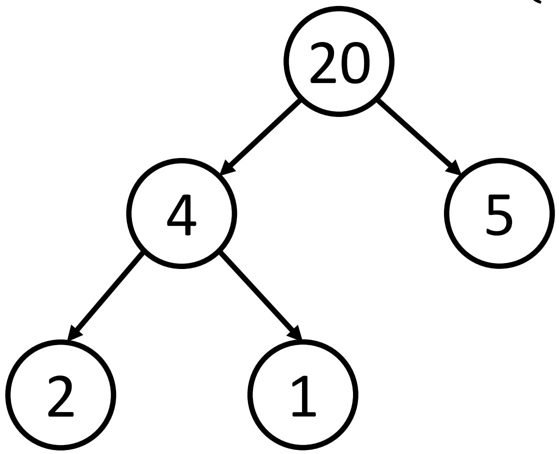
Dado un árbol binario A y un valor entero k > 0, se desea obtener una lista con los elementos que están en el nivel k del árbol.

$$nodosNivelK: \text{ArbolBinario} \times \mathbb{N} \rightarrow \text{Lista}$$

$$nodosNivelK([\,], k) = [\,]$$

$$nodosNivelK([r, T_{izq}, T_{der}], 1) = [r]$$

$$nodosNivelK([r, T_{izq}, T_{der}], k) = nodosNivelK(T_{izq}, k-1).nodosNivelK(T_{der}, k-1)$$



`L = nodosNivelK(A,2)`

`L = (4,5)`

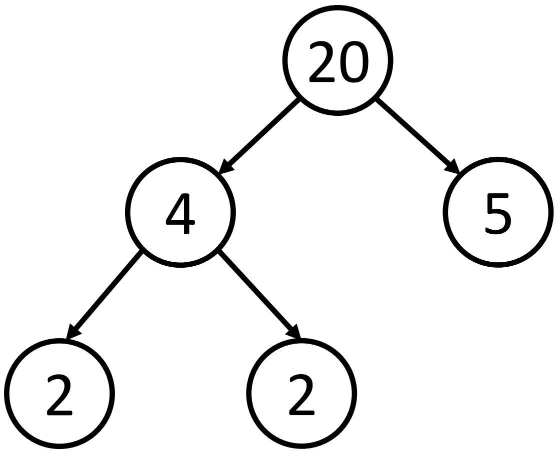
Árbol Binario: Obtener Nodos Nivel k

```
NodoLista* nodosNivelK(NodoAB* A, int k){
  if (esVacio(A))
    return NULL;
  NodoLista* result = NULL;
  if (k == 1)
    result = new NodoLista(A->dato);
  else
    result = concatListas(nodosNivelK(A->izq, k-1),
                          nodosNivelK(A->der, k-1));
  return result;
}
```

Ejercicio 1

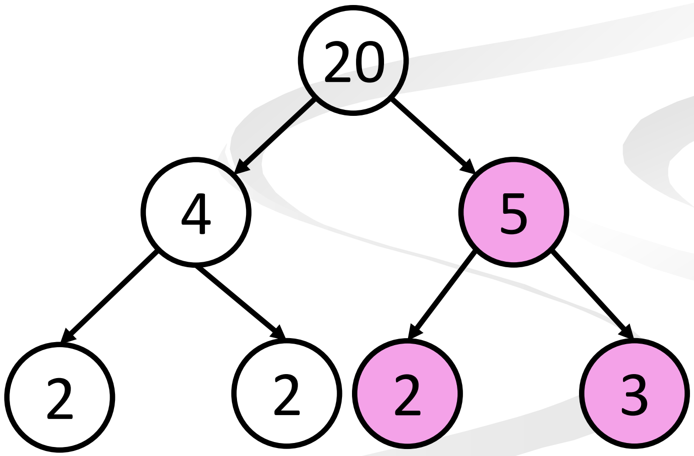
Dado un árbol binario de enteros, determinar si "está factorizado". Esto es que cada nodo es igual a la multiplicación de las raíces de sus subárboles hijos. Si el árbol es hoja se considera que está factorizado; y si tiene sólo un hijo vacío entonces la raíz debe ser igual a la raíz de su hijo no vacío.

Caso 1:



Está factorizado

Caso 2:



Retorna **falso**, porque $2 * 3 \neq 5$

Ejercicio 1: Solución

Caso base 1: Si el árbol es vacío retornar `false`

Caso base 2: Si el árbol es hoja retornar `true`

Casos inductivos:

1,2 - Si el subárbol izquierdo (derecho) es vacío, retornar `true` en caso que la raíz sea igual a la raíz del subárbol derecho (izquierdo), y el subárbol derecho (izquierdo) esté factorizado. Retornar `false` en caso contrario.

3 - Si ambos subárboles no son vacíos, retornar `true` si se cumplen las siguientes condiciones

- `raiz = izq.raíz * der.raíz`
- Están factorizados el subárbol izquierdo y el subárbol derecho

Ejercicio 1: Solución

```
bool estaFactorizado(NodoAB* A){
  if (esVacio(A))          //Caso base 1
      return false;
  if (esHoja(A))           //Caso base 2
      return false;
  if (esVacio(A->izq))  //Caso inductivo 1
      return (A->dato == A->der->dato)
          && estaFactorizado(A->der);
  if (esVacio(A->der))  //Caso inductivo 2
      return (A->dato == A->izq->dato)
          && estaFactorizado(A->izq);
  return                    //Caso inductivo 3
       (A->dato == A->izq->dato * A->der->dato)
    && estaFactorizado(A->izq)
    && estaFactorizado(A->der);
}
```

Ejercicio 2

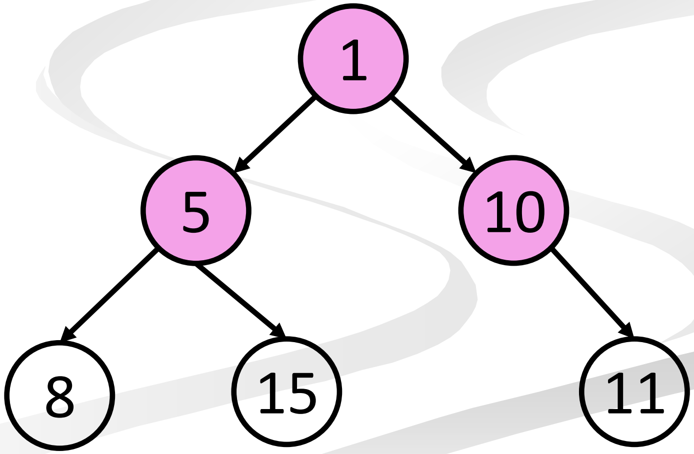
Dado un árbol binario de enteros, y un entero N, retorna la suma de todos los elementos del árbol que están antes del nivel N (se incluye el propio nivel N).

Si el árbol es vacío debe retornar 0.

La raíz del árbol se encuentra en el nivel 1.

Asumir que N nunca será mayor a la altura del árbol.

Ejemplo: En el árbol de la figura la suma para N = 2, es 1 + 5 + 10 = 16.



Ejercicio 2: Solución

Caso base 1: Si el árbol es vacío retornar **0**

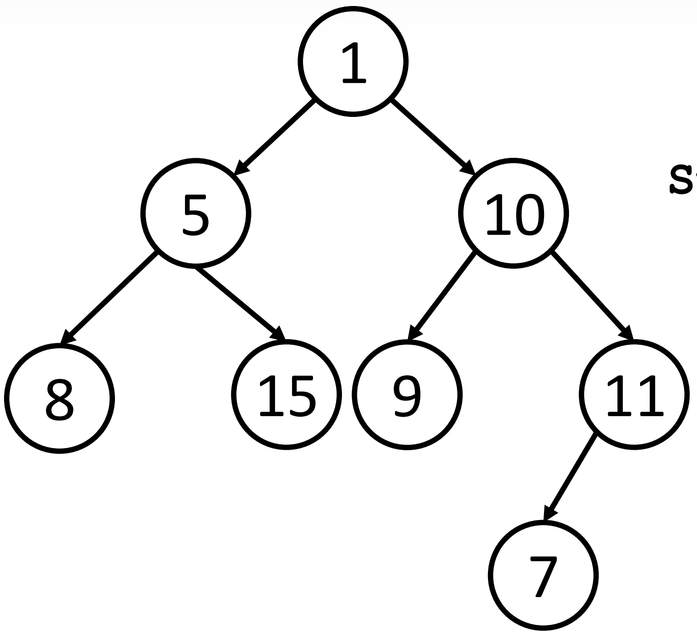
Caso base 2: Si N = 1 retornar el valor de la raíz del árbol

Caso inductivo: Retornar **`raíz + SumaHastaNivel(N-1, izq)`**
`+ SumaHastaNivel(N-1, der)`

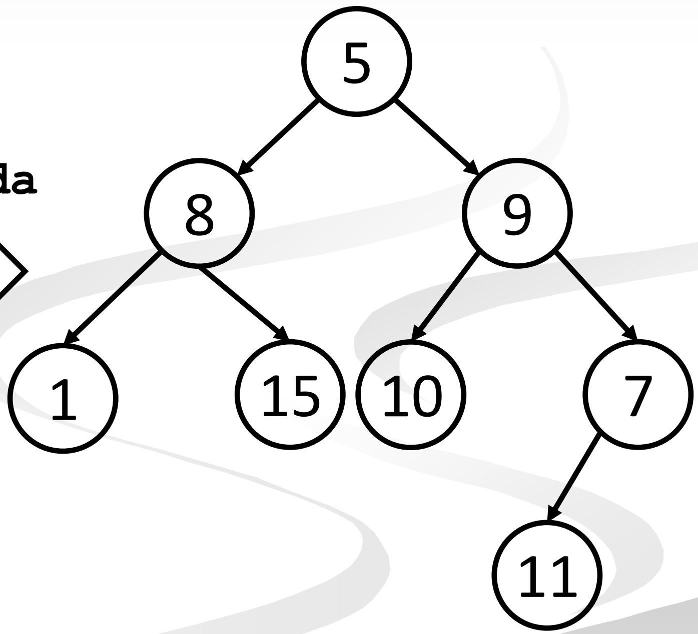
```
int sumaHastaNivel(NodoAB* A, int N){
  if (esVacio(A))          //Caso base 1
      return 0;
  if (N == 1)              //Caso base 2
      return A->dato;
  return                    //Caso inductivo
       A->dato + sumaHastaNivel(A->izq, N-1)
               + sumaHastaNivel(A->der, N-1);
}
```

Ejercicio 3

Implemente un método **`SumergirIzquierda`**, el cual comenzando desde el nivel 0 del árbol, intercambia para cada subárbol su raíz con la raíz de su subárbol izquierdo.



`SumergirIzquierda`



Intercambios

1. 1 con 5
2. 1 con 8
3. 10 con 9
4. 11 con 7

Ejercicio 3: Solución

Caso base: Si el árbol es vacío o es hoja, no hacer nada.

Caso inductivo: Si subárbol izquierdo no es vacio, intercambiar valores de raíz con raíz de subárbol izquierdo.

Luego **SumergirIzquierda(izq)** y **SumergirIzquierda(der)**

```
void SumergirIzquierda(NodoAB*& A){
   if (esVacio(A)||esHoja(A))          //Caso base
       return;
   if (!esVacio(A->izq)){              //Caso inductivo
       int tmp = A->dato;
       A->dato = A->izq->dato;
       A->izq->dato = tmp;
       SumergirIzquierda(A->izq);
   }
   SumergirIzquierda(A->der);
}
```

¿Cómo representar árboles generales?

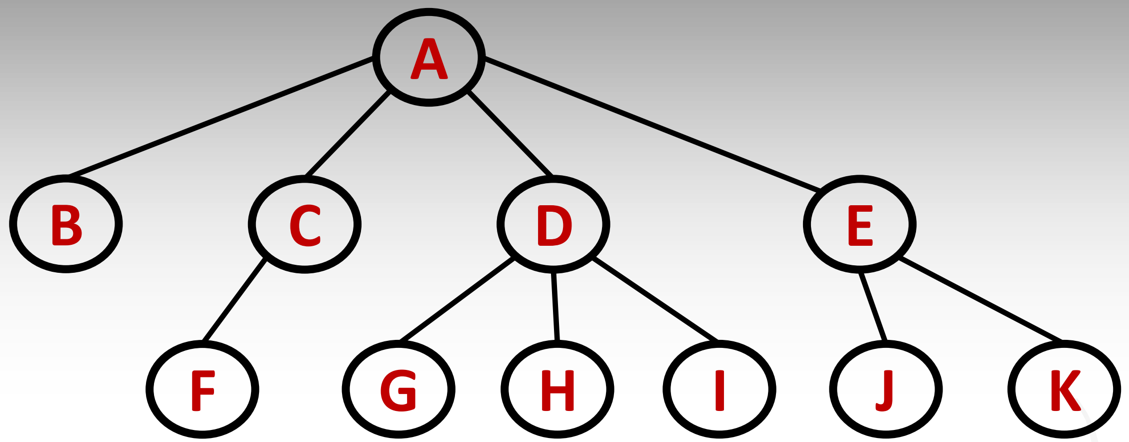
En un **árbol general** el número de hijos por nodo puede variar. Una idea de representación es que cada nodo tiene una “lista” de árboles asociados (sus subárboles).

Representación: Cada nodo se relaciona con su “primer hijo (pH)” y con su “siguiente hermano (sH)”, conformando una estructura de árbol binario.

Esto establece una **equivalencia entre árboles generales y árboles binarios**: todo árbol general puede representarse como uno binario y, todo árbol binario corresponde a un determinado árbol general.

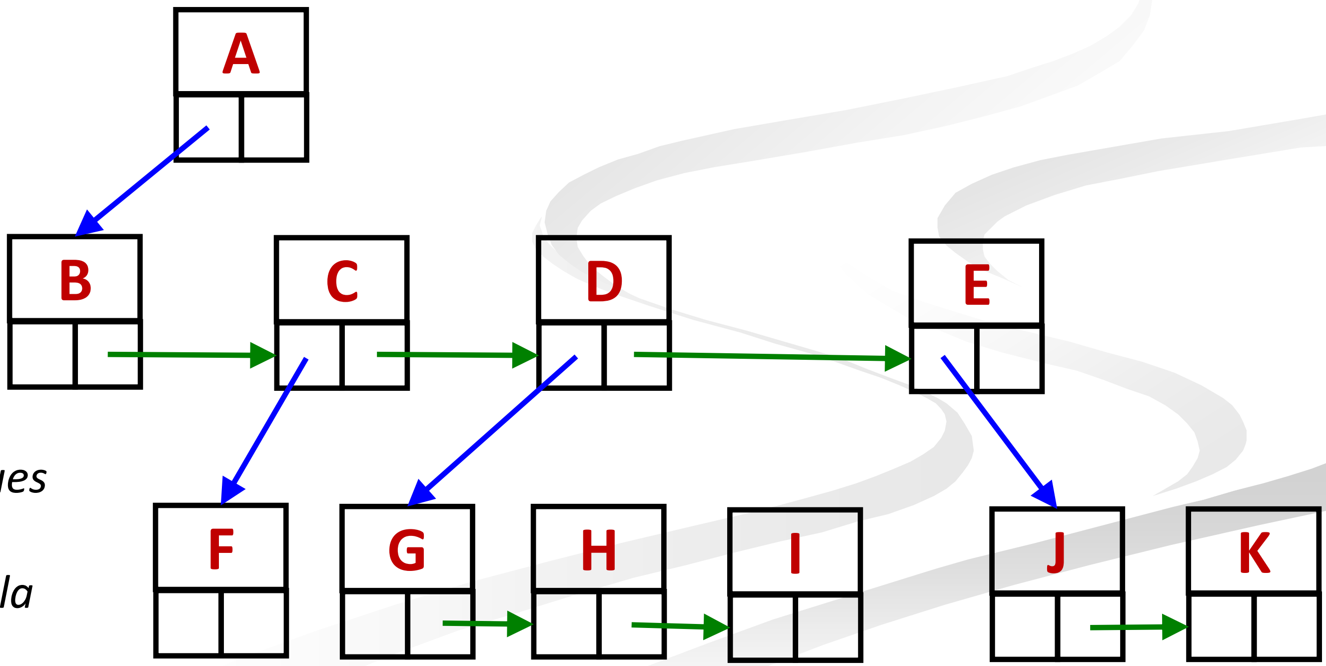
Árbol general – Árbol Binario

Cada nodo tiene un núnero *finito* de subárboles



Representación como árbol binario:

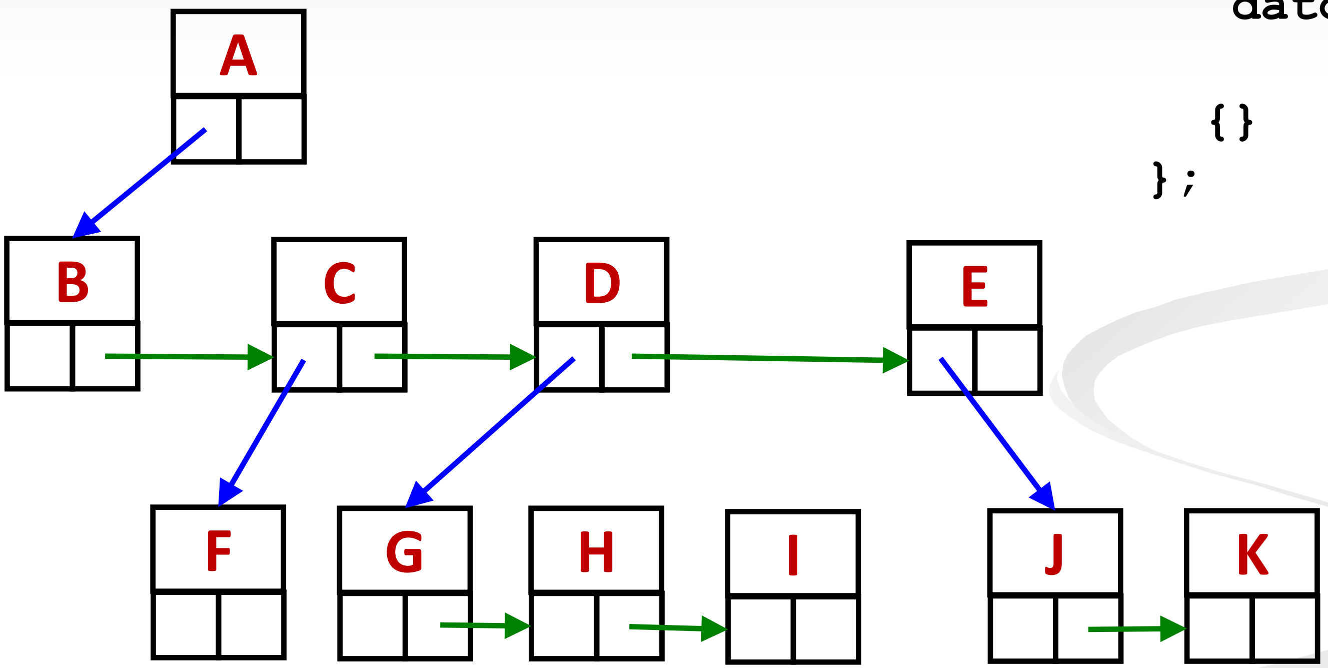
- Primer hijo (pH)
- Siguiente hermano (sH)



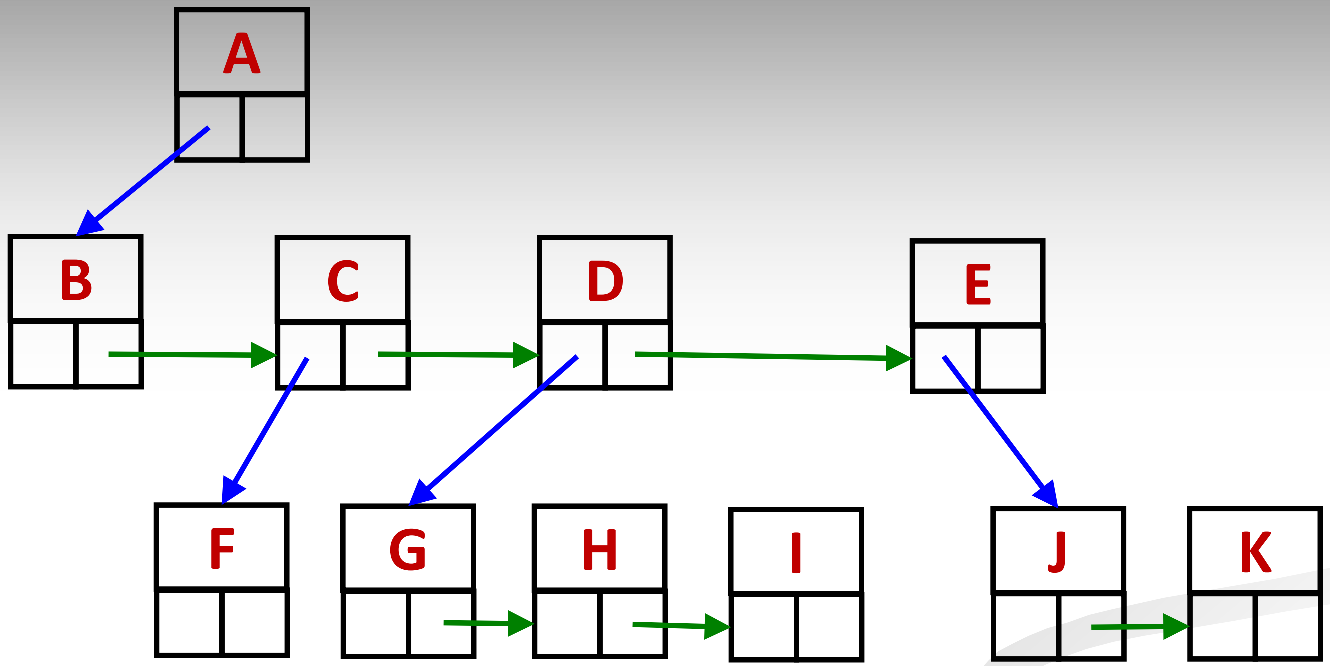
Notar que estamos representando tanto bosques de árboles como árboles (bosques de un sólo árbol; la raíz)

Árbol general – Árbol Binario

```
struct NodoAG {
   int dato;
   NodoAG* pH;
   NodoAG* sH;
   NodoAG(int d) :
    dato(d), pH(NULL),
                 sH(NULL)
   {}
};
```



¿Cuándo un nodo es hoja?




```
bool esHoja(NodoAG* A){
  return (A->pH == NULL);
}
```

Ejemplo 1: Cantidad de nodos de árbol general

```
int cantNodos(NodoAG* A){
  if (esVacio(A))
    return 0;
  return 1 + cantNodos(A->pH) + cantNodos(A->sH);
}
```

Notar que:

- el código es idéntico al de contar nodos en un árbol binario tradicional.
- Si se invoca a **`cantNodos`** con un bosque **`T`** (**`T->sH != NULL`**), se tiene la cantidad de nodos del bosque.

Ejemplo 2: Altura de árbol general

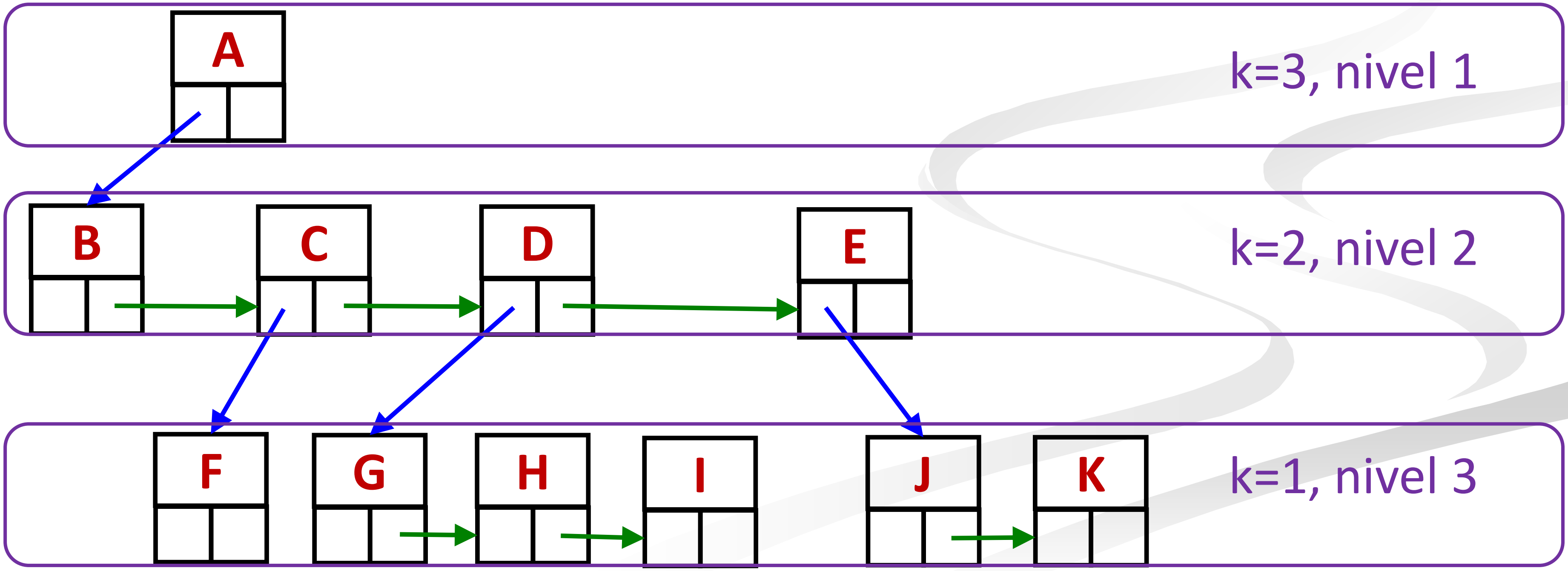
```
int altura(NodoAG* A){
  if (esVacio(A))
    return 0;

  int maxAlturaHijo = 0;
  NodoAG* subArbol = A->pH;
  while (subArbol != NULL){
      maxAlturaHijo = max(maxAlturaHijo, altura(subArbol));
      subArbol = subArbol->sH;
  }
  return 1 + maxAlturaHijo;
}
```

Notar que:

- “primer hijo” (**pH**) aumenta la altura.
- “siguiente hermano” (**sH**) no aumenta la altura, la mantiene.

Ejemplo 3: Imprime elementos Nivel K

```
int impNivel(NodoAG* A, int k){
  if (!esVacio(A) && k > 0){
    if (k == 1) cout << A->dato;
    else impNivel(A->pH, k-1);
    impNivel(A->sH, k);
  }
}
```



Ejemplo 4: Buscar un elemento

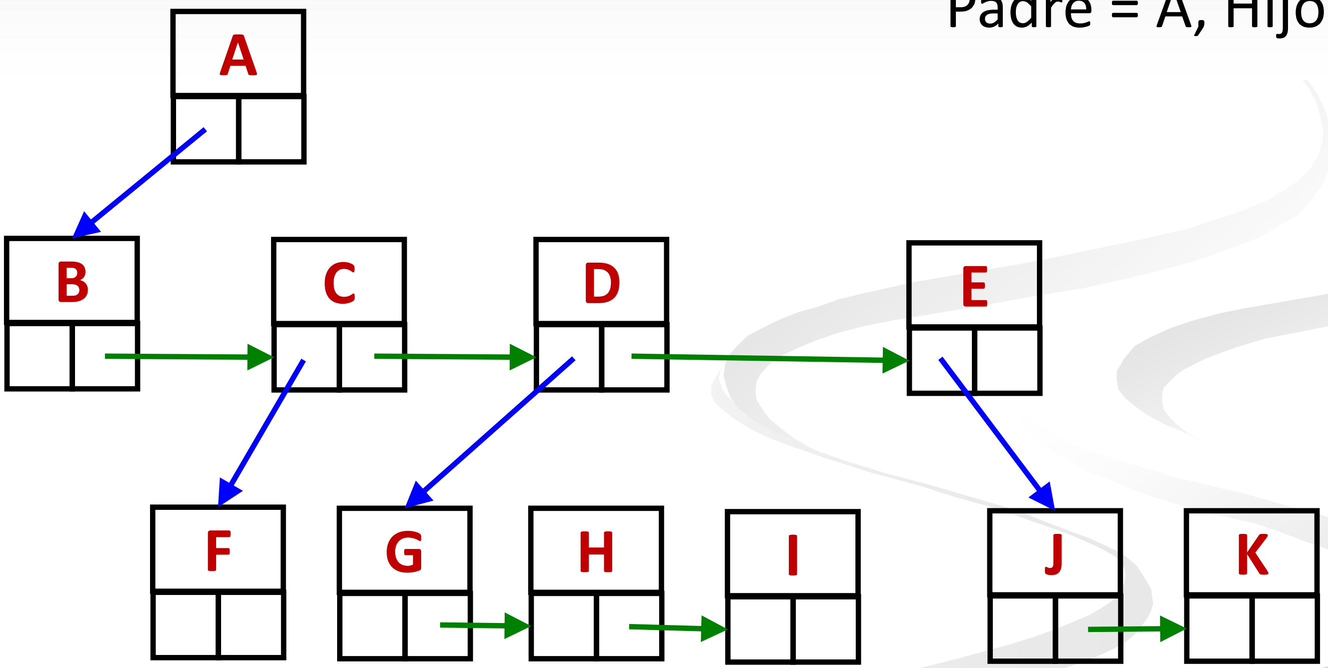
Dados un árbol general (pH-sH) y un elemento, retorna un puntero al nodo del árbol que tiene dicho elemento, o NULL si no está. Asumimos que el árbol no tiene repetidos

```
NodoAG* buscar(NodoAG* A, int x){
  if (esVacio(A))
    return NULL;
  if (A->dato == x)
    return A;
  NodoAG* esta_sH = buscar(A->sH, x);
  if (esta_sH != NULL)
      return esta_sH;
  else
      return buscar(A->pH, x);
}
```

Ejemplo 5: Insertar un elemento

Dados un árbol general (pH-sH) y dos elementos “padre” e “hijo”, agregar a “hijo” como primer hijo de “padre” en el árbol. Esto se haría si “padre” está e “hijo” no está. Asumimos que el árbol no tiene repetidos

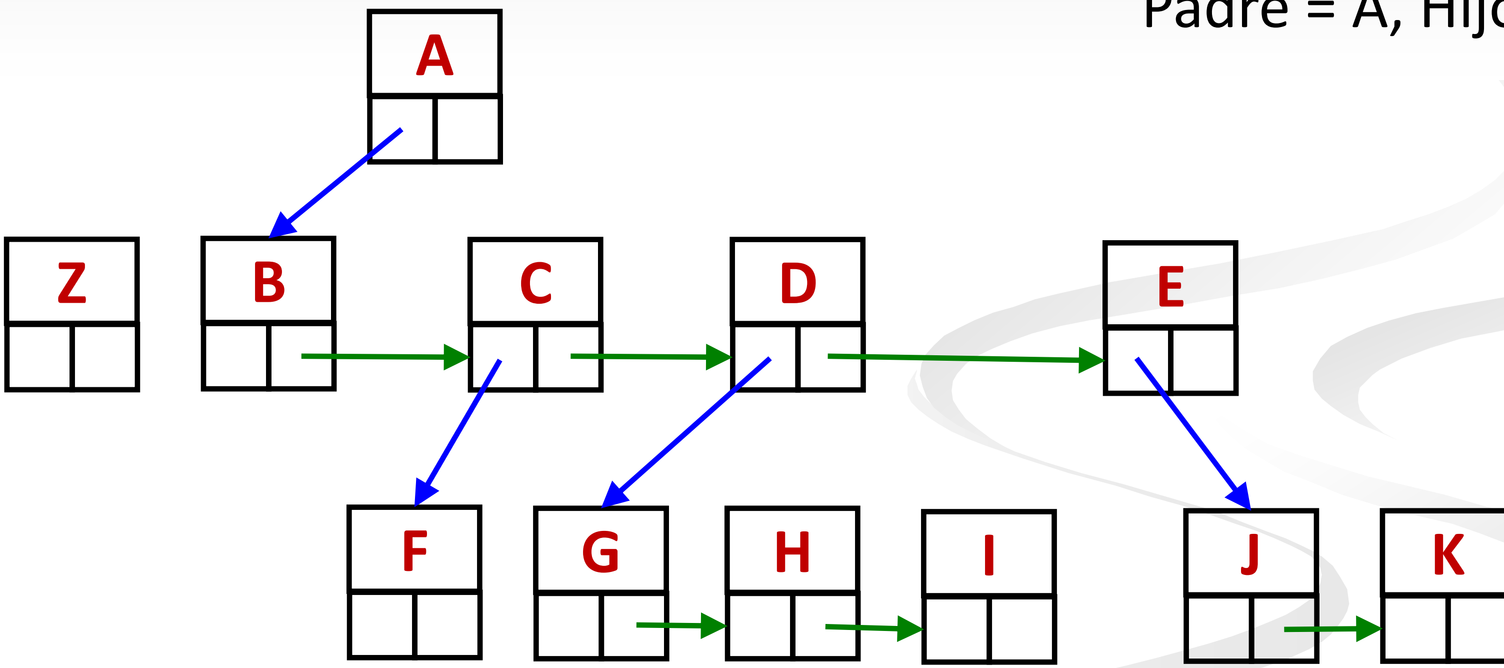
Padre = A, Hijo = Z



Ejemplo 5: Insertar un elemento

Dados un árbol general (pH-sH) y dos elementos “padre” e “hijo”, agregar a “hijo” como primer hijo de “padre” en el árbol. Esto se haría si “padre” está e “hijo” no está. Asumimos que el árbol no tiene repetidos

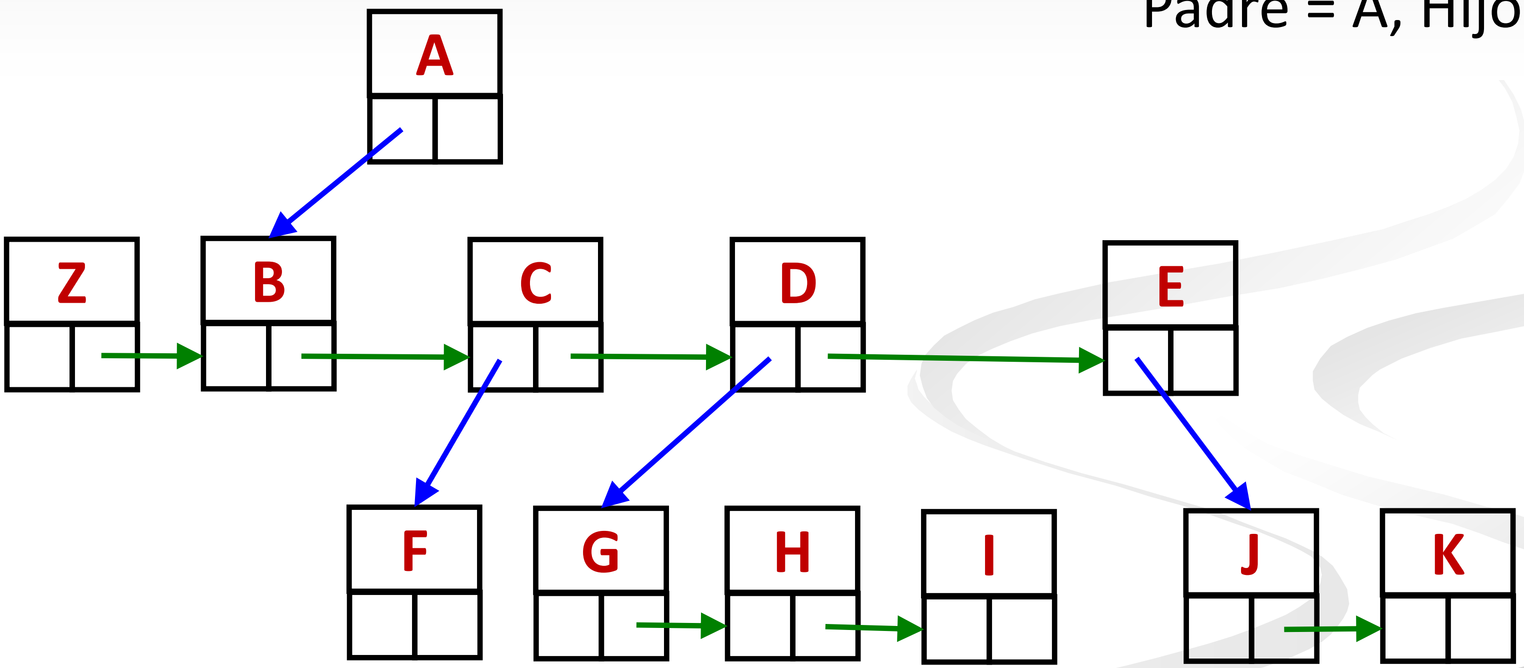
Padre = A, Hijo = Z



Ejemplo 5: Insertar un elemento

Dados un árbol general (pH-sH) y dos elementos "padre" e "hijo", agregar a "hijo" como primer hijo de "padre" en el árbol. Esto se haría si "padre" está e "hijo" no está. Asumimos que el árbol no tiene repetidos

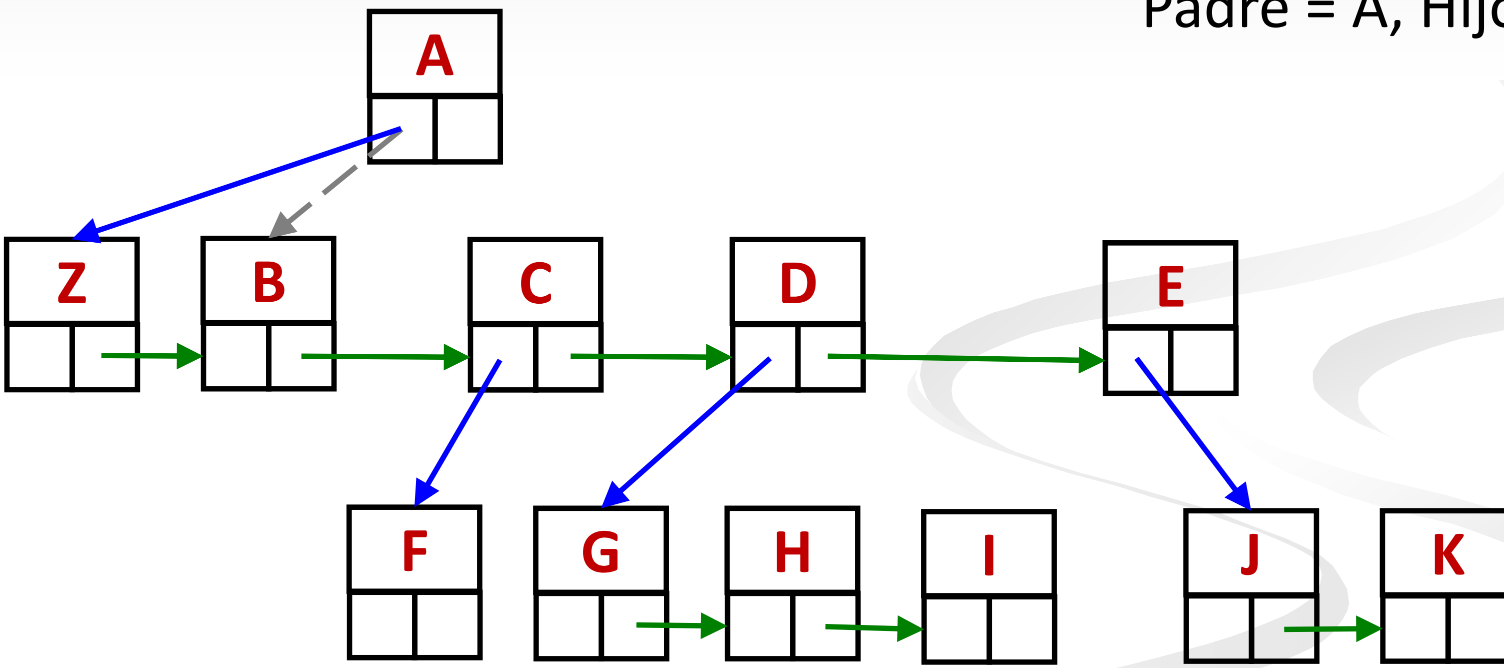
Padre = A, Hijo = Z



Ejemplo 5: Insertar un elemento

Dados un árbol general (pH-sH) y dos elementos “padre” e “hijo”, agregar a “hijo” como primer hijo de “padre” en el árbol. Esto se haría si “padre” está e “hijo” no está. Asumimos que el árbol no tiene repetidos

Padre = A, Hijo = Z



Ejemplo 5: Insertar un elemento

Dados un árbol general (pH-sH) y dos elementos “padre” e “hijo”, agregar a “hijo” como primer hijo de “padre” en el árbol. Esto se haría si “padre” está e “hijo” no está. Asumimos que el árbol no tiene repetidos

```
void insertar(NodoAG* A, int padre, int hijo){
  if (buscar(A, hijo) == NULL){
    NodoAG* nodo_padre = buscar(A, padre);
    if (nodo_padre != NULL){
      NodoAG* nodo_nuevo = new NodoAG(hijo);
      nodo_nuevo->sH = nodo_padre->pH;
      nodo_padre->pH = nodo_nuevo;
    }
  }
}
```

Ejemplo 6: Copia Parcial

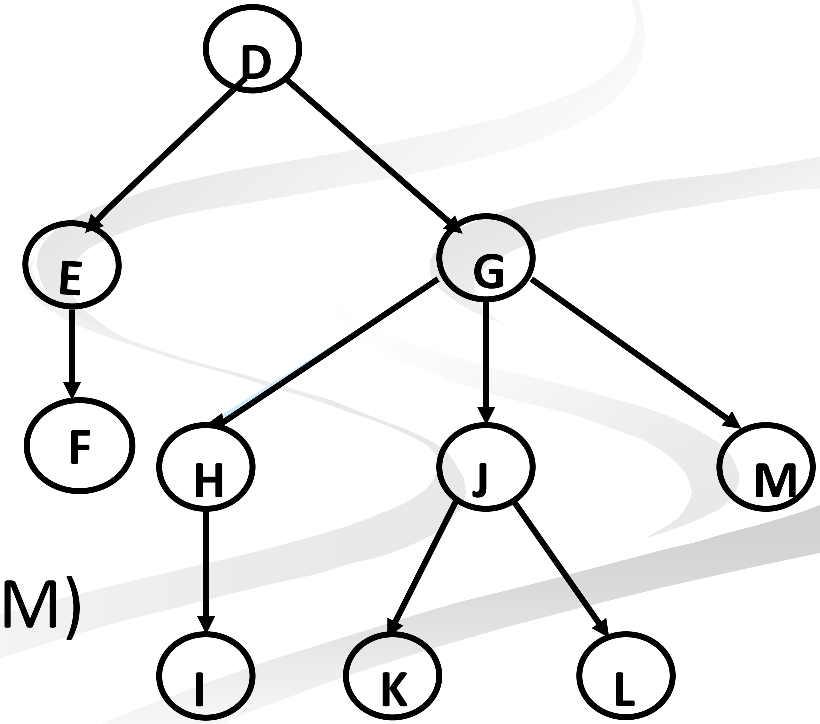
Dados un árbol general (pH-sH) y un entero no negativo k, retorna una copia del árbol, sin compartir memoria, hasta el nivel k inclusive.

```
NodoAG* copiaParcial(NodoAG* A, int k){
  if (k == 0 || esVacio(A))
      return NULL;
  NodoAG* nuevo = new NodoAG(A->dato);
  nuevo->pH = copiaParcial(A->pH, k-1);
  nuevo->sH = copiaParcial(A->sH, k);
  return nuevo;
}
```

Ejemplo 7: Recorrido en preorden

El recorrido en **preorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Visitando la raíz del árbol
2. Recorriendo en **preorden** cada uno de los subárboles de la raíz.



preorden : (D, E, F, G, H, I, J, K, L, M)

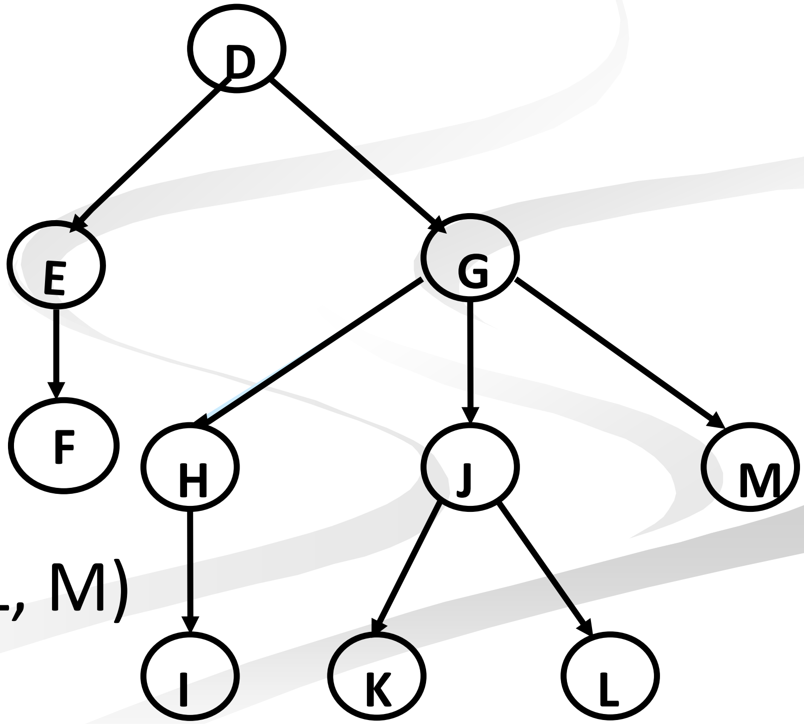
Ejemplo 7: Recorrido en preorden

```
void preorden(NodoAG* A){
  if (!esVacio(A)){
    visitar(A->dato);                   //Visitar raiz
    NodoAG* subArbol = A->pH;
    while (subArbol != NULL){           //Preorden de cada hijo
      preorden(subArbol);
      subArbol = subArbol->sH;
    }
}
```

Recorrido en entreorden

El recorrido en **entreorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Recorriendo en **entreorden** uno de los subárboles de la raíz
2. Visitando la raíz del árbol
3. Recorriendo en **entreorden** cada uno de los restantes subárboles de la raíz.



entreorden : (F, E, D, I, H, G, K, J, L, M)

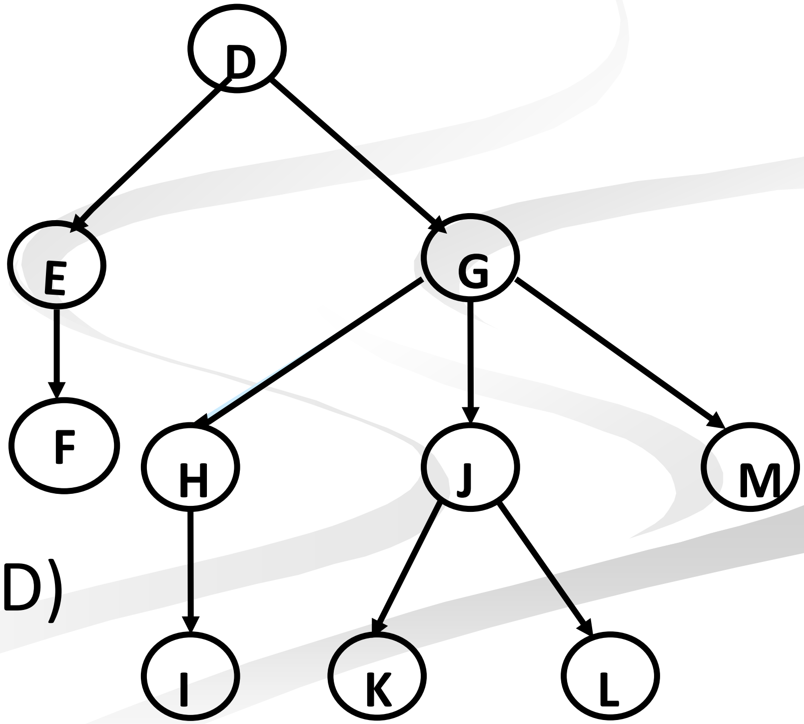
Ejemplo 8: Recorrido en entreorden

```
void entreorden(NodoAG* A){
  if (!esVacio(A)){
    NodoAG* subArbol = A->pH;
    entreorden(subArbol);        //Entreorden primer hijo
    visitar(A->dato);            //Visitar raiz
    while (subArbol != NULL){    //Entreorden resto de
                                 //los hijos
      subArbol = subArbol->sH;
      entreorden(subArbol);
    }
}
```

Recorrido en posorden

El recorrido en **posorden** de un árbol (con más de un nodo) está formada por la lista de nodos que se obtiene de la siguiente forma:

1. Recorriendo en **posorden** cada uno de los subárboles de la raíz
2. Visitando la raíz del árbol



posorden : (F E, I, H, K, L, J, M, G, D)

Ejemplo 9: Recorrido en posorden

```
void posorden(NodoAG* A){
  if (!esVacio(A)){
    NodoAG* subArbol = A->pH;
    while (subArbol != NULL){  //Posorden de cada hijo
      posorden(subArbol);
      subArbol = subArbol->sH;
    }
    visitar(A->dato);          //Visitar raiz
}
```