

# ÁRBOLES GENERALES (N-ARIOS)

Un árbol es un tipo especial de relación que es muy útil para el estudio de una gran variedad de aplicaciones en las ciencias de la computación e ingeniería. Un árbol se representa como un grafo dirigido o no dirigido. Los árboles son muy usados en el estudio y construcción de base de datos, modelamiento jerárquico de clases y en la teoría de lenguajes y construcción de compiladores, son muy usados para describir los árboles sintácticos correspondientes a gramáticas de lenguajes.

# ÁRBOLES GENERALES (N-ARIOS)

## Definición

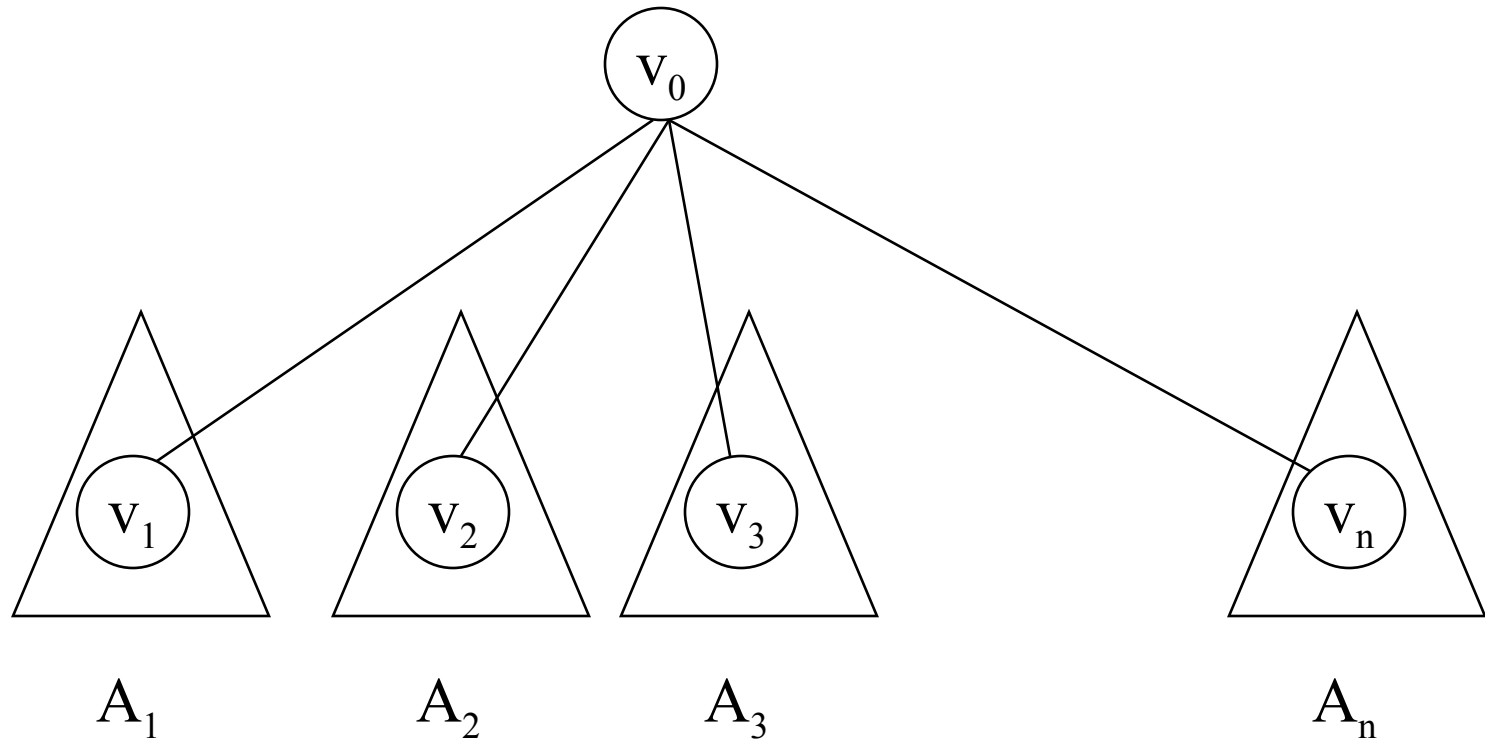
Un árbol  $A$  se define como un conjunto de elementos llamados nodos o vértices, de forma que:

- $A$  es vacío, en cuyo caso se llama árbol vacío o árbol nulo, o
- $A$  contiene un nodo distinguido  $v_0$  llamado raíz de  $A$  y los nodos restantes de  $A$  forman un conjunto de árboles  $A_1, A_2, A_3, \dots, A_n$

Cada árbol  $A_i$  tiene como raíz al nodo  $v_i$

Un árbol se representa mediante un grafo en donde la raíz  $v_0$  es el nodo en  $A$  en la parte superior. Una línea hacia abajo de izquierda a derecha, un arco señala a los hijos de  $v_0$

# ÁRBOLES GENERALES (N-ARIOS)



- $A_1, A_2, A_3, \dots, A_n$  son llamados subárboles de  $v_0$
- Si  $A_i$  no es vacío, entonces su raíz  $v_i$ , es llamado hijo de  $v_0$ , y  $v_0$  es llamado padre de  $v_i$

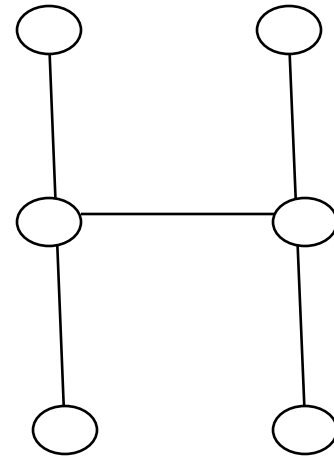
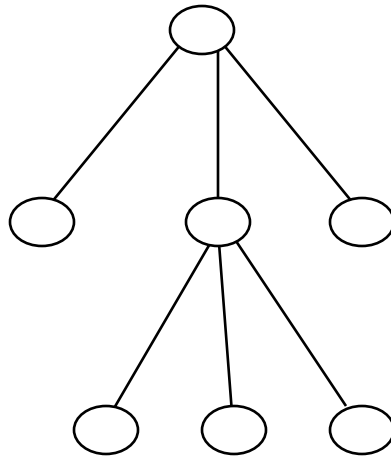
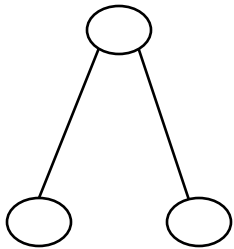
# ÁRBOLES GENERALES (N-ARIOS)

## ÁRBOLES LIBRES Y ÁRBOLES CON RAÍZ

Un árbol libre es un tipo especial de grafo no dirigido, cualquiera de sus nodos puede ser considerado una raíz.

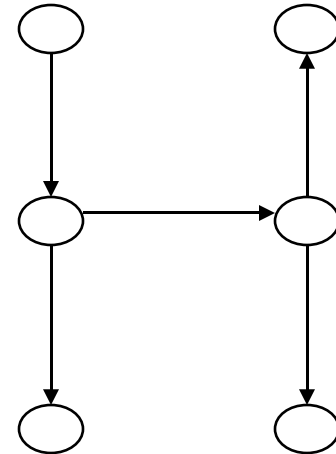
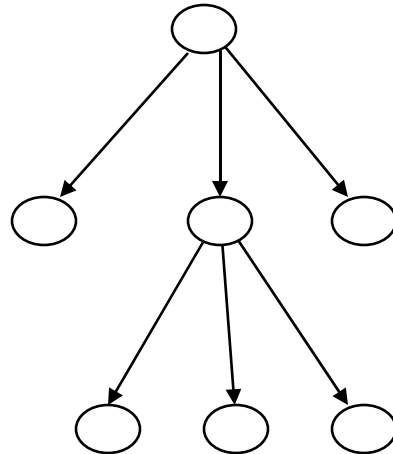
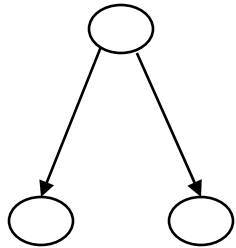
Un árbol con raíz es un tipo especial de grafo dirigido. Es un árbol con única raíz.

Ejemplos de árboles libres:



# ÁRBOLES GENERALES (N-ARIOS)

Ejemplos de árboles con raíz:



# ÁRBOLES GENERALES (N-ARIOS)

## Definición

Sea  $V$ : conjunto de vértices o nodos

$A$ : relación: aristas o caminos entre los elementos de  $V$

$(A, v_0)$  es un árbol si y solo si:

1. Existe un único vértice  $v_0 \in V$  tal que  $v_0$  no tiene ninguna entrada.  $v_0$  es llamado raíz del árbol  $A$ .
2.  $\forall v \in V$  tal que  $v \neq v_0$ ,  $v$  tiene solo una entrada.

# ÁRBOLES GENERALES (N-ARIOS)

Si  $(A, v_0)$  es un árbol

1.  $A$  es una relación irreflexiva (no existen ciclos).
2.  $A$  es asimétrica, si  $u$  es padre de  $v$ ,  $v$  no puede ser padre de  $u$ .
3. Si  $a R b$  y  $b R c$  entonces  $a \neg R c \quad \forall a, b, c \in V$

$R$  no es transitiva. Es decir si  $a$  es padre de  $b$ , y  $b$  es padre de  $c$ , no es cierto que  $a$  es padre de  $c$ .

# ÁRBOLES GENERALES (N-ARIOS)

## Definición recursiva de árbol

Un árbol  $A$  es un conjunto de nodos  $V$ , donde

Si  $V = \text{vacío}$   $A$  es el árbol nulo

Si  $V \neq \text{vacío}$

Existe  $v_0 \in V$  llamado raíz de  $A$

y existen  $A_1, A_2, \dots, A_n$ , con raíces  $v_1, v_2, \dots, v_n$  respectivamente tal que existe una arista  $(v_0, v_i) \forall i$

Cada  $A_i$  es un subárbol del árbol  $A$

1. Un vértice es por si mismo un árbol, este nodo es la raíz de dicho árbol.

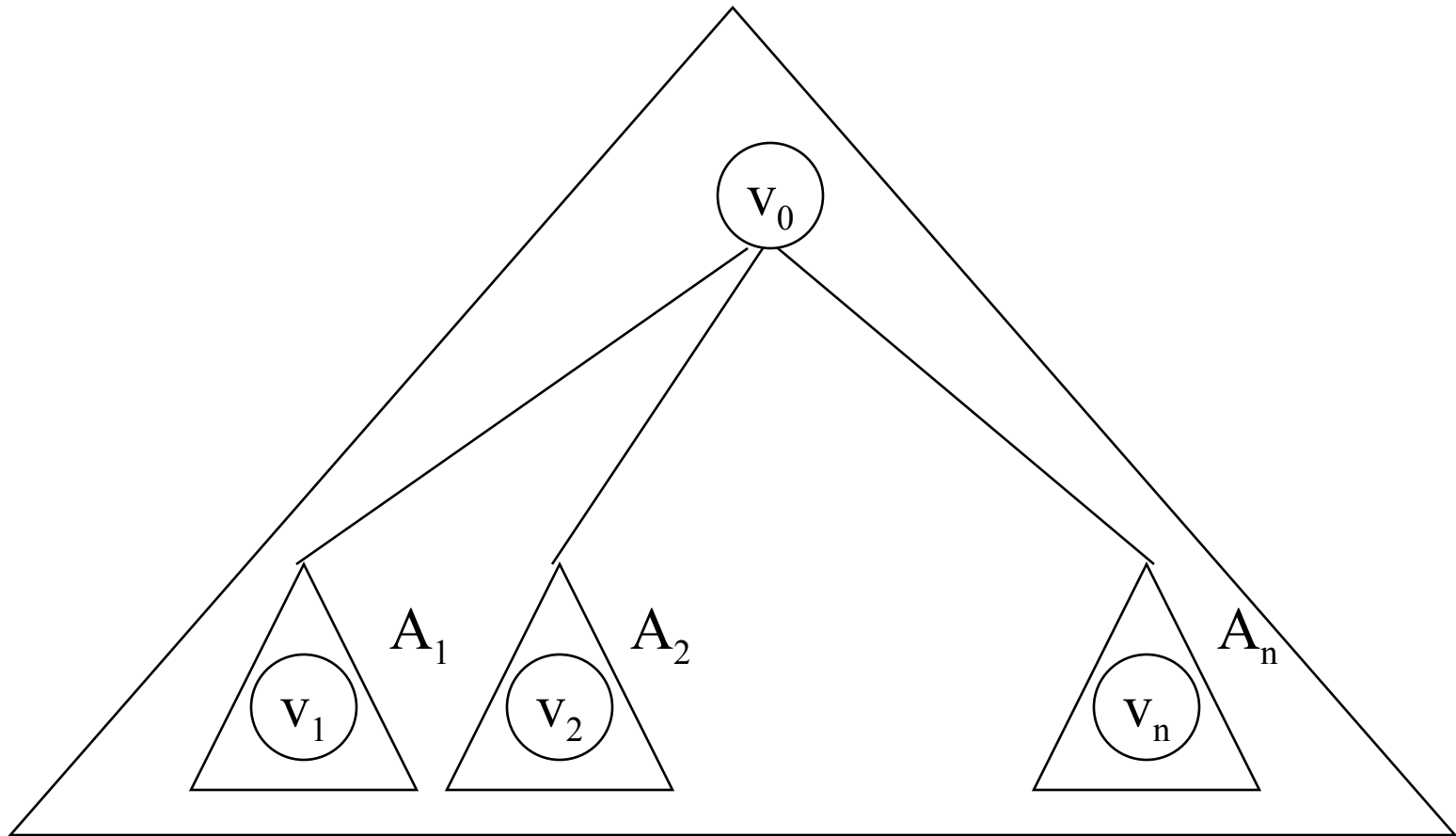
2. La raíz del árbol  $A_i$  es  $v_i$

$v_i$  es llamado hijo de  $v_0$

$v_0$  es llamado padre de  $v_i$ .



# ÁRBOLES GENERALES



# ÁRBOLES GENERALES (N-ARIOS)

## Niveles jerárquicos de un árbol

Cada nodo de un árbol  $A$  tiene asignado un número de nivel. La raíz de  $A$  tiene asignado el nivel 0. Los demás nodos de  $A$  tienen asignado un número de nivel que es mayor en 1 al número de nivel de su padre.

Los nodos del mismo nivel se dicen hermanos.

## Terminología Fundamental

Sea  $n_1, n_2, \dots, n_k$  secuencia de vértices o nodos tal que  $n_i$  es el padre de  $n_{i+1}$ , para todo  $i$  de 1 a  $k$

- Denominamos **raíz** al único nodo que no tiene padre, que corresponde al nodo de nivel 0.
- Denominamos **hoja** (o **terminal**) a los nodos que no tienen hijos.

# ÁRBOLES GENERALES (N-ARIOS)

- La secuencia de nodos  $(n_1, n_2, \dots, n_k)$  se denomina camino del nodo  $n_1$ , al nodo  $n_k$
- La **longitud de camino** es el número de nodos de la secuencia menos uno. La longitud de camino está determinado por el número de aristas del camino, no por el número de nodos del camino. Así el camino  $(a,b,c)$  es de longitud 2, el camino  $(a, b)$  es de longitud 1, el camino  $(a)$  es de longitud 0.
- El camino de un nodo hacia si mismo es de longitud 0.
- Si existe un camino del nodo  $a$  hacia el nodo  $b$ , se dice que  $a$  es **antecesor** de  $b$  y  $b$  es **sucesor** o descendiente de  $a$ .

# ÁRBOLES GENERALES (N-ARIOS)

- Un antecesor o un descendiente de un nodo que no sea el mismo, recibe el nombre de **antecesor propio** o **descendiente propio** respectivamente. La raíz de un árbol es el único nodo que no tiene antecesores propios. Un nodo sin descendientes propios se denomina hoja.
- La **altura** o **profundidad** de un árbol, es la longitud de camino más largo de ese nodo a una hoja. La altura del árbol es la altura de la raíz a ese nodo.

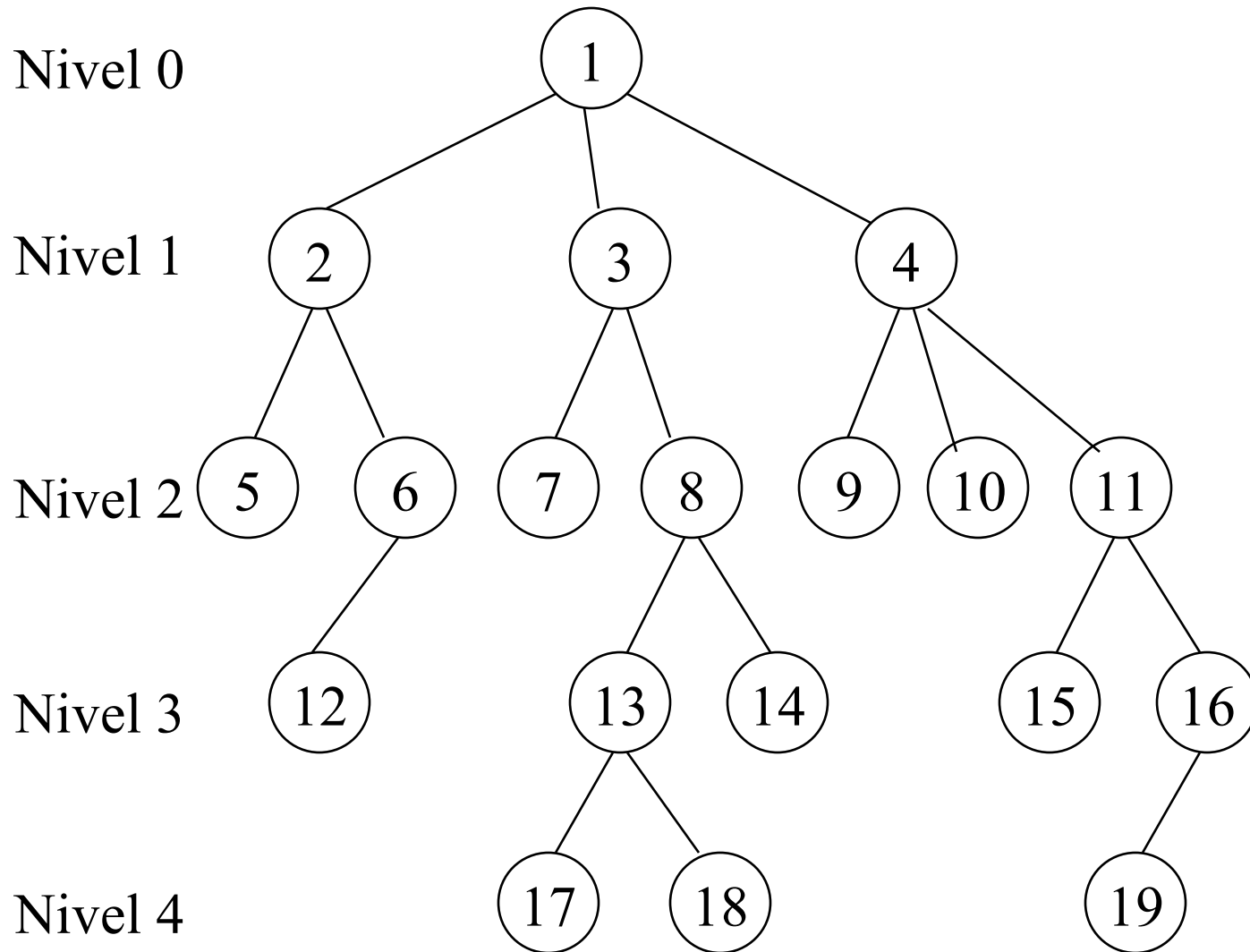
# ÁRBOLES GENERALES (N-ARIOS)

## Otros términos

- El **momento** o **tamaño** de un árbol, es el número de nodos que contiene el árbol.
- El **peso** es el número de hojas del árbol.
- Llamamos **grado de un nodo** al número de hijos del nodo. Podemos decir entonces que las hojas son los nodos de grado 0. Todos los nodos tiene hijos a excepción de las hojas. Todos los nodos tienen padre a excepción de la raíz.

# ÁRBOLES GENERALES (N-ARIOS)

**Ejemplo:** Considere el árbol A siguiente:



# ÁRBOLES GENERALES (N-ARIOS)

Raíz:	1	
Momento:	19	
Peso:	10	
Altura:	4	
Grado(3):	2	
Grado(4):	3	
Descendientes de 8:	8, 13, 14, 17 y 18	
Descendientes propios de 8:	13, 14, 17 y 18	
Antecesoros de 8:	1, 3 y 8	
Antecesoros propios de 8:	1 y 3	

# ÁRBOLES GENERALES (N-ARIOS)

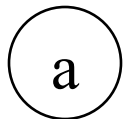
**Ejemplo:** Sea el árbol de vértices  $V$  y altura  $h$

Definimos  $h(u)$  la altura del subárbol con raíz  $u$

Si  $V = \Phi$   $h = -1$

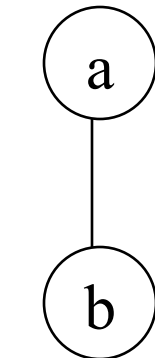
$V = \{a\}$

$h = 0$



$V = \{a, b\}$

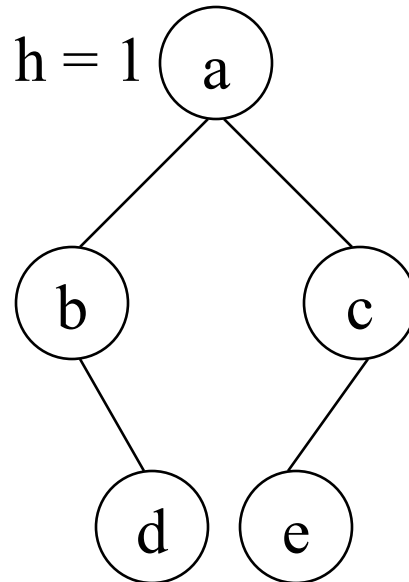
$h = 1$



$h(b) = 0$

$V = \{a, b, c, d, e\}$

$h = 2$



$h(a) = h = 1$

$h(a) = h = 2$

$h(b) = h(c) = 1$

$h(d) = h(e) = 0$



# ÁRBOLES GENERALES (N-ARIOS)

## TEOREMA

Sea  $A$  un grafo con  $n$  vértices

Las siguientes afirmaciones son equivalentes

1.  $A$  es un árbol
2.  $A$  es conexa y acíclica
3.  $A$  es conexa y tiene  $n-1$  aristas
4.  $A$  es acíclica y tiene  $n-1$  aristas

## TEOREMA

Si  $a, b$  son vértices distintos de un árbol  $A$ , entonces existe un único camino que conecta estos vértices.

Se puede demostrar que como  $A$  es conexo, existe al menos un camino de  $a$  hacia  $b$ . Si hubiera más caminos de este tipo, por medio de dos de ellos, algunas aristas podrían tener ciclos, pero sabemos que un árbol no tiene ciclos.

# ÁRBOLES GENERALES (N-ARIOS)

## PRIMITIVAS DE ACCESO

**a) HIJO\_MAS\_IZQ(n):** devuelve el nodo hijo más a la izquierda del nodo  $n$  en el árbol. Devuelve Nulo si  $n$  es una hoja y, por tanto no tiene hijos.

**b) HERMANO\_DER(n):** devuelve el hermano a la derecha, el cual se define como el nodo  $m$  que tiene el mismo padre  $p$  que  $n$ , de forma que  $m$  esta inmediatamente a la derecha de  $n$  en el ordenamiento de los hijos de  $p$ .

**c) VALOR(n):** devuelve el campo valor del nodo  $n$  en el árbol.

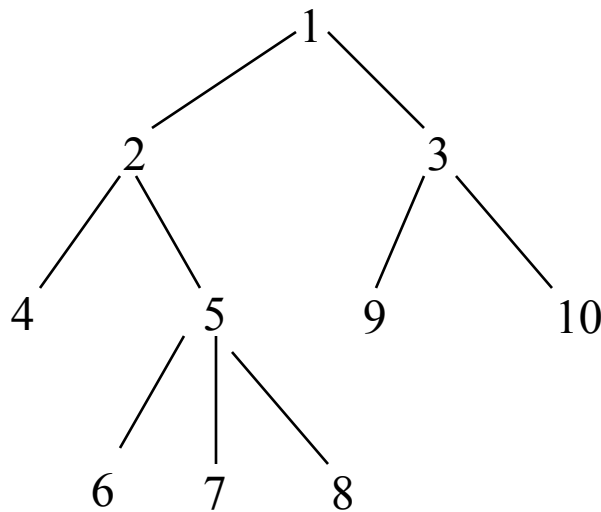
**d) RAIZ:** devuelve el nodo raíz del árbol o Nulo si el árbol es nulo.

**e) INIT\_ARBOL:** Permite inicializar el árbol como árbol nulo.

# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

## Representación mediante arreglos

Cuando se tiene los nodos etiquetados con enteros, es posible usar una representación mediante un arreglo unidimensional  $A$ , en donde el índice  $i$  representa el valor del nodo  $i$ , y  $A(i)$  representa el valor del padre de  $i$ .



$A(i)$  contiene el padre del nodo  $i$

$A(4) = 2$  es el padre del nodo 4

0	1	1	2	2	5	5	5	3	3		
1	2	3	4	5	6	7	8	9	10		

# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

## Representación mediante lista de hijos

Cuando los nodos están etiquetados con mayor información, es conveniente utilizar una estructura de listas, en donde cada nodo es una estructura de la forma

Registro    Nodo

Inicio

    T        Valor

    Nodo    \*sig

FinRegistro

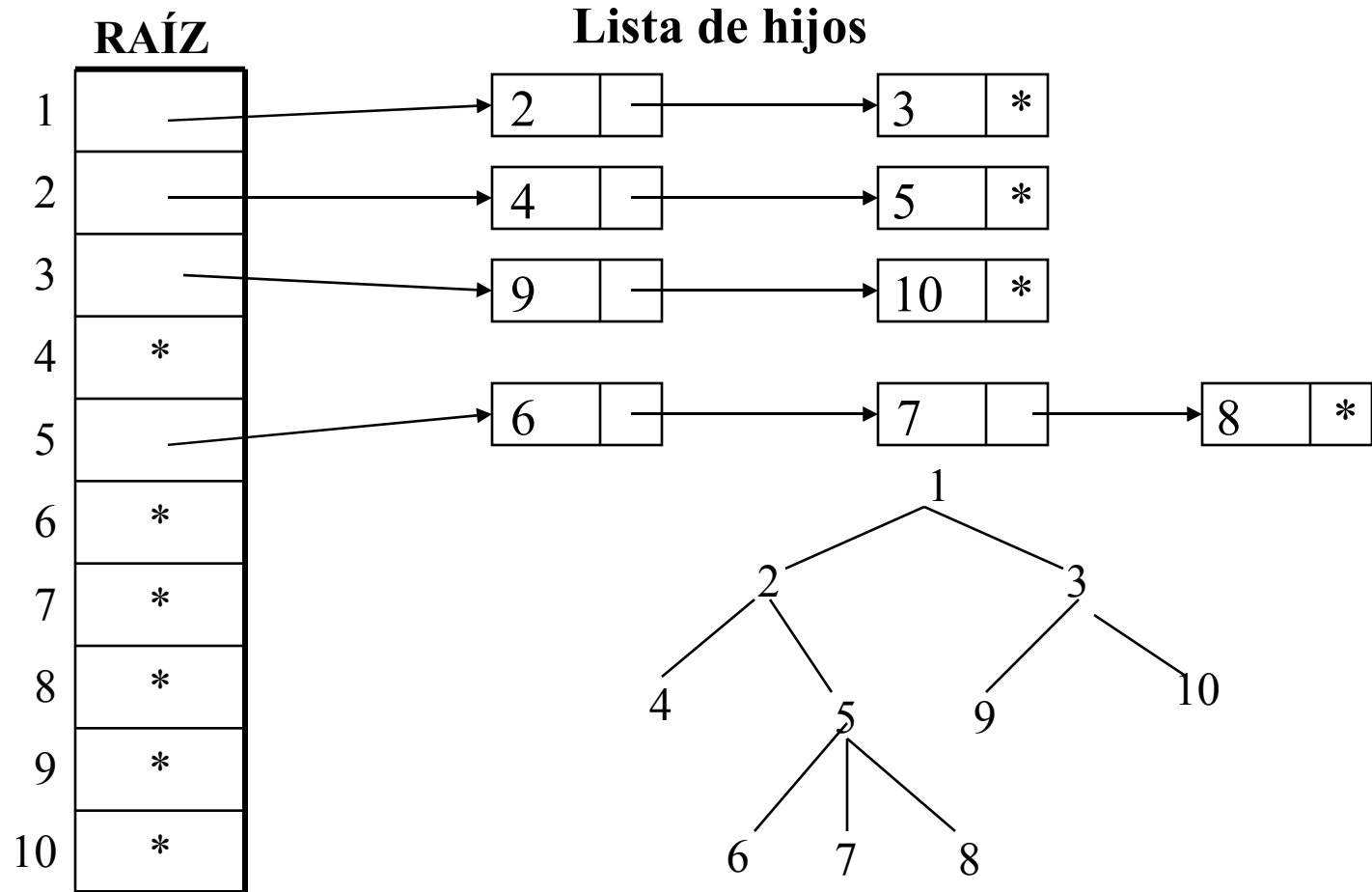
Este modelo presenta la restricción que el número de nodos está limitado al tamaño del vector.

# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

## Representación mediante lista de hijos

**RAÍZ** es un arreglo de tamaño  $n$  (tamaño de  $V$ )

**RAÍZ( $i$ )** es la cabeza de la lista que contiene los hijos del nodo  $i$ .



# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

**Acción Recorrido(N)**

Inicio

Para I desde 1 hasta N

$p = \text{Raíz}(I)$

Mientras  $p \neq \text{Nulo}$

    Escribir  $p \rightarrow \text{Valor}$

$p \leftarrow p \rightarrow \text{sig}$

FinMientras

FinPara

Fin

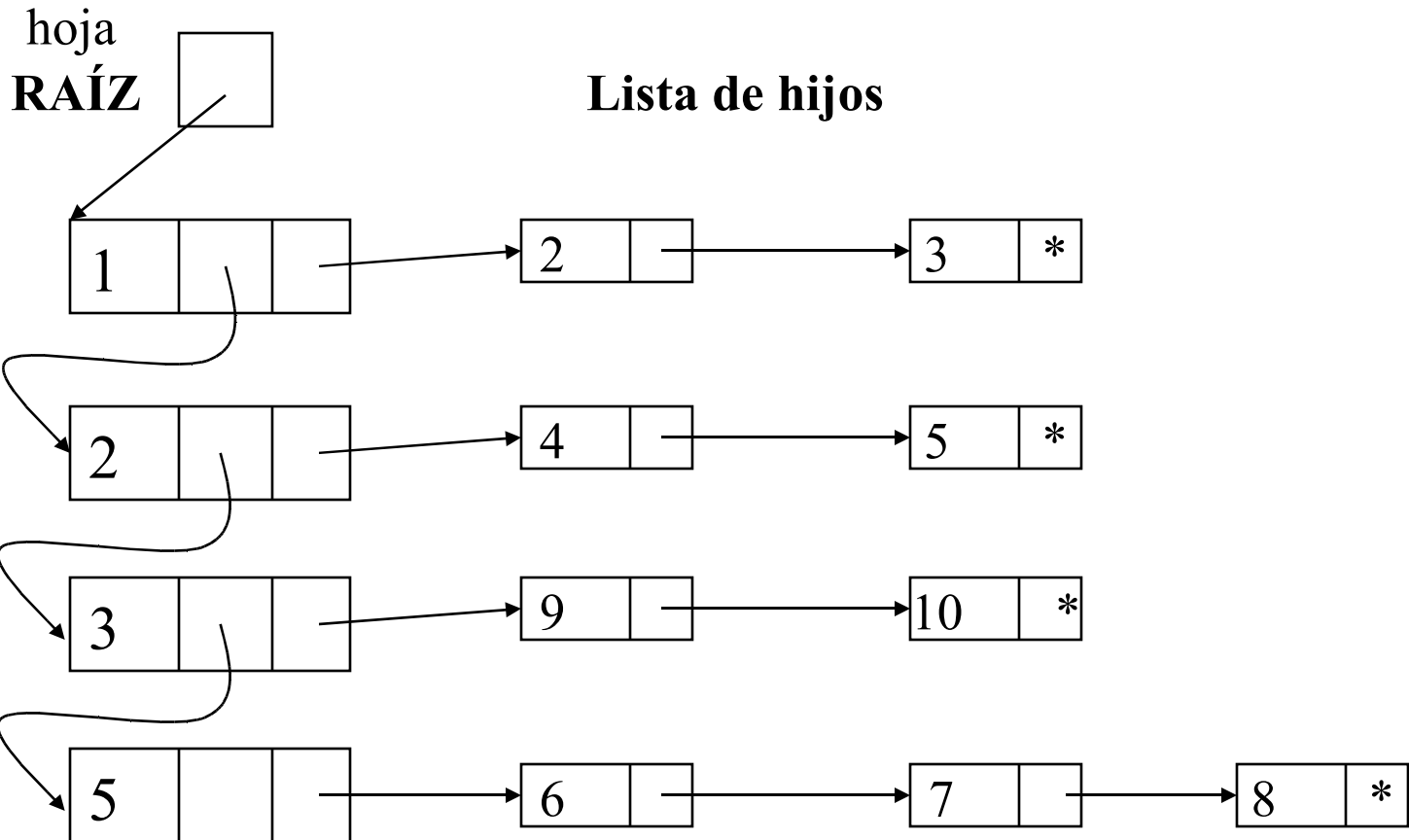
# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

Raíz(i) es la cabeza de la lista de los hijos del nodo i

i	Raíz(i)	Hijos del nodo i
1	Raíz(1)	2, 3
2	Raíz(2)	4, 5
3	Raíz(3)	9, 10
4	Raíz(4) = Nulo	No tiene
5	Raíz(5)	6, 7, 8
6	Raíz(6) = Nulo	No tiene
7	Raíz(7) = Nulo	No tiene
8	Raíz(8) = Nulo	No tiene
9	Raíz(9) = Nulo	No tiene
10	Raíz(10) = Nulo	No tiene

# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

La acción recorre todo el vector RAÍZ, aunque alguna raíz sea nula, por tanto no tiene hijos. Este recorrido resulta innecesario, y se puede solucionar haciendo que el vector Raíz sea una lista enlazada que contenga un nodo por cada nodo del árbol que no es

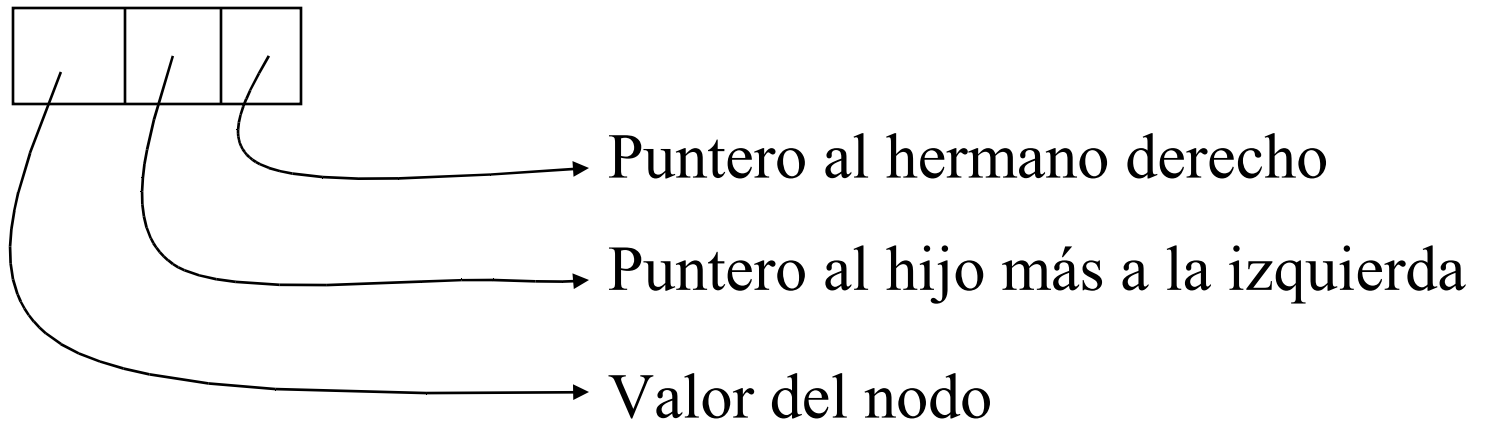




# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

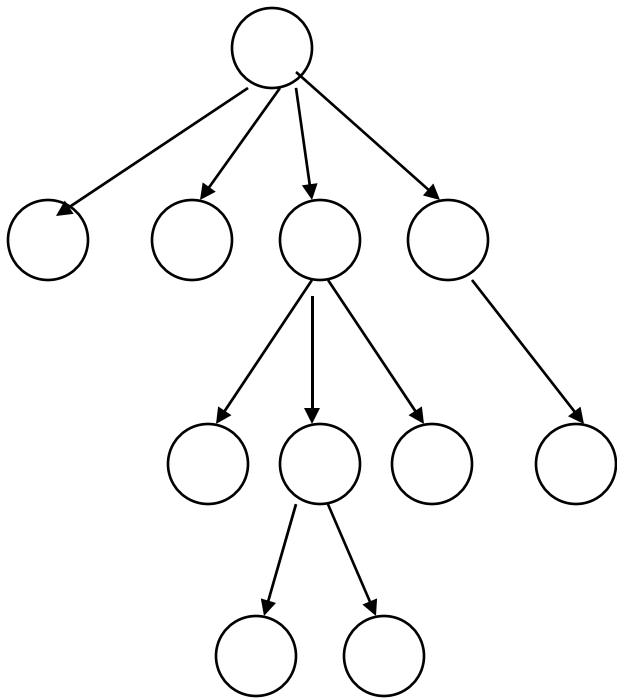
**Puntero al hijo de más a la izquierda–hermano derecho (HI-HD)**

Los nodos tienen dos apuntadores uno al hijo de más a la izquierda y el otro al hermano derecho.

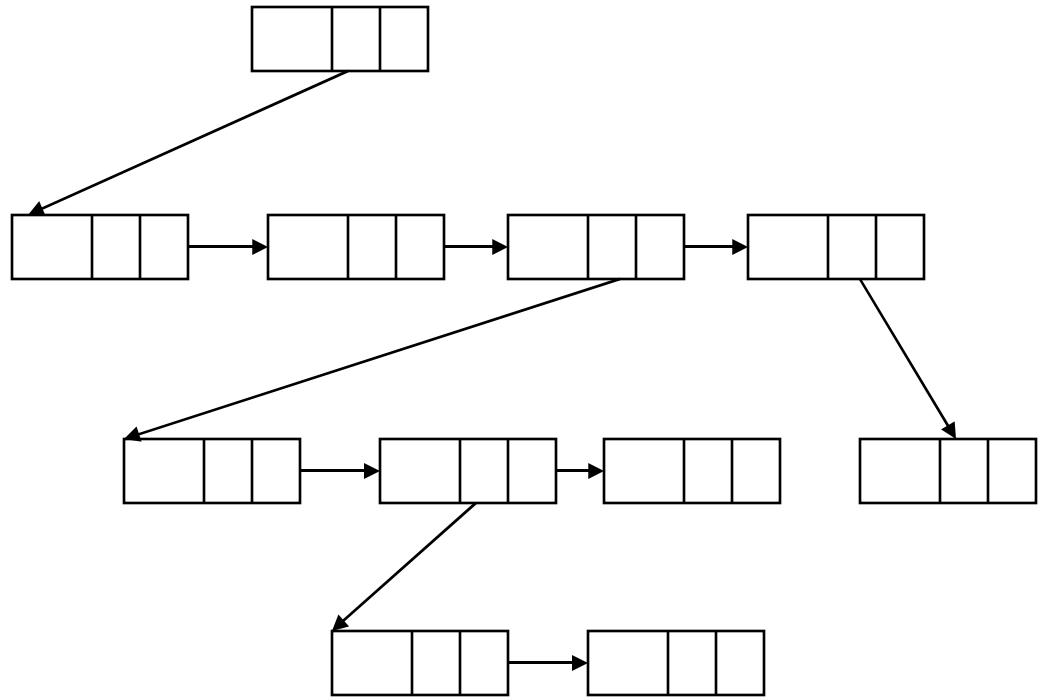


# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

**Ejemplo:** El árbol de la figura (a) puede representarse mediante el modelo HI-HD mostrado en la figura (b)



(a)



(b)

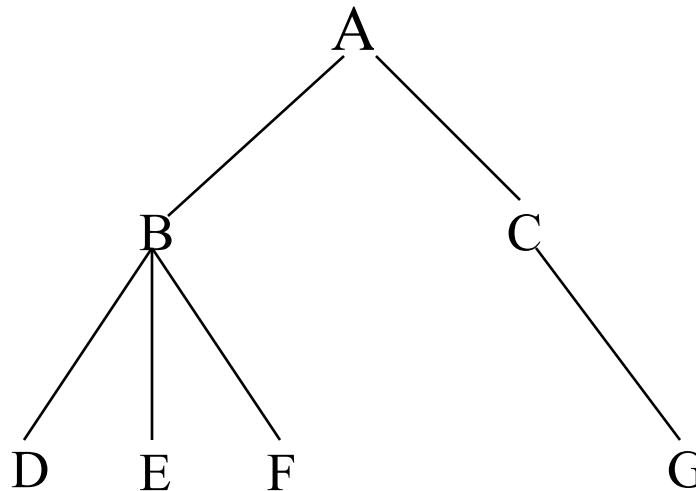
# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

**Representación mediante cursores hijo más a la izquierda-hermano derecho**

La variable **CABEZA** apunta a la dirección de la raíz del árbol (**VALOR(i)**).

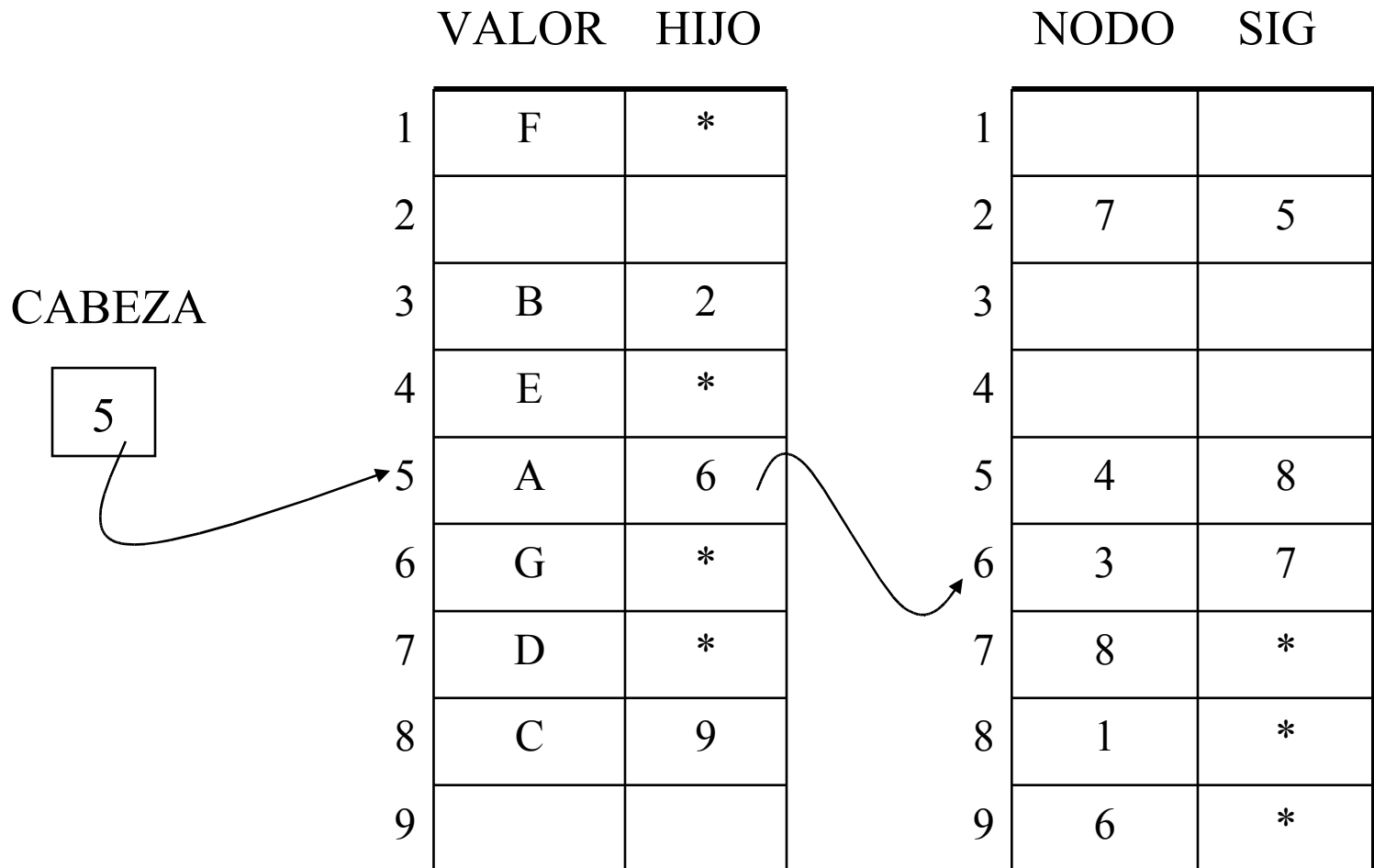
**HIJO(i)** apunta a la posición (**NODO(HIJO(i))**) donde se encuentra la dirección del hijo de más a la izquierda.

**SIG(i)** apunta a la posición (**NODO(i)**) donde se encuentra la dirección del hermano derecho.



# REPRESENTACIÓN EN MEMORIA DE ÁRBOLES GENERALES

Representación mediante cursores hijo más a la izquierda-hermano derecho



## Mediante hijo más a la izquierda-hermano derecho

RAIZ variable global apunta a la raíz del árbol

VALOR(i) contiene el valor del nodo con dirección i

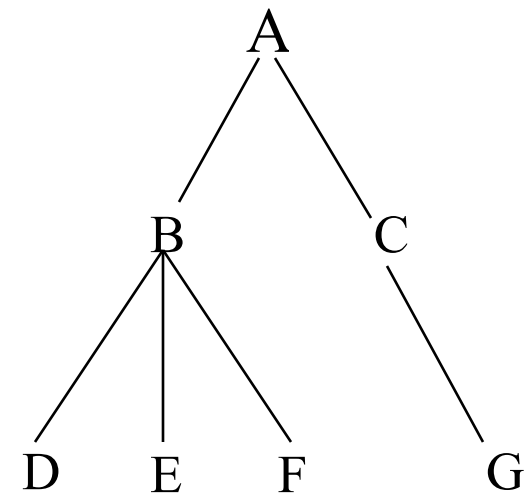
HI(i) contiene la dirección del hijo más a la izquierda del nodo i

HD(i) contiene la dirección del hermano derecho del nodo i

RAIZ

5

	HI	VALOR	HD
1	0	F	0
2			
3	7	B	8
4	0	E	1
5	3	A	0
6	0	G	0
7	0	D	4
8	6	C	0
9			



# RECORRIDO DE ÁRBOLES GENERALES

Considérese el árbol con Raíz y los subárboles  $A_1, A_2, A_3, \dots, A_n$

## ORDEN PREVIO

Raíz

Los nodos de  $A_1$  en orden previo

Los nodos de  $A_2$  en orden previo

.....

Los nodos de  $A_n$  en orden previo

## ORDEN SIMÉTRICO

Los nodos de  $A_1$  en orden simétrico

Raíz

Los nodos de  $A_2$  en orden simétrico

Los nodos de  $A_3$  en orden simétrico

.....

Los nodos de  $A_n$  en orden simétrico

# RECORRIDO DE ÁRBOLES GENERALES

## ORDEN POSTERIOR

Los nodos de  $A_1$  en orden posterior

Los nodos de  $A_2$  en orden posterior

.....

Los nodos de  $A_n$  en orden posterior

Raíz

## ALGORITMOS RECURSIVOS PARA EL RECORRIDO DE UN ARBOL GENERAL

Los algoritmos son abstractos, dependerá de la estructura de datos en la que se implementen

### Acción **ORDEN\_PREVIO(n)**

Inicio

Listar n

Para cada hijo h de n de izquierda a derecha

ORDEN\_PREVIO(h)

FinPara

Fin

# RECORRIDO DE ÁRBOLES GENERALES

**Acción ORDEN\_SIMÉTRICO(n)**

Inicio

Si n es hoja

Listar n

Sino

$h \leftarrow \text{HIJO\_MAS\_IZQ}(n)$

ORDEN\_SIMÉTRICO(h)

Listar n

Para cada hijo h de n excepto el de más a la izquierda

ORDEN\_SIMÉTRICO(h)

FinPara

FinSi

Fin



# RECORRIDO DE ÁRBOLES GENERALES

**Acción ORDEN\_POSTERIOR(n)**

Inicio

Para cada hijo  $h$  de  $n$

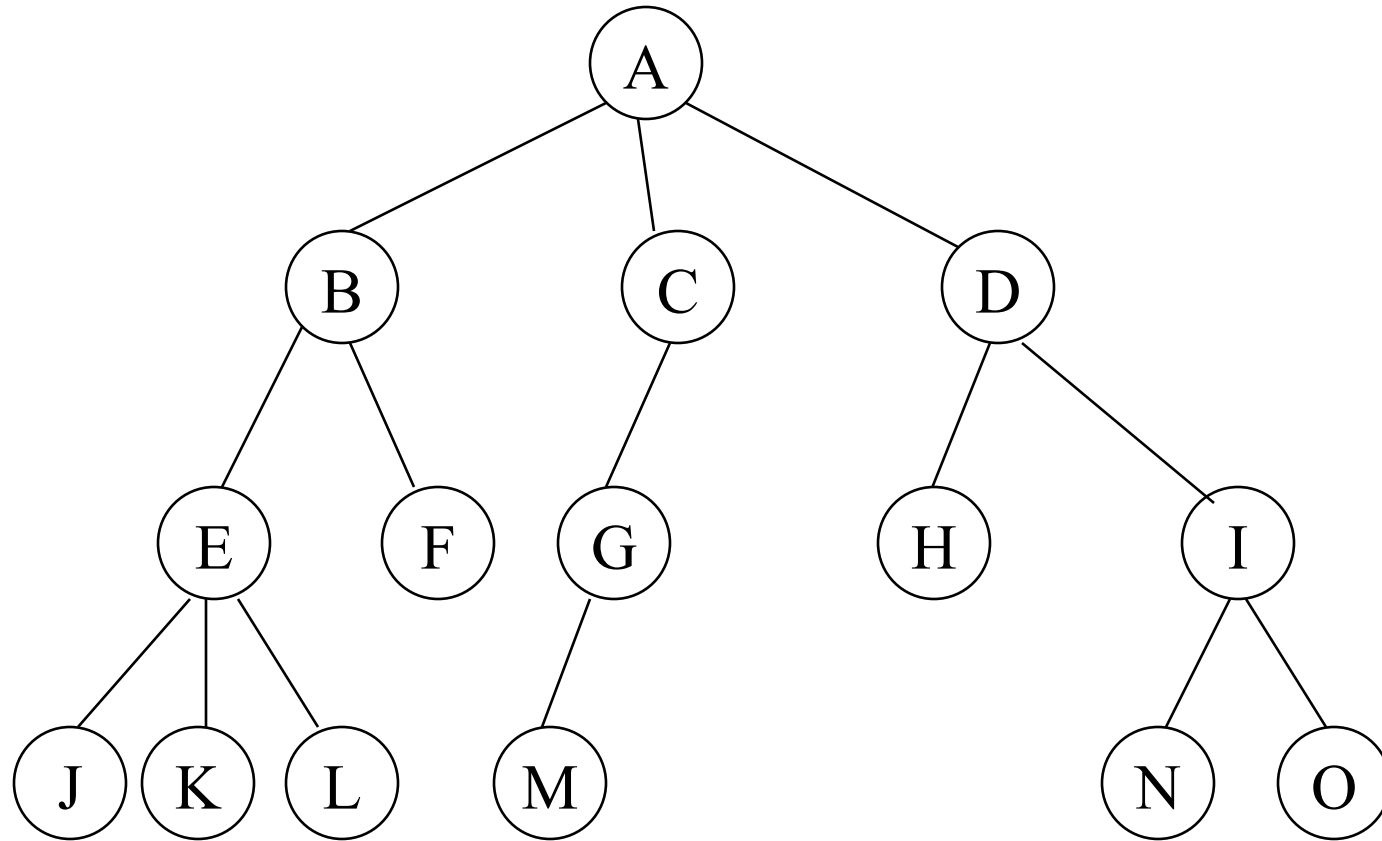
ORDEN\_POSTERIOR( $h$ )

FinPara

Listar  $n$

Fin

# RECORRIDO DE ÁRBOLES GENERALES



Orden previo: A, B, E, J, K, L, F, C, G, M, D, H, I, N, O

Orden simétrico: J, E, K, L, B, F, A, M, G, C, H, D, N, I, O

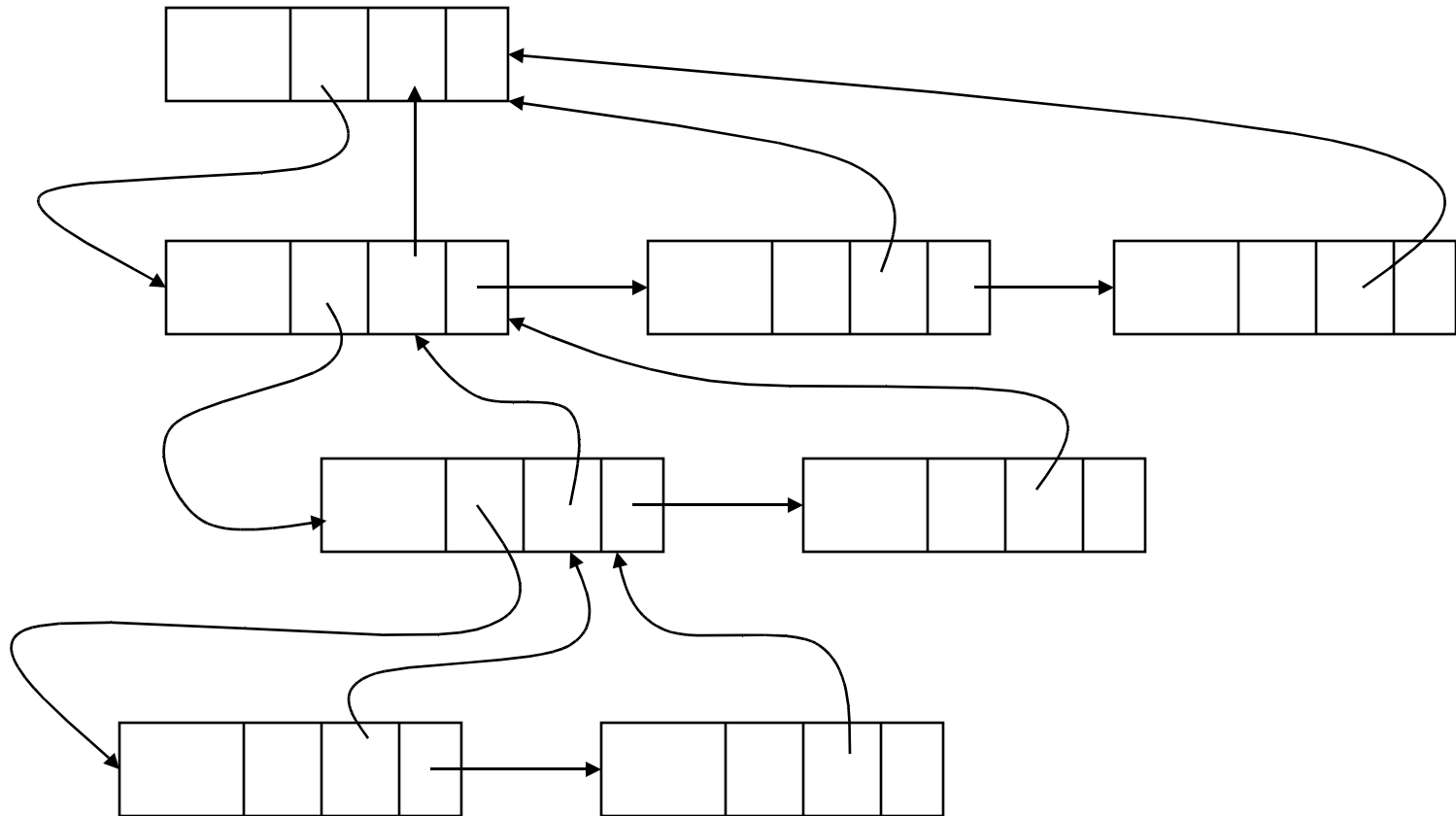
Orden posterior: J, K, L, E, F, B, M, G, C, H, N, O, I, D, A

# ÁRBOL MULTILENLACE

## Árbol de enlace triple

Los nodos tienen tres apuntadores, uno hacia el hijo de más a la izquierda, el segundo apunta al nodo padre, y el tercero apunta al hermano derecho.

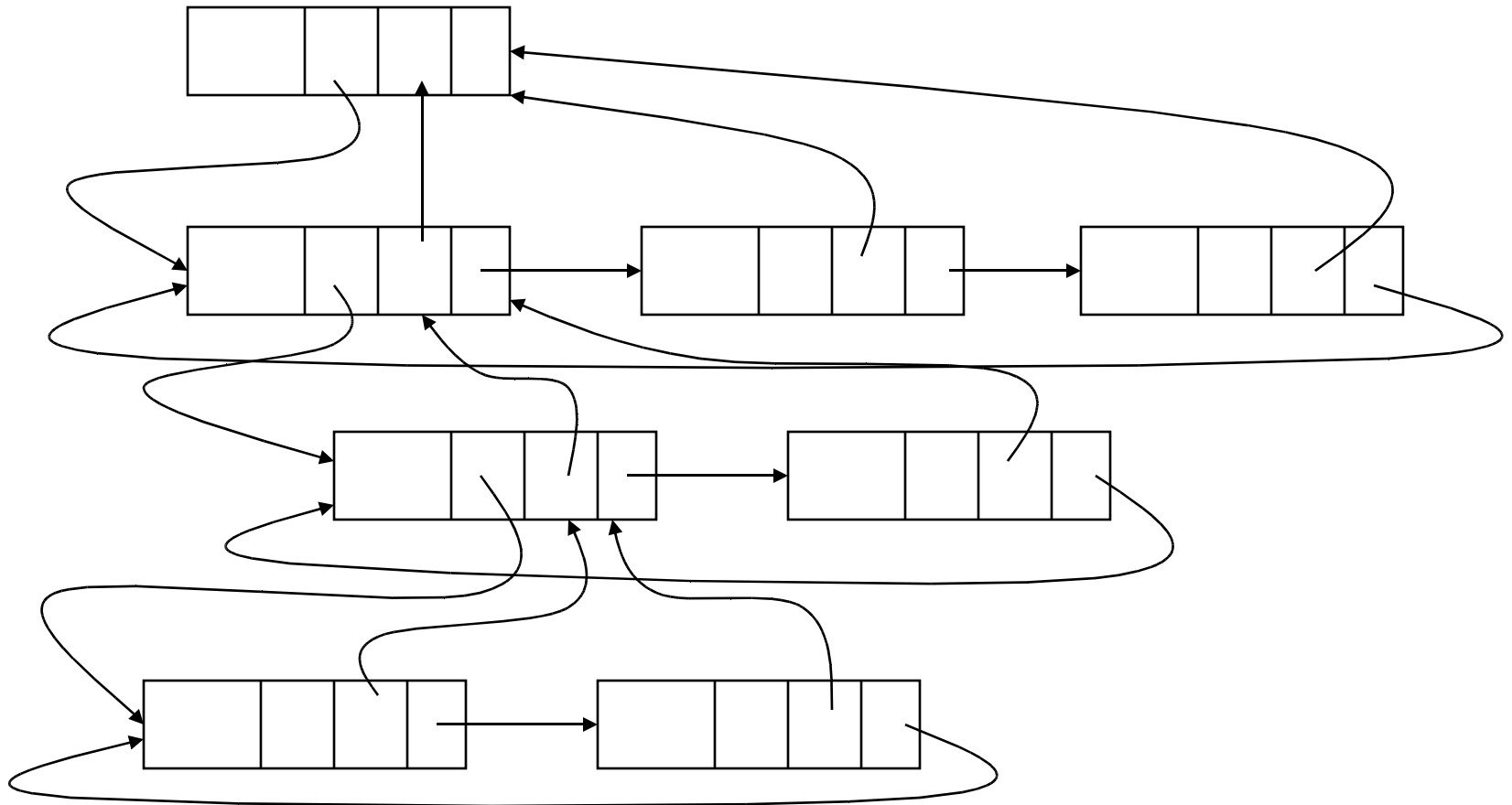
Esta representación presenta una ventaja para volver al nivel superior.



# ÁRBOL MULTILENLACE

## Árbol con enlace en anillo

Resulta conveniente utilizar listas circulares que cierren cadenas de árboles multienlace. Cada nodo tiene un puntero al hijo de más a la izquierda, uno al hermano derecho y otro al padre: En el último de los hijos se reemplaza el puntero nulo por un puntero al primero de los hermanos, conformando una lista circular de hermanos, facilitando el acceso a cualquiera de ellos.



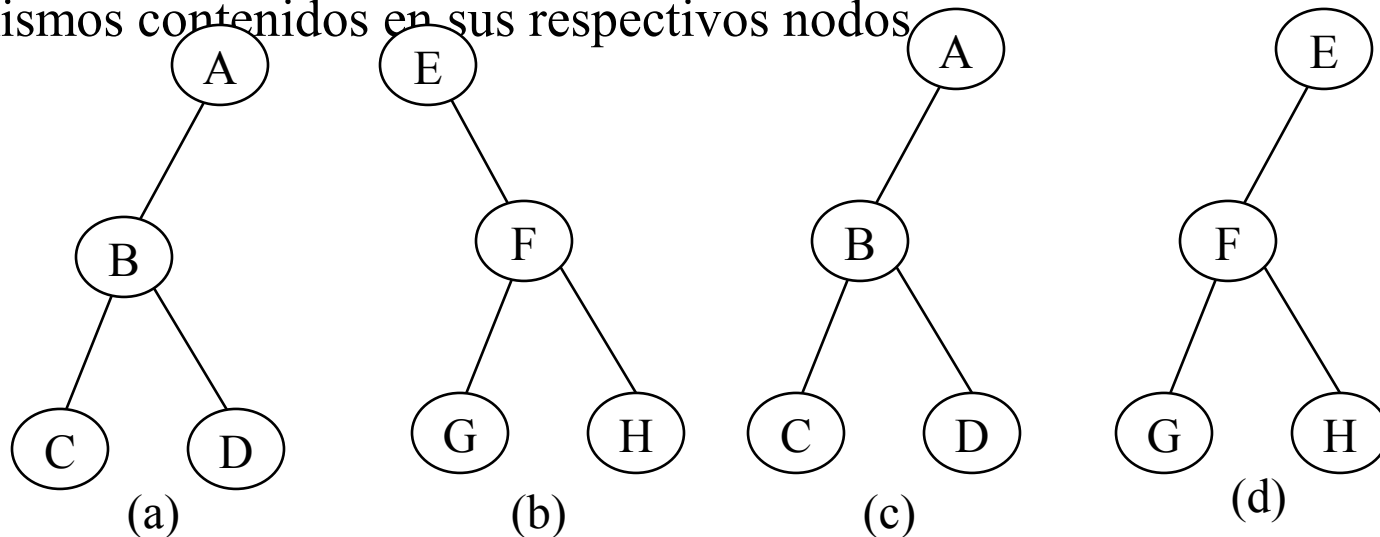
# ÁRBOLES SIMILARES

## ÁRBOLES SIMILARES

Sean  $A_1$  y  $A_2$  dos árboles

$A_1$  y  $A_2$  se llaman árboles similares si tienen la misma estructura (misma forma).

$A_1$  y  $A_2$  se llaman árboles copias, si son similares y tienen los mismos contenidos en sus respectivos nodos



Los árboles de las figuras (a), (c) y (d) son similares.

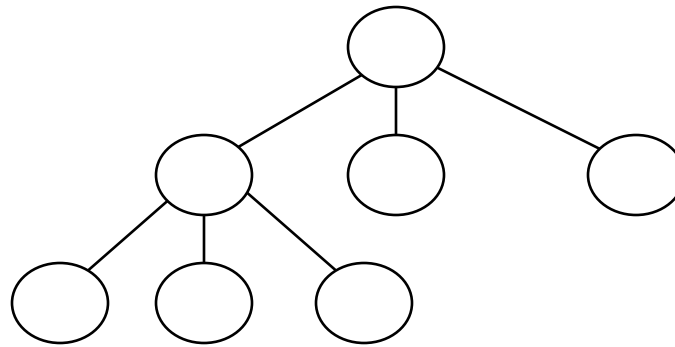
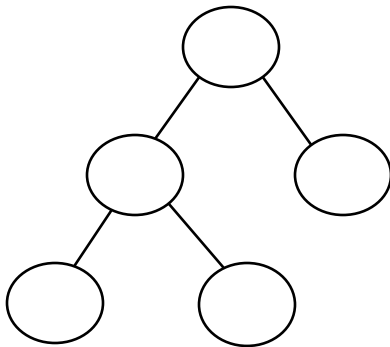
Los árboles de las figuras (a) y (c) son árboles copias.

El árbol (b) no es similar ni copia de (d)

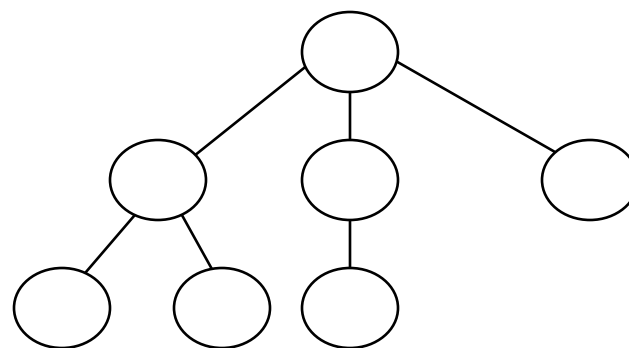
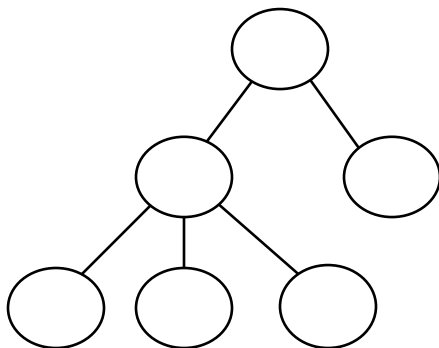
# ÁRBOLES BALANCEADOS

## Árbol balanceado

Todos los nodos que no son hojas tienen el mismo número de ramas. Un árbol balanceado se construye desde la raíz, descendiendo hacia los niveles inferiores. Cada nivel se completa de izquierda a derecha, de modo que todos los niveles excepto el último deben estar completos. Ejemplo: Los siguientes son árboles balanceados



Los siguientes son árboles desbalanceados.



# ÁRBOL BINARIO AB

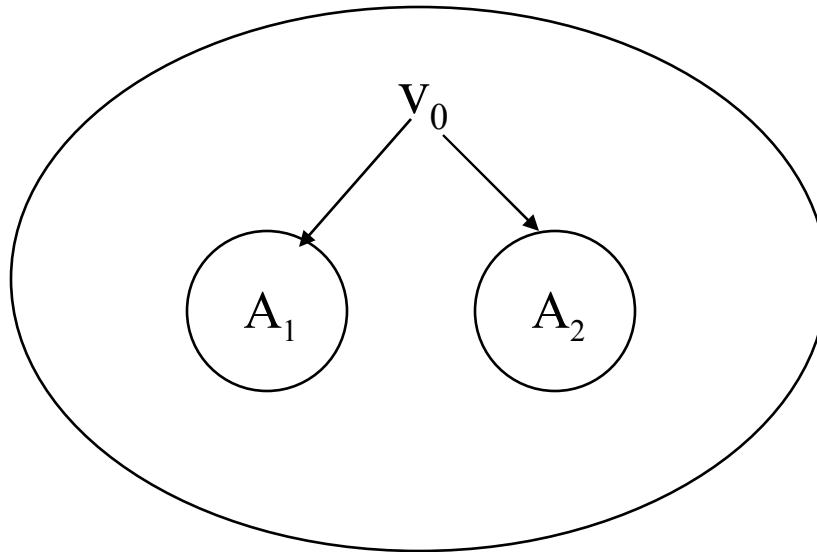
Un árbol binario es un tipo especial de árbol en el que todo nodo del árbol tiene a lo más dos hijos. Son muy utilizados para realizar búsquedas y ordenamientos. También son utilizados para representar expresiones aritméticas.

## **Definición**

Un árbol binario AB se define como un conjunto de elementos llamados nodos o vértices, de forma que:

- A es vacío, en cuyo caso se llama árbol vacío o árbol nulo, o
- A contiene un nodo distinguido  $v_0$  llamado raíz de A y los nodos restantes de A forman un par de conjuntos ordenados de árboles  $A_1$  y  $A_2$   
 $A_1$  es llamado subárbol izquierdo y  $A_2$  es llamado subárbol derecho.

# ÁRBOL BINARIO AB



## Definición

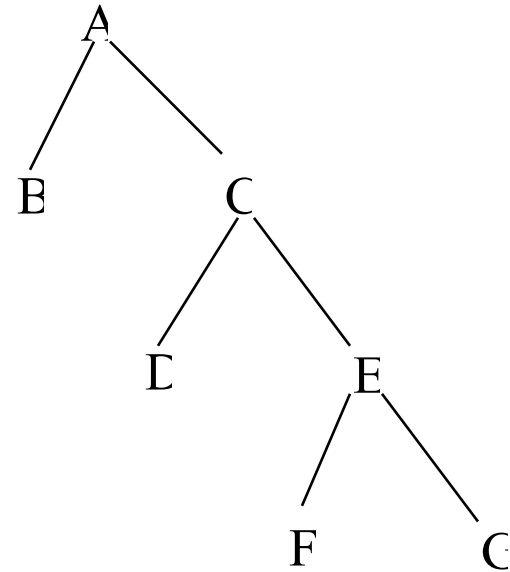
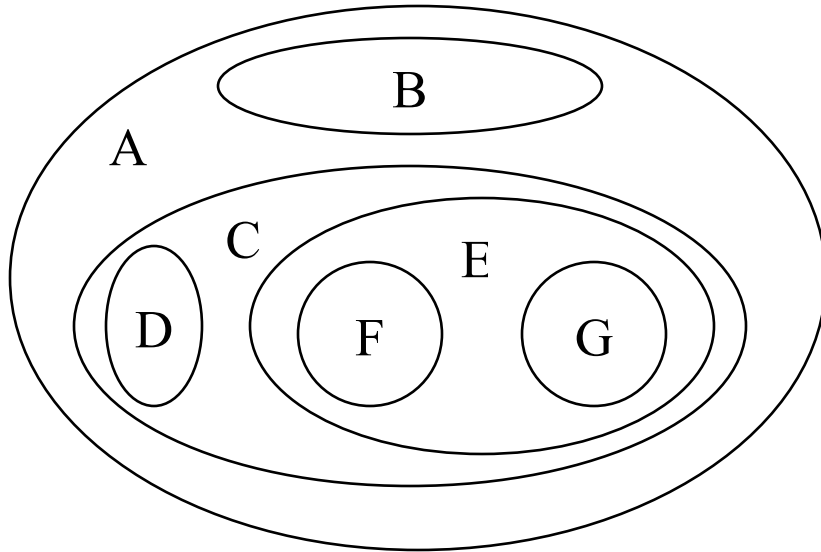
Se define un árbol binario como

- Un árbol vacío
- Cada nodo no tiene hijos, o
  - tiene un hijo izquierdo
  - tiene un hijo derecho
  - tiene un hijo izquierdo y un hijo derecho



# ÁRBOL BINARIO AB

Gráficamente podemos definir un conjunto como



# ÁRBOL BINARIO

## Características de árboles binarios

1. El número máximo de nodos en un árbol binario  $h$  es  $2^{h+1}-1$
2. Un árbol con  $n$  nodos internos tiene  $n+1$  nodos externos (hojas).
3. La altura de un árbol binario lleno con  $n$  nodos internos es aproximadamente  $\log_2 n$ .

# ÁRBOL BINARIO

## Modelo 1: Mediante apuntables a los hijos

Cada nodo consiste de un registro:

Registro Nodo

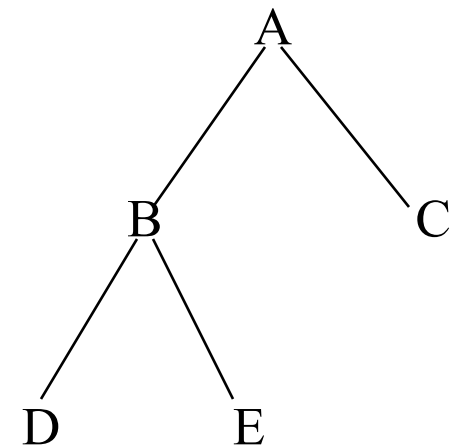
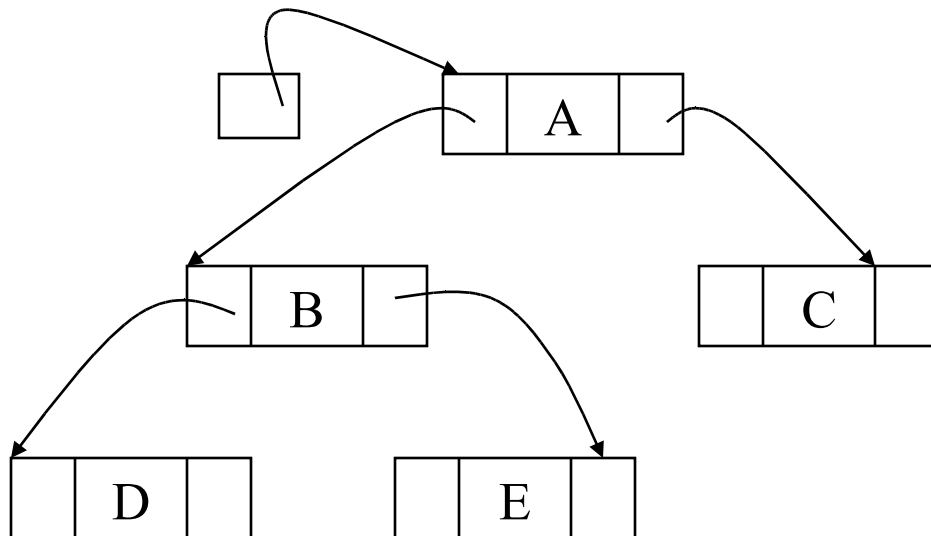
Inicio

T	Valor
---	-------

Nodo	*HI	// apuntador al hijo izquierdo
------	-----	--------------------------------

Nodo	*HD	// apuntador al hijo derecho
------	-----	------------------------------

FinRegistro

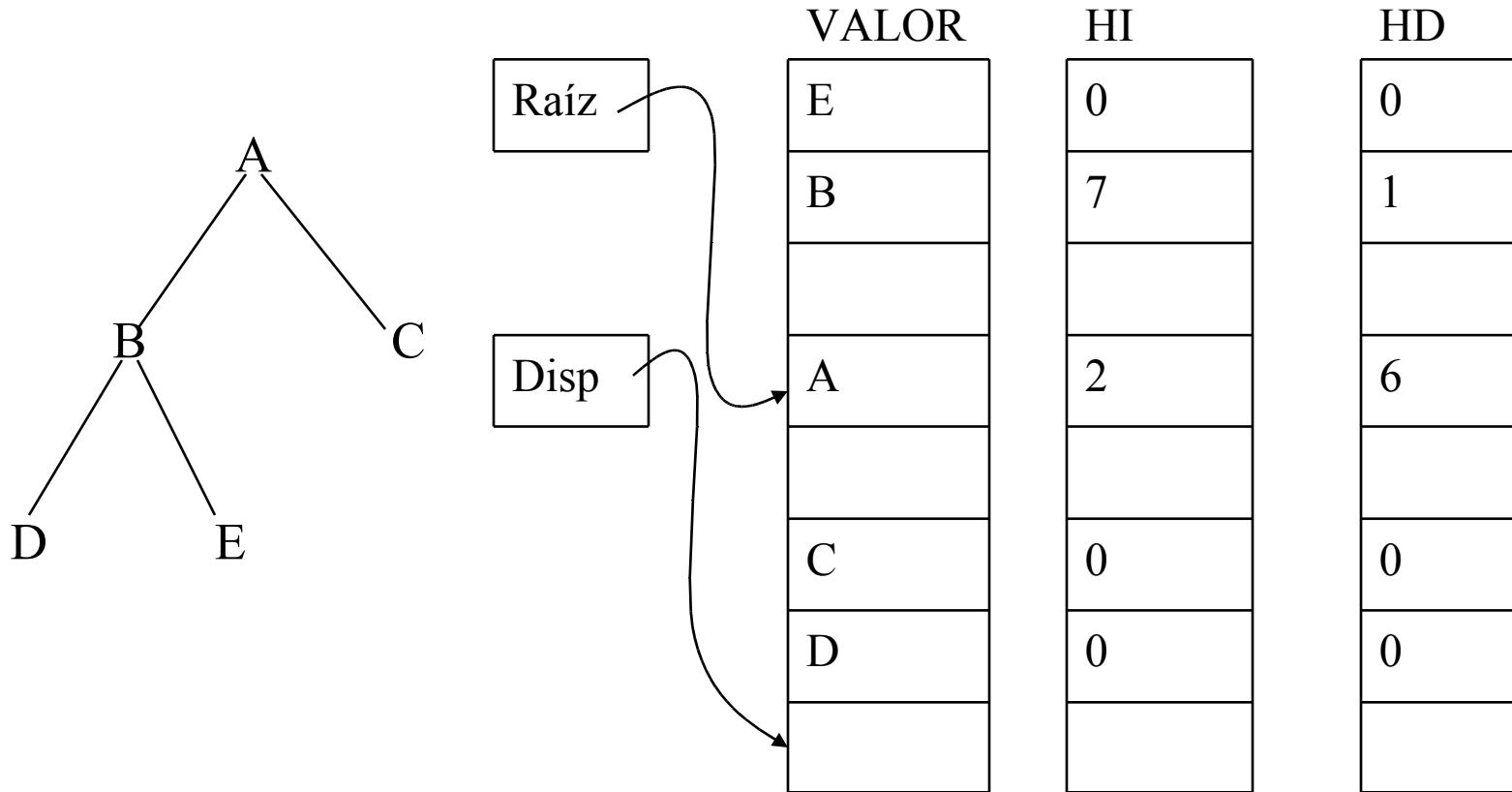


# ÁRBOL BINARIO

## Modelo 2: Mediante arreglos paralelos (cursores)

**Raíz:** apunta al índice de la entrada que contiene el valor de la raíz del árbol.

**Disp:** apunta al índice de la primera entrada disponible en el vector.



# ÁRBOLES Estrictamente BINARIOS

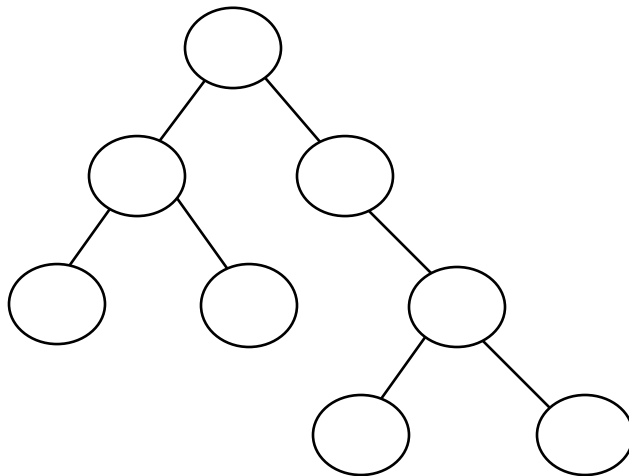
## Definición

Sea  $(A, v_0)$  árbol binario con raíz  $v_0$

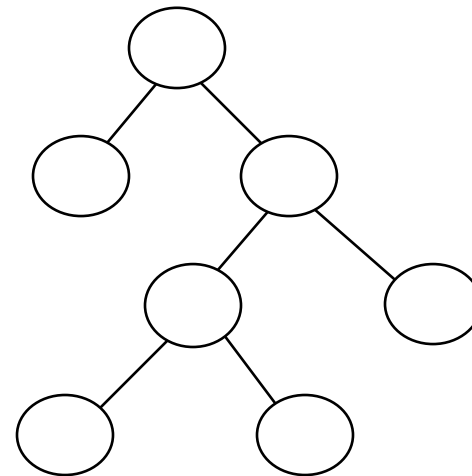
$A$  es un **árbol estrictamente binario**, si cumple

$\forall v \in V$  Si  $v$  no es hoja entonces  $v$  tiene subárboles izquierdo y derecho no vacíos.

Un árbol estrictamente binario con  $n$  hojas tiene  $2n-1$  nodos.



(a)



(b)

El árbol de la Figura (a) no es estrictamente binario, mientras que el árbol de la Figura (b) si lo es.

# ÁRBOLES COMPLETOS Y ÁRBOLES CUASI COMPLETOS

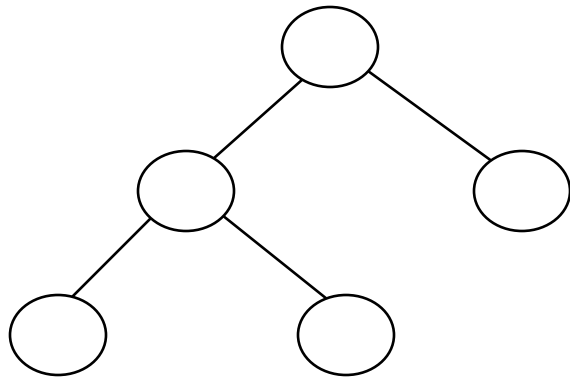
## Definición

Sea  $(A, v_0)$  árbol binario con raíz  $v_0$

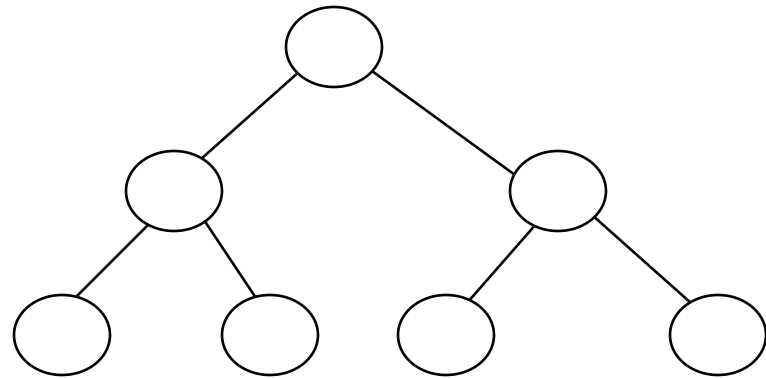
A es un **árbol completo**, si cumple

Si A es de altura h entonces,  $\forall v \in V$  Si v es hoja entonces v está en el nivel h.

Es decir todas las hojas deben estar en el nivel h..



(a)



(b)

El árbol de la figura (a) no es completo, mientras que el árbol de la figura (b) si lo es.

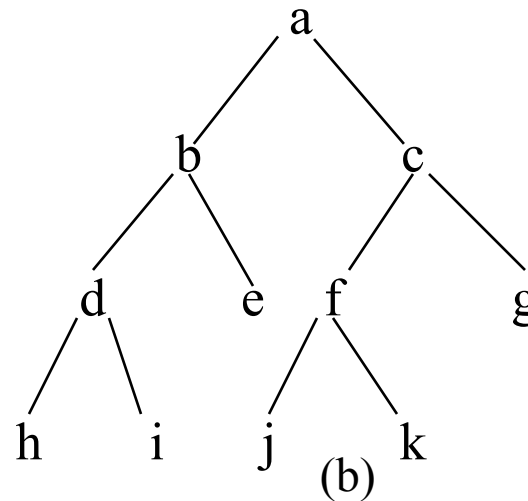
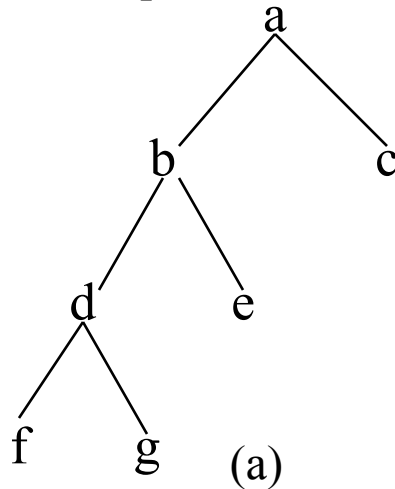
# ÁRBOLES COMPLETOS Y ÁRBOLES CUASI COMPLETOS

## Definición

Sea  $(A, v_0)$  árbol binario con raíz  $v_0$  y profundidad  $h$

$A$  es un **árbol cuasi completo**, si cumple

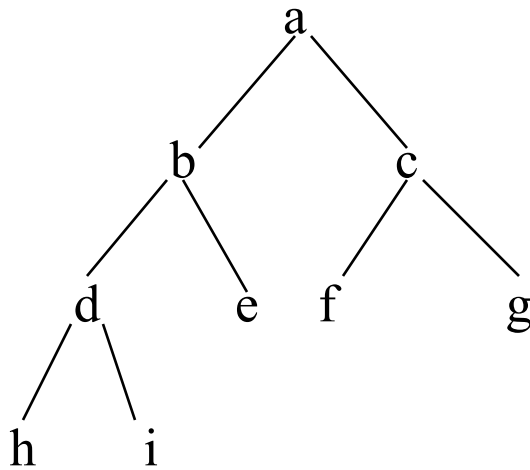
1.  $\forall v \in V$  Si  $v$  es hoja entonces  $v$  está en el nivel  $h$  o nivel  $h-1$ .
2.  $\forall v \in V$  Si  $v$  tiene un descendiente derecho en el nivel  $h$ , entonces todos los descendientes izquierdos de  $v$  que sean hojas también deben estar en el nivel  $h$



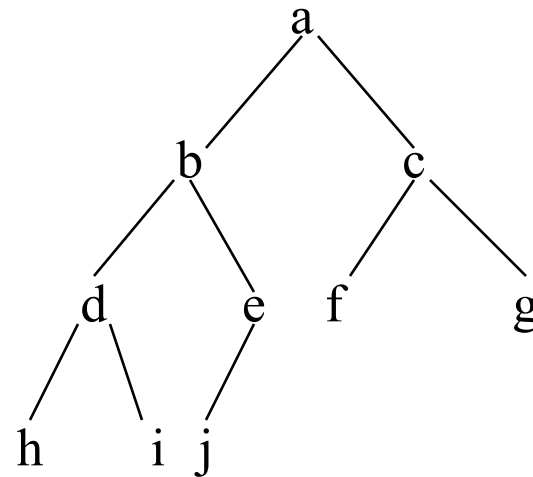
El árbol de la figura (a) no es cuasi completo pues las hojas están el nivel 1, 2 y 3, por lo que no cumple la condición 1.

El árbol de la figura (b) no es cuasi completo pues aunque cumple la condición 1, las hojas están en el nivel 2 y 3. Pero no cumple la condición 2, ya que el nodo  $a$  tiene descendiente derecho  $j$  en el nivel 3, y un descendiente izquierdo  $e$  en el nivel 2.

# ÁRBOLES COMPLETOS Y ÁRBOLES CUASI COMPLETOS



(c)



(d)

El árbol de la figura (c) es estrictamente binario y cumple ambas condiciones, por tanto es cuasi completo.

El árbol de la figura (d) aunque no es estrictamente binario, cumple ambas condiciones, por tanto es cuasi completo.



# ÁRBOLES COMPLETOS Y ÁRBOLES CUASI COMPLETOS

Un árbol binario tiene en el nivel  $k$  a lo más  $2^k$  nodos.

Si el árbol  $A$  es completo,  $A$  es estrictamente binario, por tanto el nivel  $k$  tendrá exactamente  $2^k$  nodos.

Así            el nivel 0 tendrá  $2^0 = 1$  nodos

                 el nivel 1 tendrá  $2^1 = 2$  nodos

                 el nivel 2 tendrá  $2^2 = 4$  nodos

de esto se deduce que un árbol completo de profundidad  $h$  tiene  $2^0 + 2^1 + \dots + 2^h$  nodos, esto equivale a

$$\sum_{i=0}^h 2^i = 2^{h+1} - 1$$

El árbol binario tiene  $2^h$  nodos hojas y  $2^h - 1$  nodos no hojas.

# **REPRESENTACIÓN DE UN ÁRBOL COMPLETO O CUASI COMPLETO**

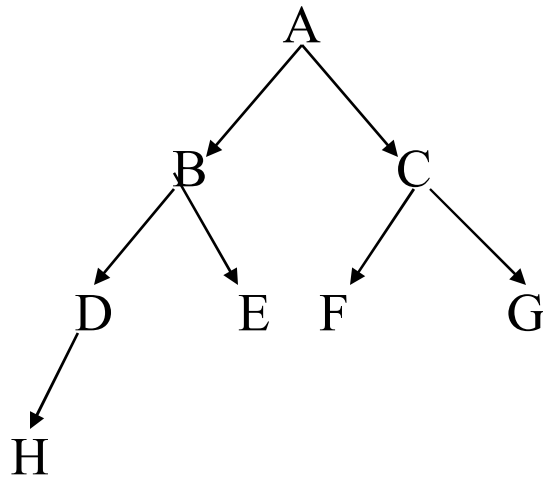
Un árbol binario es cuasi completo, si y sólo si todos sus niveles excepto posiblemente el último, tienen el máximo número de nodos posible y los nodos del último nivel están lo más a la izquierda posible.

## **REPRESENTACIÓN DE UN ÁRBOL COMPLETO O CUASI COMPLETO**

Si enumeramos los nodos de un árbol binario completo o cuasi completo con enteros 1, 2, 3, ... De izquierda a derecha y por niveles de arriba hacia abajo, se puede implementar en un arreglo en donde el índice representa la etiqueta del nodo.

Con este etiquetado se puede determinar fácilmente quienes son los hijos o el padre de cada nodo  $k$ .

# REPRESENTACIÓN DE UN ÁRBOL COMPLETO O CUASI COMPLETO



A	B	C	D	E	F	G	H
1	2	3	4	5	6	7	8

Adviértase que el nodo con índice  $k$ ,

Tiene como hijo izquierdo al nodo con índice  $2*k$

Tiene como hijo derecho al nodo con índice  $2*k+1$

Tiene como padre al nodo con índice  $k/2$

# REPRESENTACIÓN DE UN ÁRBOL BINARIO COMPLETO O CUASI COMPLETO

## Representación secuencial

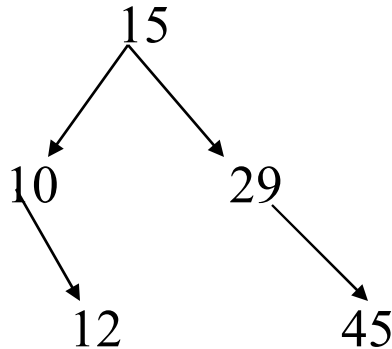
Supongamos que  $A$  es un árbol binario completo o cuasi-completo.

El árbol  $A$  puede mantenerse en forma eficiente en un arreglo ARBOL, en donde:

- La raíz de  $A$  se guarda en  $ARBOL[1]$
- Si un nodo  $N$  esta en la posición  $ARBOL[k]$ , entonces
  - Su hijo izquierdo se encuentra en la posición  $ARBOL[2*k]$
  - Su hijo derecho se encuentra en la posición  $ARBOL[2*k+1]$
- Si  $ARBOL[1] = \text{Nulo}$  el árbol esta vacío

# REPRESENTACIÓN DE UN ÁRBOL BINARIO COMPLETO O CUASI COMPLETO

**Ejemplo:**



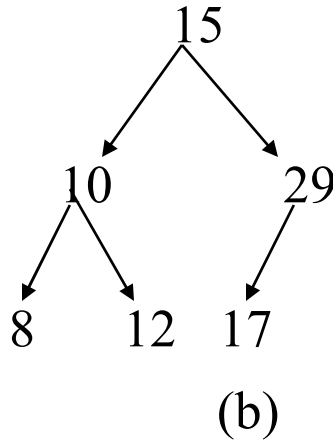
(a)

15	10	29	0	12	0	45	
1	2	3	4	5	6	7	8

El árbol de la figura (a) no es completo. Se necesitan siete espacios para almacenar el árbol, aunque tenga cinco nodos. Si colocáramos nulo para cada hijo izquierdo e hijo derecho de los nodos hojas, el nodo 7, por ejemplo tendría a sus hijos izquierdo y derecho en las posiciones catorce y quince respectivamente. Necesitaríamos quince nodos, lo que hace que resulte ineficiente esta representación para árboles que no son completos.

# REPRESENTACIÓN DE UN ÁRBOL BINARIO COMPLETO O CUASI COMPLETO

**Ejemplo:**



15	10	29	8	12	17		
1	2	3	4	5	6	7	8

El árbol de la figura (b) es cuasi completo, necesita seis espacios, igual al número de nodos.

# RECORRIDO DE ÁRBOLES BINARIOS

Considérese el árbol binario con **RAÍZ** y los subárboles **SAI** y **SAD** subárbol izquierdo y subárbol derecho respectivamente.

## PRE-ORDEN

Raíz

Subárbol izquierdo en pre-orden

Subárbol derecho en pre-orden

## IN-ORDEN

Subárbol izquierdo en in-orden

Raíz

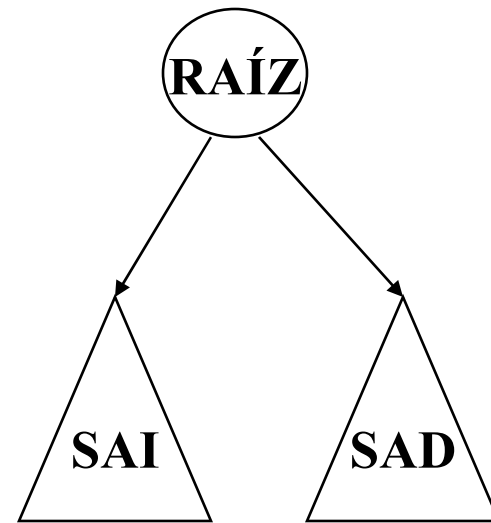
Subárbol derecho en in-orden

## POST-ORDEN

Subárbol izquierdo en post-orden

Subárbol derecho en post-orden

Raíz



# RECORRIDO DE ÁRBOLES BINARIOS

## **Acción PRE-ORDEN(n)**

Inicio

Listar n

PRE-ORDEN(HI(n))

PRE-ORDEN(HD(n))

Fin

## **Acción POST-ORDEN(n)**

Inicio

POST-ORDEN(HI(n))

POST-ORDEN(HD(n))

Listar n

Fin

## **Acción IN-ORDEN(n)**

Inicio

IN-ORDEN(HI(n))

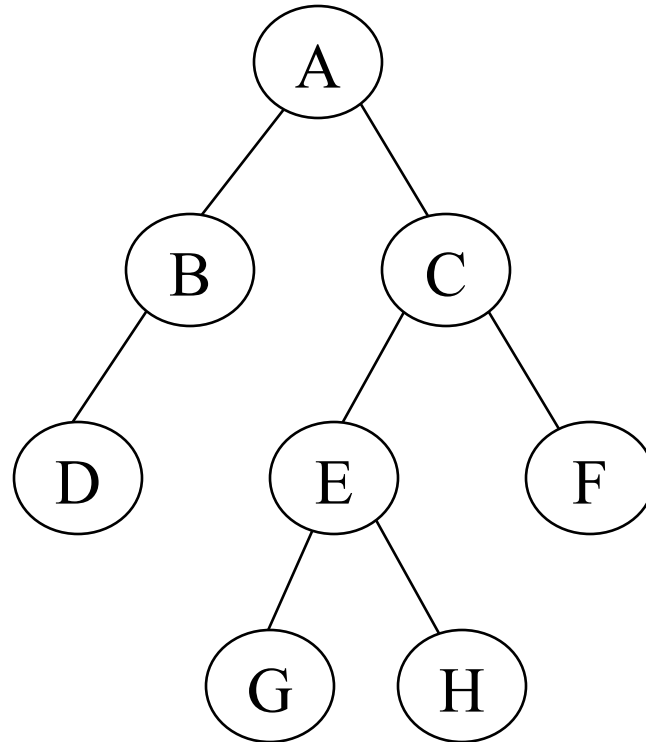
Listar n

IN-ORDEN(HD(n))

Fin



# RECORRIDO DE ÁRBOLES BINARIOS



PRE ORDEN: A, B, D, C, E, G, H, F

IN ORDEN: D, B, A, G, E, H, C, F

POST ORDEN: D, B, G, H, E, F, C, A

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## RECORRIDO INORDEN

**Entrada:** Raíz (raíz del árbol)

**Salida:** recorrido inorden

**Precondición:** El árbol existe y es binario

Raíz apunta a la raíz del árbol

La pila P debe estar vacía

Se recorre el árbol, avanzando hasta su hijo de más a la izquierda y en el camino se va empilando los nodos padres. Cuando se encuentra la hoja más izquierda, se depila el padre, se lista su valor y se avanza por la rama derecha. En el subárbol derecho se aplica el mismo proceso.

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## Acción Recorrido\_Inorden (Raíz)

Inicio

$R \leftarrow \text{Raíz}$

Init(P)

Mientras( $R \neq \text{Nulo} \vee \neg \text{Vacía}(P)$ )

    Si ( $R \neq \text{Nulo}$ )

        Empilar(P, R)

$R \leftarrow R \rightarrow \text{HI}$

    Sino

        Depilar(P, R)

        Escribir  $R \rightarrow \text{Valor}$

$R \leftarrow R \rightarrow \text{HD}$

    FinSi

FinMientras

Fin

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## RECORRIDO PREORDEN

**Entrada:** Raíz (raíz del árbol)

**Salida:** recorrido preorden

**Precondición:** El árbol existe y es binario

Raíz apunta a la raíz del árbol

La pila P debe estar vacía

Se recorre el árbol, avanzando hasta su hijo de más a la izquierda y en el camino se va listando y empilando los nodos padres. Cuando se encuentra la hoja más izquierda, se depila el padre y se avanza por la rama derecha. En el subárbol derecho se aplica el mismo proceso.

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## Acción Recorrido\_Preorden (Raíz)

Inicio

$R \leftarrow \text{Raíz}$

Init(P)

Mientras( $R \neq \text{Nulo} \vee \neg \text{Vacía}(P)$ )

    Si ( $R \neq \text{Nulo}$ )

        Escribir  $R \rightarrow \text{Valor}$

        Empilar(P, R)

$R \leftarrow R \rightarrow \text{HI}$

    Sino

        Depilar(P, R)

$R \leftarrow R \rightarrow \text{HD}$

    FinSi

FinMientras

Fin

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## RECORRIDO POSTORDEN

**Entrada:** Raíz (raíz del árbol)

**Salida:** recorrido postorden

**Precondición:** El árbol existe y es binario

Raíz apunta a la raíz del árbol

La pila P debe estar vacía, constituido por dos campos: apuntador al nodo y un valor booleano indicando si desciende de un subárbol izquierdo o derecho.

### **Método:**

Se recorre el árbol, avanzando hasta su hijo de más a la izquierda y en el camino se va empilando los nodos padres y su valor booleano. Cuando se encuentra la hoja más izquierda, se depila el padre; si su valor booleano es derecho se lista y se pasa al nodo siguiente sobre la pila; si su valor booleano es izquierdo se avanza por la rama derecha empilando los nodos padres junto con su valor booleano.

# RECORRIDO DE ÁRBOLES BINARIOS EN FORMA NO RECURSIVA

## Acción Recorrido\_Postorden (Raíz)

$R \leftarrow \text{Raíz}$

Init(P)

Mientras( $R \neq \text{Nulo} \vee \neg \text{Vacía}(P)$ )

    Si ( $R \neq \text{Nulo}$ )

        Empilar(P, (R, IZQ))

$R \leftarrow R \rightarrow \text{HI}$

    Sino

        Depilar(P, (R, INDICADOR))

        Si (INDICADOR = IZQ)

            Empilar(P, (R, DER))

$R \leftarrow R \rightarrow \text{HD}$

        Sino

            Escribir  $R \rightarrow \text{Valor}$

$R \leftarrow \text{Nulo}$

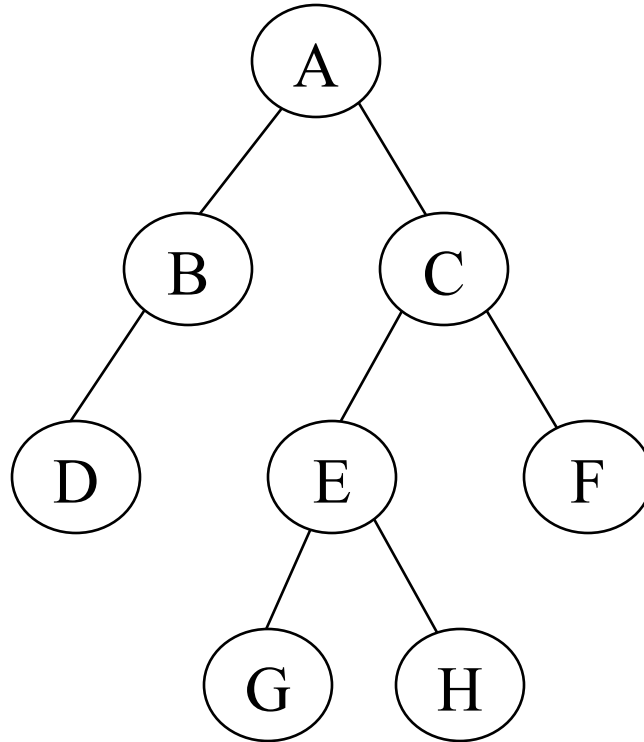
    FinSi

FinSi

FinMientras

# RECORRIDO POR NIVELES DE UN ÁRBOL BINARIO

El recorrido por niveles o por anchura de un árbol binario consiste en visitar primero la raíz, luego los nodos del nivel 1, después los nodos del nivel 2, etc., hasta visitar los nodos del último nivel. Cada nivel se visita de izquierda a derecha.



Recorrido: A, B, C, D, E, F, G, H



# ÁRBOL BINARIO EXTENDIDO

## Definición

Un árbol binario  $A$  se llama árbol-2 o árbol binario extendido, si cumple que:

Todo vértice tiene cero o dos hijos

Los nodos con dos hijos se denominan nodos internos.

Los nodos con cero hijos se denominan nodos externos u hojas

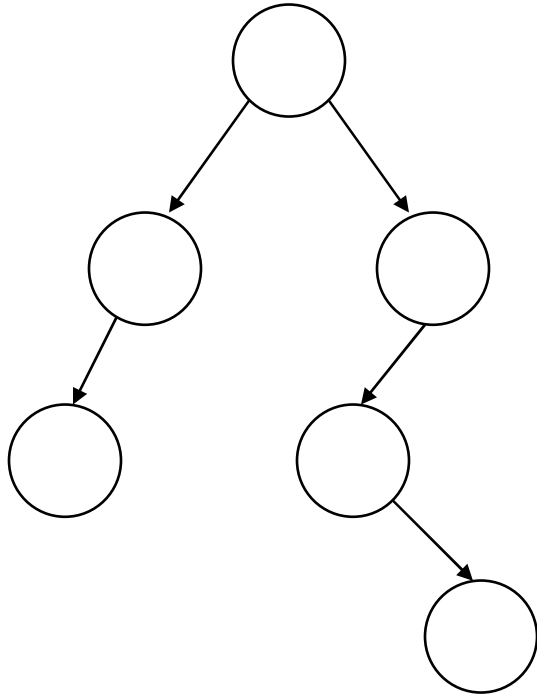
En un árbol-2, los nodos internos se grafican mediante círculos, y las hojas mediante cuadrados.

## Definición

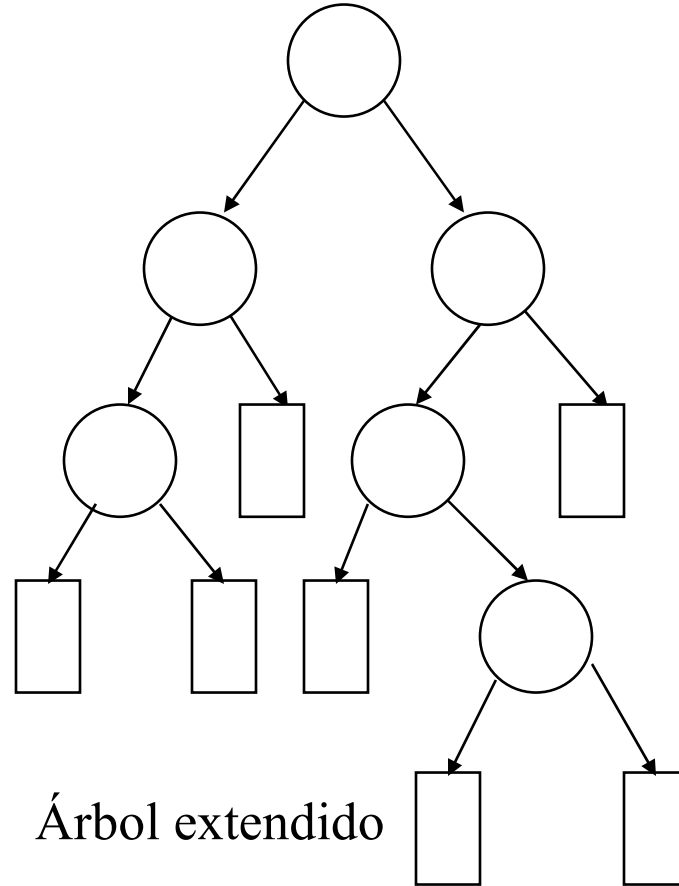
En un árbol extendido, el número de nodos externos es igual al número de nodos internos mas uno.

$$N_E = N_I + 1$$

# ÁRBOL BINARIO EXTENDIDO



Árbol binario



Árbol extendido

Un árbol binario puede convertirse a árbol-2 reemplazando todo árbol vacío por un nuevo nodo. Al extenderse un árbol binario, sus hojas se convierten en nodos internos, y los nodos añadidos se convierten en hojas. Los árboles extendidos se usan para representar árboles de expresiones aritméticas.

# ÁRBOLES DE EXPRESIONES

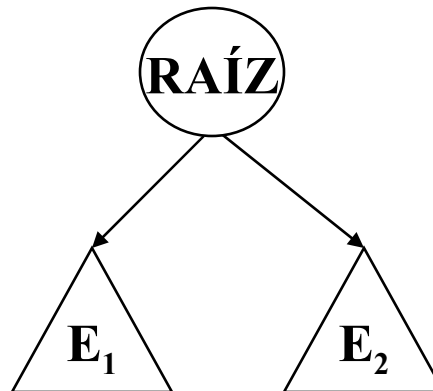
Un árbol ordenado puede usarse para representar una expresión aritmética. El recorrido inorden del árbol produce la expresión en notación infija. De igual forma los recorridos preorden y postorden producen la notación prefija y postfija respectivamente.

Un árbol de expresión es un árbol binario que cumple:

Los nodos hojas están etiquetados con operandos

Los nodos no hojas están etiquetados con operadores

La sintaxis abstracta describe una expresión construida mediante la aplicación de un operador binario **OP** a dos subexpresiones **E<sub>1</sub>** y **E<sub>2</sub>**

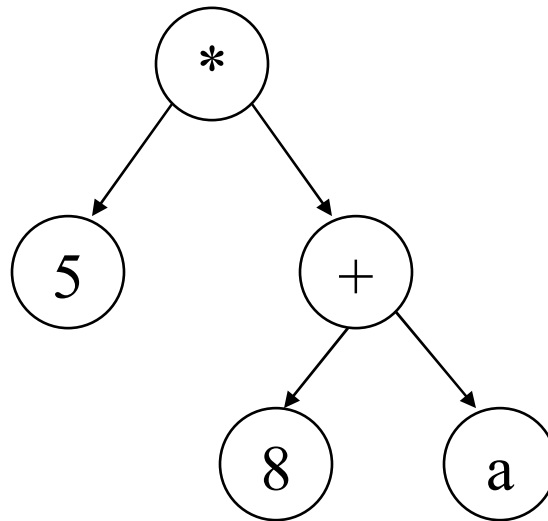


# ÁRBOLES DE EXPRESIONES

Sea  $E$  una expresión aritmética

Si  $\mathbf{OP} = +$ , entonces el valor de  $E$  es la suma de los valores de  $E_1$  y  $E_2$ . Si  $\mathbf{OP} = *$  entonces el valor de  $E$  es el producto de los valores de  $E_1$  y  $E_2$

Ejemplo:

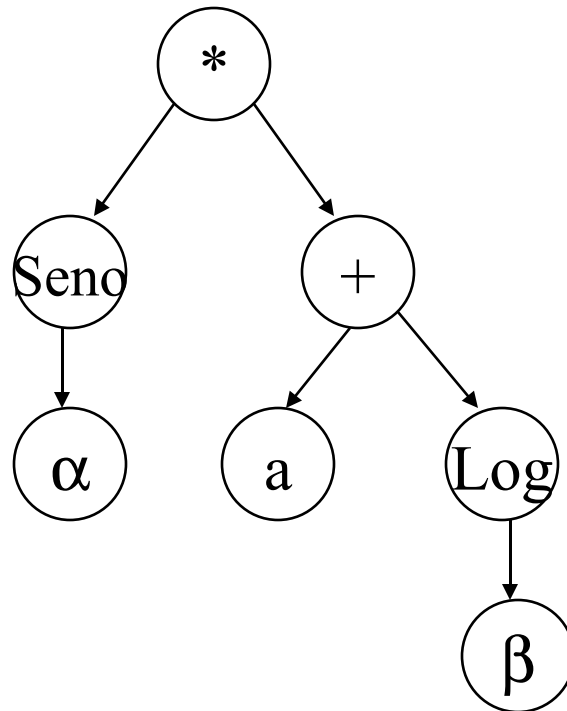


Pre-orden	$* 5 + 8 a$	notación prefija
In-orden	$5 * 8 + a$	notación infija
Post-orden	$5 8 a + *$	notación postfija

# ÁRBOLES DE EXPRESIONES

Aunque generalmente en una expresión los operadores son binarios. Un árbol de expresión puede representar también expresiones de operadores no binarios

Ejemplo:



Orden previo

\* Seno α + a Log β

Orden simétrico

α Seno \* a + β Log

Orden Posterior

α Seno a β Log + \*

# APLICACIÓN DE ÁRBOLES BINARIOS: EVALUACIÓN DE EXPRESIONES

Hasta ahora todas las estructuras han sido consideradas como estructuras homogéneas, ya que todos los elementos son del mismo tipo. Los árboles que contienen nodos de tipos diferentes se denominan heterogéneos y un ejemplo de ellos son los árboles de expresión en los que los nodos contienen en su parte valor un operador si son internos, y un operando si son externos.

Se presenta a continuación una aplicación para evaluar una expresión en notación postfija.

Registro Nodo

Booleano Tipo Nodo // Tipo de Nodo: operando u  
Union // operador

Carácter

ValorCarácter

Real

ValorNumérico

FinUnion

Nodo \*HI

Nodo \*HD

FinRegistro

Nodo \*Raíz // raíz del árbol

# APLICACIÓN DE ÁRBOLES BINARIOS: EVALUACIÓN DE EXPRESIONES

## Acción Evalúa(Raíz): Tipo Real

Inicio

Real      operando1, operando2

Carácter   operador

Si (Raíz  $\rightarrow$  TipoNodo = operando)

    Retornar (Raíz  $\rightarrow$  ValorNumérico)

FinSi

Operando1  $\leftarrow$  Evalúa(Raíz  $\rightarrow$  HI)

Operando2  $\leftarrow$  Evalúa(Raíz  $\rightarrow$  HD)

Operador  $\leftarrow$  Raíz  $\rightarrow$  ValorCarácter

Evalúa  $\leftarrow$  Operación(Operador, Operando1, Operando2)

Fin

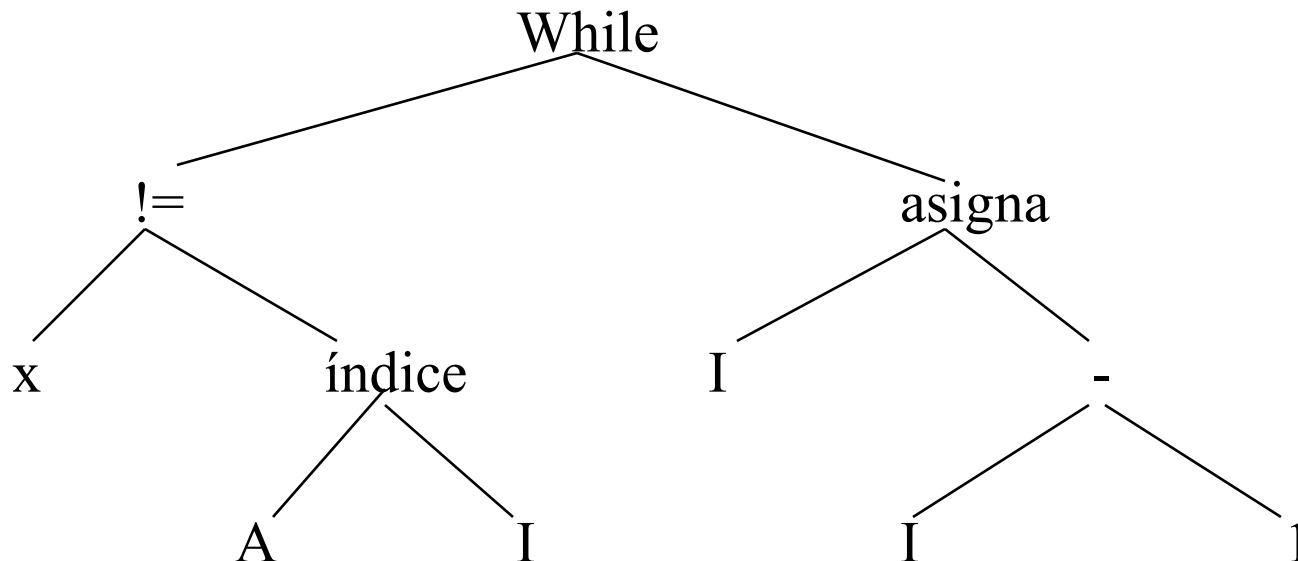
# ÁRBOLES DE EXPRESIONES

Considérese el siguiente fragmento en C

**While** (**X** **!=** **a**[**I**])

**I** = **I** – **1**;

El siguiente árbol muestra la sintaxis abstracta del fragmento. El **While** para el ciclo While, tiene dos hijos, que corresponden a la verificación de la expresión y al cuerpo del ciclo While respectivamente. Los otros operadores en el árbol son **!=** para verificar la desigualdad, **índice** para la indicación de arreglos, **asigna** para la asignación y **–** para la resta.





# CONSTRUCCIÓN DE UN ÁRBOL DE EXPRESIÓN

Considérese una expresión  $W$  en notación postfija. Presentamos un algoritmo para construir un árbol de expresión a partir de  $W$

**Entrada:**  $W$  expresión postfija

**Salida:** A Árbol de expresión

**Método:**

Mientras existan símbolos

1. Leer carácter  $C$  leer expresión postfija símbolo por símbolo

2. Si  $C$  es operando Crear un árbol de un nodo

Empilar en  $P$ , un apuntador  $p$  al nodo

Sino ( $C$  es operador) Depilar los dos últimos apuntadores

a árboles  $p_1$  y  $p_2$ ,

Construimos un árbol con raíz  $C$  (el operador)

y sus hijos izquierdo y derecho que apuntan a  $p_1$  y  $p_2$  respectivamente,

Empilamos  $p$ , el apuntador al nuevo árbol

3. En la pila quedará finalmente un apuntador al árbol de expresión

# CONSTRUCCIÓN DE UN ÁRBOL DE EXPRESIÓN

## Declaración de un nodo de un árbol binario

Registro Nodo

Inicio

T      Valor

Nodo \*HI    // apuntador al hijo izquierdo

Nodo \*HD    // apuntador al hijo derecho

FinRegistro

## Acción Construye\_Árbol\_de\_Expresión(P)

Inicio

Leer C

Mientras  $C \neq '\dagger'$  //  $'\dagger'$  usado como centinela

Si C es operando

$p \leftarrow$  nuevo Nodo

$p \rightarrow \text{Valor} \leftarrow C$

Empilar(P, p)

Sino

Depilar(P,  $p_2$ )

Depilar(P,  $p_1$ )

$p \leftarrow$  nuevo Nodo

$p \rightarrow \text{Valor} \leftarrow C$

$p \rightarrow \text{HI} \leftarrow p_1$

$p \rightarrow \text{HD} \leftarrow p_2$

Empilar (P, p)

FinSi

Leer C

FinMientras

Fin

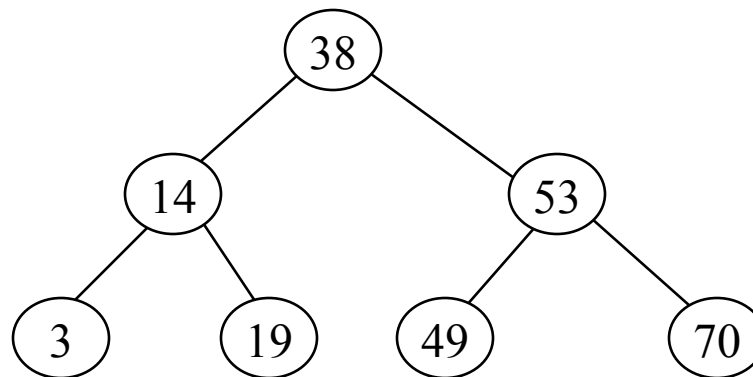
# ÁRBOL BINARIO DE BÚSQUEDA

Un árbol binario de búsqueda ABB (binary search tree), también llamado árbol binario ordenado, es aquel en el cual sus nodos están etiquetados con un valor de llave tomado de un conjunto E, estos elementos pueden ser complejos, pero por simplicidad supondremos que son enteros. En un árbol de búsqueda se cumple que:

Para todo nodo N:

- Todos los valores de los nodos del subárbol izquierdo son menores que el valor de llave de N.
- Todos los valores de los nodos del subárbol derecho son mayores que el valor de la llave de N.

Ejemplo:



# ÁRBOL BINARIO DE BÚSQUEDA

El árbol binario de búsqueda ABB permite buscar y encontrar un elemento, también permite insertar y borrar elementos fácilmente.

Esta estructura contrasta con los arreglos y las listas:

- **Arreglo unidimensional:** aquí se puede buscar y encontrar fácilmente, pero es costoso insertar y eliminar, porque se tienen que hacer desplazamientos a la izquierda o a la derecha del elemento a eliminar o insertar.
- **Listas enlazadas:** aquí se puede insertar y eliminar fácilmente, pero es costoso buscar y encontrar un elemento, ya que se debe usar una búsqueda secuencial.

La definición de árbol binario de búsqueda dada asume que todos los valores de los nodos son distintos. Existe una definición análoga de árbol binario de búsqueda que permite duplicados, esto es, en la que cada nodo  $N$  tiene la siguiente propiedad:

*El valor de  $N$  es mayor que cualquier valor del subárbol izquierdo de  $N$  y es menor o igual que cualquier valor del subárbol derecho de  $N$*

# ÁRBOL BINARIO DE BÚSQUEDA

## Búsqueda de un elemento

Supongamos que buscamos el valor  $V$  en un árbol ABB con raíz  $R$ . La acción Buscar debe devolver la dirección  $p$  (apuntador) al nodo que contiene  $V$ , o nulo si no se encuentra.

Si el árbol esta vacío (si  $R = \text{Nulo}$ ), se devuelve  $p = \text{Nulo}$ . Sino se compara el valor del nodo actual con  $V$ , si son iguales se devuelve en  $p$  la dirección del nodo. Si no es así hacemos una búsqueda bien en el lado izquierdo del nodo o bien en el lado derecho del nodo dependiendo de la relación del valor del nodo con  $V$ .

Consideremos un árbol cuyos nodos son registros tipo Nodo

Registro Nodo

Inicio

T	Valor
---	-------

Nodo	*HI	// apuntador al hijo izquierdo
------	-----	--------------------------------

Nodo	*HD	// apuntador al hijo derecho
------	-----	------------------------------

FinRegistro

# ÁRBOL BINARIO DE BÚSQUEDA

El siguiente algoritmo Buscar utiliza las siguientes variables:

Raíz :            apuntador al nodo raíz del árbol

Val:             Valor a buscar

Pos:             apuntador al nodo que contiene Val o nulo si no  
                  lo encuentra

PosPad:          apuntador al padre

Cuando se busca Val en el árbol ABB se presentan cuatro casos:

Caso1:          Pos = Nulo y PosPad = Nulo  
                  Árbol Nulo

Caso2:          Pos  $\neq$  Nulo Y PosPad = nulo  
                  Val se encuentra en el nodo raíz del árbol

Caso3:          Pos = Nulo y PosPad  $\neq$  Nulo  
                  Val no se encuentra, pero puede añadirse al árbol como  
                  hijo de PosPad

Caso4:          Pos  $\neq$  Nulo y PosPad  $\neq$  Nulo  
                  Val se encuentra en un nodo que no es raíz del árbol

## Acción Buscar(Raíz, Val, Pos, PosPad)

Inicio

Si (Raíz = Nulo)

Pos  $\leftarrow$  Nulo

PosPad  $\leftarrow$  Nulo

Sino

n  $\leftarrow$  Raíz

Si (n  $\rightarrow$  Valor = Val)

Pos  $\leftarrow$  Raíz

PosPad  $\leftarrow$  Nulo

Sino

Si (Val > n  $\rightarrow$  Valor)

n  $\leftarrow$  n  $\rightarrow$  HD

Sino

n  $\leftarrow$  n  $\rightarrow$  HI

FinSi

Padre  $\leftarrow$  Raíz

encontró  $\leftarrow$  falso

Mientras(n  $\neq$  Nulo  $\wedge$   $\neg$  encontró)

Si (n  $\rightarrow$  Valor = Val)

Pos  $\leftarrow$  n

PosPad  $\leftarrow$  Padre

encontró  $\leftarrow$  verdadero

Sino

Padre  $\leftarrow$  n

Si (Val < n  $\rightarrow$  Valor)

n  $\leftarrow$  n  $\rightarrow$  HI

Sino

n  $\leftarrow$  n  $\rightarrow$  HD

FinSi

FinSi

FinMientras

Si ( $\neg$  encontró)

Pos  $\leftarrow$  Nulo

PosPad  $\leftarrow$  Padre

FinSi

FinSi

Fin



La búsqueda en el árbol ABB puede realizarse en forma recursiva:

**Función BusRec(R, Val): Tipo P**

Inicio

Si (R = Nulo)

    Buscar  $\leftarrow$  Nulo

Sino

    Si (Val < R  $\rightarrow$  Valor)

        Buscar  $\leftarrow$  BusRec(R  $\rightarrow$  Hi, Val)

    Sino

        Si (Val > R  $\rightarrow$  Valor)

            Buscar  $\leftarrow$  BusRec(R  $\rightarrow$  HD, Val)

        Sino

            Buscar  $\leftarrow$  R

    FinSi

FinSi

FinSi

Retornar Buscar

Fin

# ÁRBOL BINARIO DE BÚSQUEDA

## INSERCIÓN DE UN ELEMENTO

Para insertar un elemento  $V$  en un árbol ABB, se procede en forma similar a la búsqueda de un elemento. Si  $V$  se encuentra, no se hace nada. Si no es así, se inserta  $V$  en el último espacio del camino recorrido.

## Acción Insertar(Raíz, Val)

Inicio

Pos  $\leftarrow$  Nulo

PosPad  $\leftarrow$  Nulo

Buscar(Raíz, Val, Pos, PosPad)

Si (Pos  $\neq$  Nulo)

    Escribir Val, “ ya existe”

Sino

    n  $\leftarrow$  nuevo Nodo

    n  $\rightarrow$  Valor  $\leftarrow$  Val

    n  $\rightarrow$  HI  $\leftarrow$  Nulo

    n  $\rightarrow$  HD  $\leftarrow$  Nulo

    Si (PosPad = Nulo)

        Raíz  $\leftarrow$  n

    Sino

        Si (Val < PosPad  $\rightarrow$  Valor)

            PosPad  $\rightarrow$  HI  $\leftarrow$  n

        Sino

            PosPad  $\rightarrow$  HD  $\leftarrow$  n

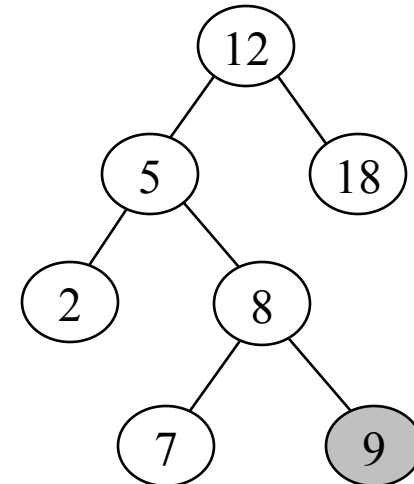
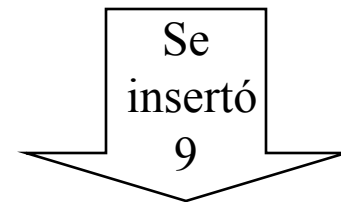
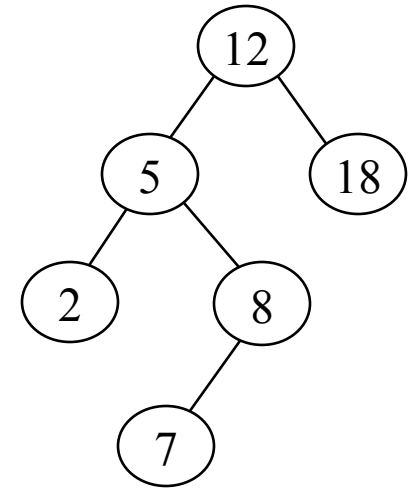
        FinSi

    FinSi

FinSi

Fin

Ejemplo:



**Acción InsertarRec(R, Val)** // algoritmo recursivo de la inserción

Inicio

Si (R = Nulo)

n  $\leftarrow$  nuevo nodo

n  $\rightarrow$  Valor  $\leftarrow$  Val

n  $\rightarrow$  HI  $\leftarrow$  Nulo

n  $\rightarrow$  HD  $\leftarrow$  Nulo

R  $\leftarrow$  n

Sino

Si (Val < R  $\rightarrow$  Valor)

InsertarRec (R  $\rightarrow$  HI, Val)

Sino

Si (Val > R  $\rightarrow$  Valor)

InsertarRec(R  $\rightarrow$  HD, Val)

Sino

Escribir Val, “ ya existe”

FinSi

FinSi

FinSi

Fin

# ÁRBOL BINARIO DE BÚSQUEDA

## ELIMINACIÓN DE UN ELEMENTO

Cuando se quiere eliminar un elemento, se presentan tres casos:

- Si el nodo es hoja, se elimina de inmediato.
- Si el nodo tiene un hijo, el nodo se puede eliminar haciendo un ajuste en un apuntador del padre, para saltar el nodo, modificando el apuntador del padre para que apunte al nodo hijo del nodo a eliminar.
- El caso complicado se da cuando el nodo a eliminar tiene dos hijos. En este caso, la estrategia consiste en sustituir el nodo a eliminar por el nodo con el valor más pequeño del subárbol derecho, y eliminar este nodo.

# ÁRBOL BINARIO DE BÚSQUEDA

## Acción Eliminar()

Inicio

    Escribir “Ingrese valor a eliminar ”

    Leer Val

**Buscar(Raíz, Val, Pos, PosPad)**

    Si (Pos = Nulo)

        Escribir Val, “ no existe en el árbol”

    Sino

        Si ( $\text{Pos} \rightarrow \text{HI} \neq \text{Nulo} \wedge \text{Pos} \rightarrow \text{HD} \neq \text{Nulo}$ )

**Caso\_B(Raíz, Val, Pos, PosPad)**

        Sino

**Caso\_A(Raíz, Val, Pos, PosPad)**

        FinSi

    FinSi

Fin

# ÁRBOL BINARIO DE BÚSQUEDA

## Caso\_A

Elimina el nodo N de la posición Pos, donde N no tiene dos hijos. El puntero PosPad da la posición del padre de N o si PosPad = Nulo, es que N es el nodo Raíz. El puntero Hijo da la posición del único hijo de N, o si Hijo = Nulo es que N no tiene hijos.

## Acción Caso\_A(Raíz, Val, Pos, PosPad)

Inicio

Si ( $\text{Pos} \rightarrow \text{Hi} = \text{Nulo} \wedge \text{Pos} \rightarrow \text{HD} = \text{Nulo}$ )

Hijo  $\leftarrow$  Nulo

Sino

Si ( $\text{Pos} \rightarrow \text{HI} \neq \text{Nulo}$ )

Hijo  $\leftarrow \text{Pos} \rightarrow \text{HI}$

Sino

Hijo  $\leftarrow \text{Pos} \rightarrow \text{HD}$

FinSi

FinSi

Si ( $\text{PosPad} \neq \text{Nulo}$ )

Si ( $\text{Pos} = \text{PosPad} \rightarrow \text{HI}$ )

$\text{PosPad} \rightarrow \text{Hi} \leftarrow \text{Hijo}$

Sino

$\text{PosPad} \rightarrow \text{HD} \leftarrow \text{Hijo}$

FinSi

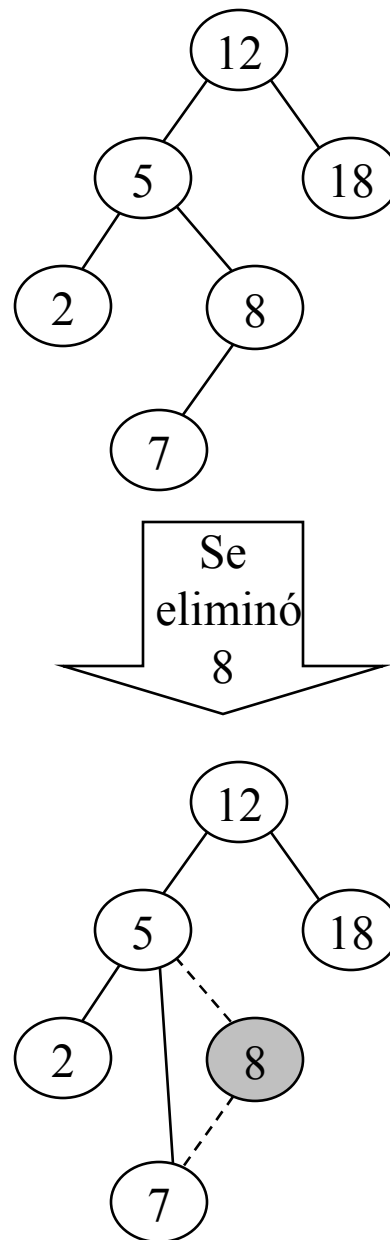
Sino

$\text{Raiz} \leftarrow \text{Hijo}$

FinSi

Fin

Ejemplo:





# ÁRBOL BINARIO DE BÚSQUEDA

## Caso\_B

Elimina el nodo N de la posición Pos, donde N tiene dos hijos. El puntero PosPad da la posición del padre de N o si PosPad = Nulo, es que N es el nodo Raíz. El puntero Suc da la posición del sucesor inorden de N y PadSuc da la posición del padre del sucesor inorden.

## Acción Caso\_B(Raíz, Val, Pos, PosPad)

Inicio

$T \leftarrow \text{Pos}$

$n \leftarrow \text{Pos} \rightarrow \text{HD}$

Mientras ( $n \rightarrow \text{HI} \neq \text{Nulo}$ )

$T \leftarrow n$

$n \leftarrow n \rightarrow \text{HI}$

FinMientras

$\text{Suc} \leftarrow n$

$\text{PadSuc} \leftarrow T$

**Caso\_A(Raíz, Val, Suc, PadSuc)**

Si ( $\text{PosPad} \neq \text{Nulo}$ )

Si ( $\text{Pos} = \text{PosPad} \rightarrow \text{HI}$ )

$\text{PosPad} \rightarrow \text{HI} \leftarrow \text{Suc}$

Sino

$\text{PosPad} \rightarrow \text{HD} \leftarrow \text{Suc}$

FinSi

Sino

$\text{Raíz} \leftarrow \text{Suc}$

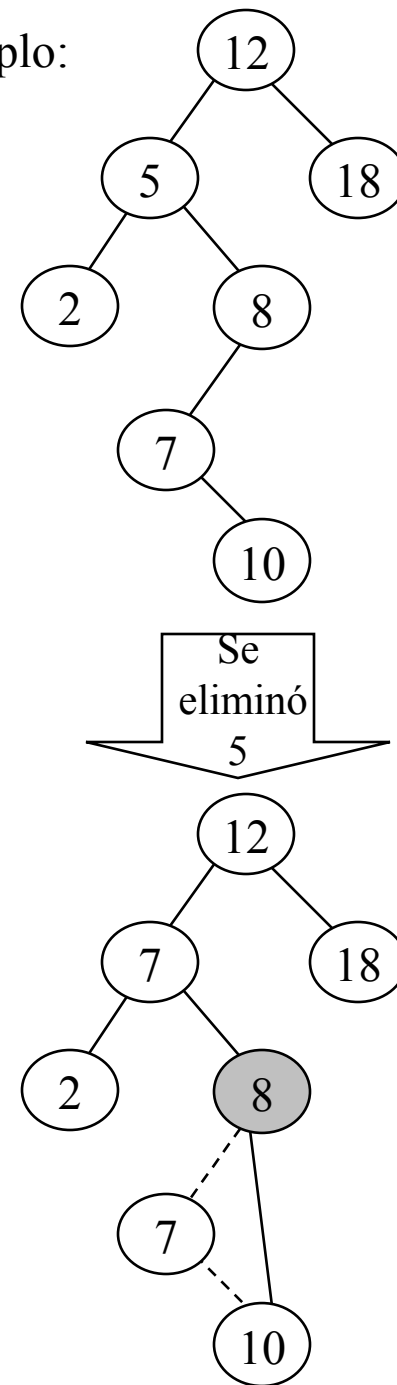
FinSi

$\text{Suc} \rightarrow \text{HI} \leftarrow \text{Pos} \rightarrow \text{HI}$

$\text{Suc} \rightarrow \text{HD} \leftarrow \text{Pos} \rightarrow \text{HD}$

Fin

Ejemplo:



# ÁRBOLES ENHEBRADOS

En un recorrido inorden, la pila contiene la dirección del siguiente elemento inorden en el recorrido. Consideremos como alternativa que cada nodo con un subárbol derecho nulo, tiene en el apuntador hijo derecho, la dirección al sucesor inorden, esto haría que sea innecesario sacar de la pila la dirección del siguiente elemento en el recorrido. A este tipo de apuntadores se le conoce como hebra, y al árbol que contiene hebras se le denomina enhebrado. Este tipo de árbol requiere que cada nodo contenga además de dos punteros a los hijos, un campo hebra para indicar si su apuntador derecho es o no una hebra.

Un árbol enhebrado (Threaded tree) se basa en la propiedad de que una representación de árbol basada en apuntadores desaprovecha muchos apuntadores que no apuntan a ningún nodo.

Cuando un nodo no tiene hijo derecho, se sustituye su valor de apuntador derecho por su sucesor inorden, y cuando no tiene hijo izquierdo por su predecesor inorden.

# ÁRBOLES ENHEBRADOS

Consideremos un árbol representado por nodos definidos de la siguiente forma

Registro Nodo

T	Valor
Nodo	*HI
Nodo	*HD
Booleano	Hebra

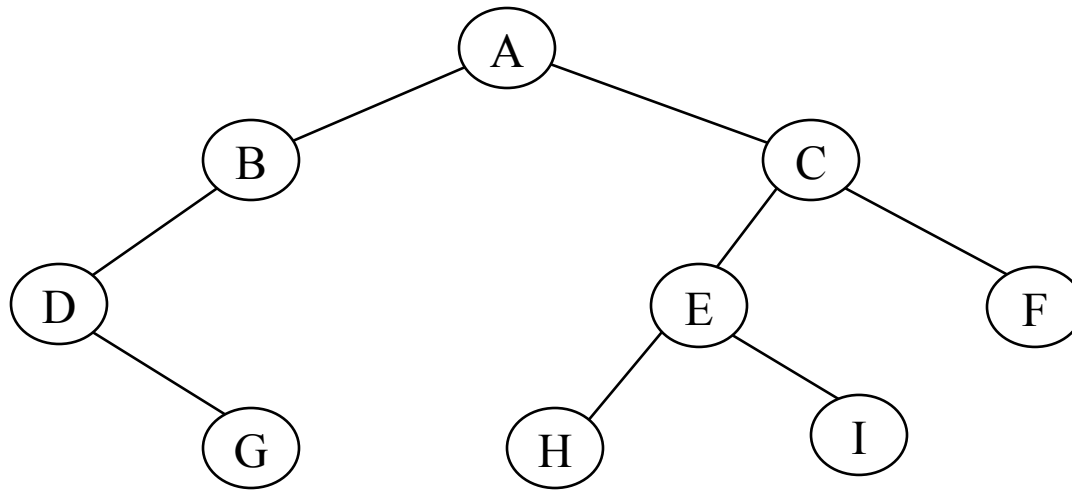
FinRegistro

Hebra es verdadero si HD es nulo o una hebra diferente de nulo, falso en otro caso.

Nodo            \*Raíz

El booleano hebra es utilizado para indicar si HI o HD es un puntero al hijo izquierdo o al hijo derecho, o si son apuntadores (hebra) al predecesor inorden o al sucesor inorden respectivamente.

# ÁRBOLES ENHEBRADOS



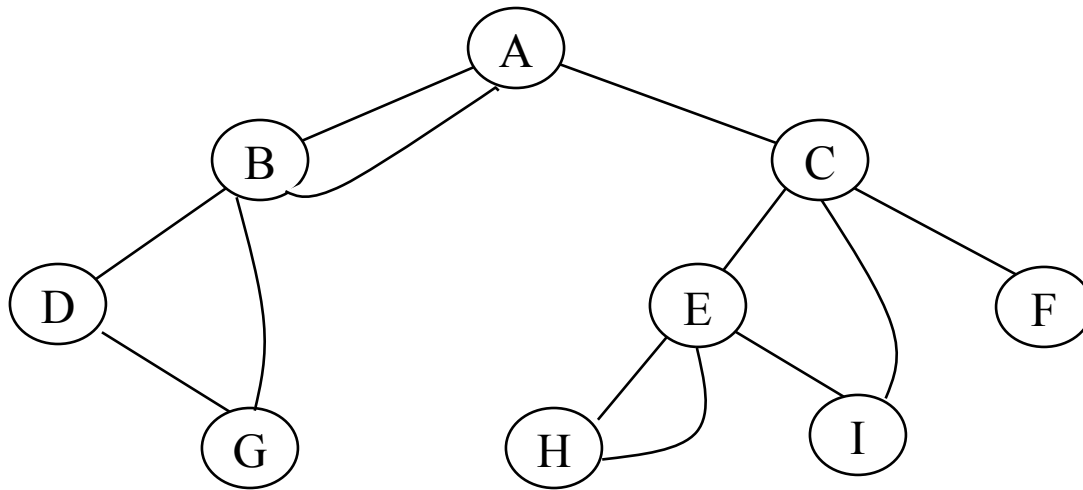
El árbol de momento 9 tiene 18 apuntadores, de los cuales 10 son nulos (más de la mitad).

Estos apuntadores pueden reciclarse, utilizándose como apuntadores al sucesor inorden

# ÁRBOLES ENHEBRADOS A LA DERECHA

Un árbol enhebrado a la derecha contiene hebras que apuntan al sucesor inorden derecho.

Los nodos que no tienen hijo derecho, reemplazan sus punteros por un apuntador al sucesor inorden.



Hebras que apuntan al sucesor inorden.

Se presenta la acción recorrido inorden no recursivo para el árbol enhebrado

### **Acción Recorrido(Raíz)**

Inicio

$R \leftarrow \text{Raíz}$

Repetir

$q \leftarrow \text{Nulo}$

Mientras( $R \neq \text{Nulo}$ )

$q \leftarrow R$

$R \leftarrow R \rightarrow \text{HI}$

FinMientras

Si ( $q \neq \text{Nulo}$ )

Escribir  $q \rightarrow \text{Valor}$

$R \leftarrow q \rightarrow \text{HD}$

Mientras ( $R \neq \text{Nulo} \wedge q \rightarrow \text{Hebra}$ )

Escribir  $R \rightarrow \text{Valor}$

$q \leftarrow R$

$R \leftarrow R \rightarrow \text{HD}$

FinMientras

FinSi

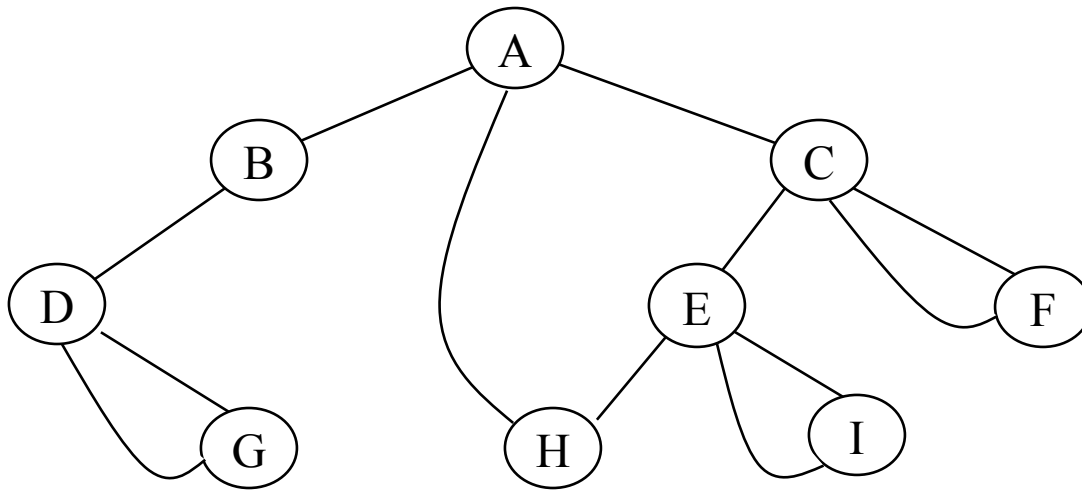
Mientras( $q \neq \text{Nulo}$ )

Fin

# ÁRBOLES ENHEBRADOS A LA IZQUIERDA

Un árbol enhebrado a la izquierda contiene hebras que apuntan al antecesor inorden izquierdo.

Los nodos que no tienen hijo izquierdo, reemplazan sus punteros por un apuntador al nodo antecesor inorden.



Hebras que apuntan al antecesor inorden.



# **ÁRBOLES PREHEBRADOS Y POSTHEBRADOS**

## **ÁRBOLES PREHEBRADOS**

Un árbol prehebrado puede definirse de manera similar a los árboles enhebrados. La diferencia radica en que los apuntadores derechos e izquierdos de nodos con valor Nulo son sustituidos respectivamente por los sucesores y antecesores preorden del nodo correspondiente, en cuyo caso se llaman prehebrado a la izquierda o prehebrado a la derecha respectivamente.

## **ÁRBOLES POSTHEBRADOS**

De manera similar en un árbol posthebrado los apuntadores derechos e izquierdos de nodos con valor Nulo son sustituidos respectivamente por los sucesores y antecesores postorden del nodo correspondiente, en cuyo caso se llamarán posthebrado a la izquierda o posthebrado a la derecha respectivamente.

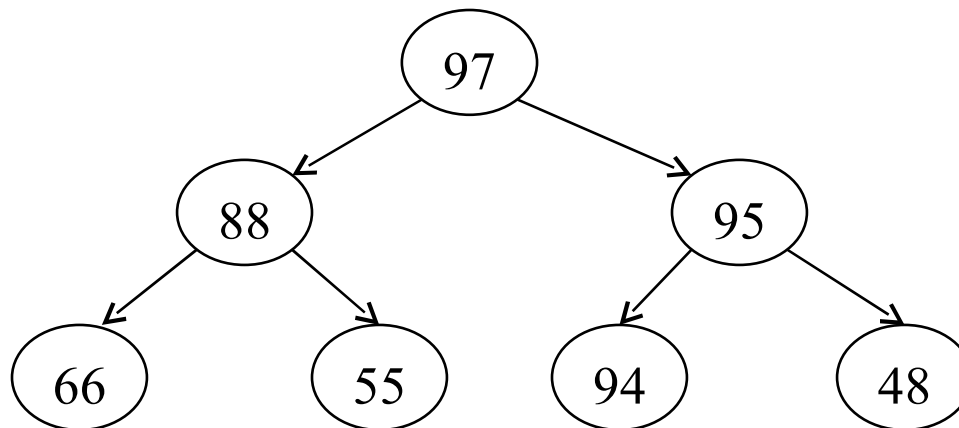
# ÁRBOL EN MONTÓN

El árbol en montón se usa en un algoritmo de ordenación llamado *ordenación por montón*. Este algoritmo mejora el tiempo de ejecución en el peor caso comparado con el método de ordenación rápida.

## Definición

Un árbol binario  $A$  se dice en montón si:

- a)  $A$  es completo
- b) El valor de cada nodo de  $A$  es mayor que los valores de cualquiera de sus hijos.



# ÁRBOL EN MONTÓN

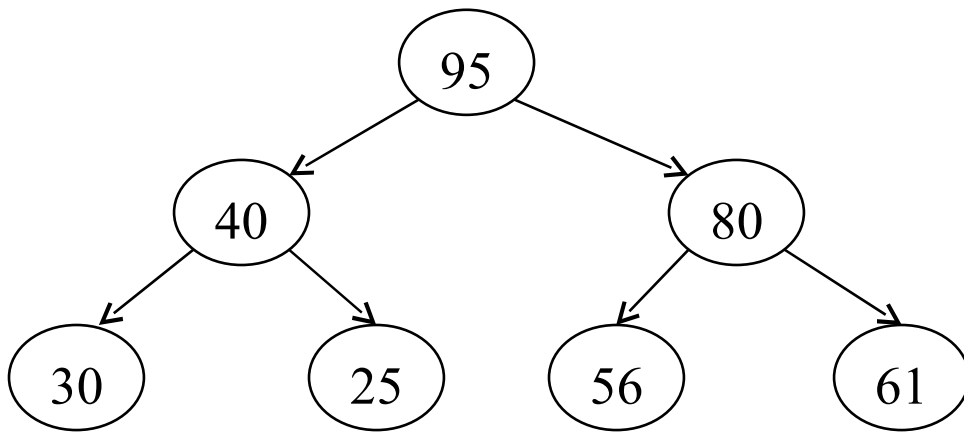
## **Inserción en un árbol en montón**

**Entrada:** Lista de elementos.

**Salida:** árbol en montón.

Mientras existan elementos  $V$  en la entrada se procederá de la siguiente forma:

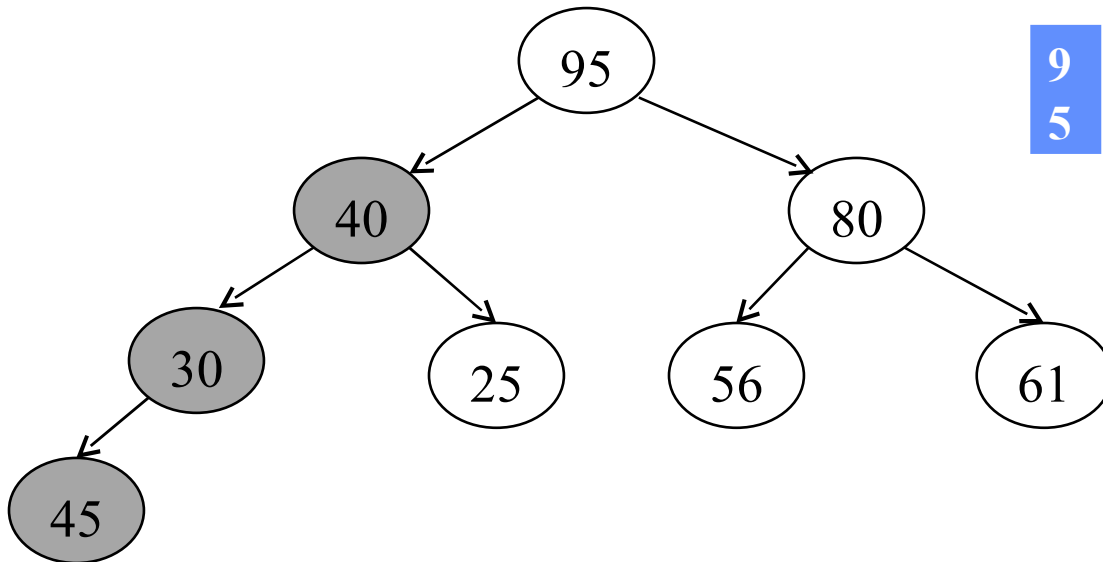
1. Se adiciona  $V$  al final de  $M$  de forma que  $M$  sigue siendo completo, aunque no necesariamente sea un árbol en montón.
2. Si al adicionar  $V$  al árbol, no resulta de montón, se hace subir  $V$  hasta su lugar apropiado en  $M$  hasta que  $M$  sea finalmente de montón.



9	4	8	3	2	5	6			
5	0	0	0	5	6	1			

Queremos añadir 45

a) Aumentamos 45 en la posición  $k = 8$



9	4	8	3	2	5	6	4		
5	0	0	0	5	6	1	5		

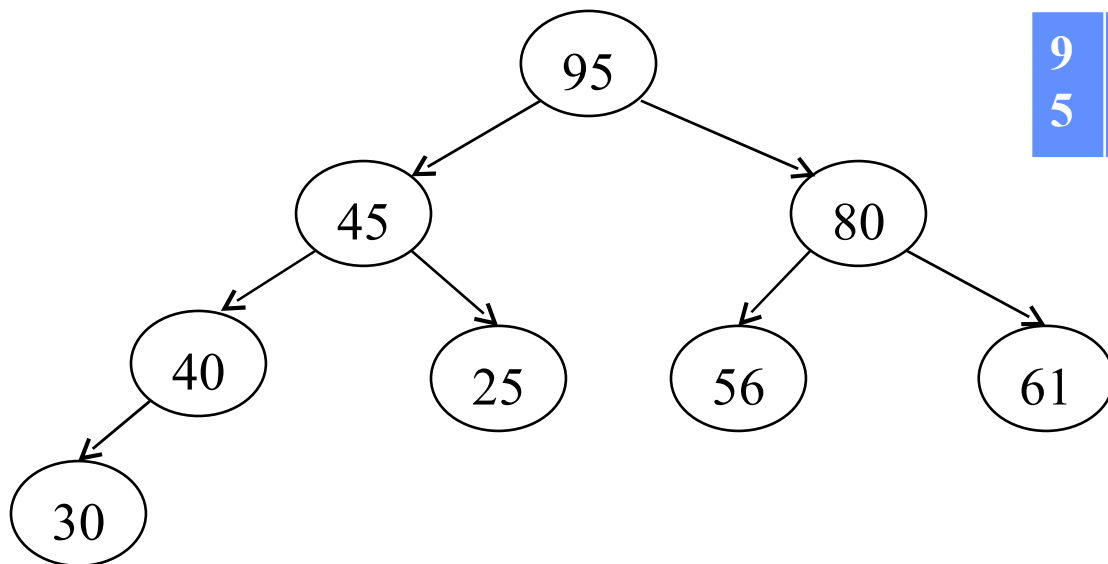
b) Comparamos  $A(k) = 45$  con su padre que se encuentra en  $A(k/2) = A(4) = 30$

c) Como  $30 < 45$  los cambiamos, así 45 aparece como padre de 30 y como hijo de 40.

d) Como  $45 > 40$  los cambiamos y 45 queda como padre de 40.

Finalmente queda

9	4	8	4	2	5	6	3		
5	5	0	0	5	6	1	0		



# ÁRBOL EN MONTÓN

## **Inserción en un árbol en montón**

El árbol en montón con  $N$  elementos está guardado en el vector  $M$ . La acción `Inserción_Montón` inserta  $V$  como un nuevo elemento del árbol.  $N$  se incrementa en 1.  $P$  da la posición de  $V$  a medida que sube por el árbol y  $PAD$  indica la posición del padre de  $V$ .

## **Acción Inserción\_Montón(M,N,V)**

Inicio

$N \leftarrow N + 1$

$P \leftarrow N$

Montón  $\leftarrow$  falso

Mientras ( $P > 1 \wedge \neg \text{Montón}$ )

$\text{Pad} \leftarrow P/2$

    Si ( $V \leq M(\text{Pad})$ )

$M(P) \leftarrow V$

        Montón  $\leftarrow$  verdad

    Sino

$M(P) \leftarrow M(\text{Pad})$

$P \leftarrow \text{Pad}$

    FinSi

FinMientras

If ( $\neg \text{Montón}$ )

$M(1) \leftarrow V$

FinSi

Fin

# ÁRBOL EN MONTÓN

## **Eliminación de la raíz en un árbol en montón**

**Entrada:** árbol en montón  $M$  con  $N$  elementos

**Salida:** árbol en montón  $M$  con  $N-1$  elementos

$V = M(1)$  raíz del árbol  $M$

## **Procedimiento:**

Para eliminar la raíz en un árbol en montón  $M$ :

1. Asignamos la raíz  $M(1)$  a la variable  $V$
2. Reemplazamos el nodo raíz a eliminar con el último nodo que fue adicionado en  $M$ , de modo que  $M$  sigue siendo completo, aunque no necesariamente un árbol en montón.
3. Reamontonamos, haciendo que la nueva raíz se mueva a su lugar adecuado para que  $M$  sea finalmente un árbol en montón.



## Acción Elimina\_Raíz(M,N,V)

HD  $\leftarrow$  2\*P+1

Inicio

V  $\leftarrow$  M(1)

U  $\leftarrow$  M(N)

N  $\leftarrow$  N - 1

P  $\leftarrow$  1

HI  $\leftarrow$  2

HD  $\leftarrow$  3

Montón  $\leftarrow$  falso

Mientras (HD  $\leq$  N  $\wedge$   $\neg$ Montón)

    Si (U  $\geq$  M(HI)  $\wedge$  U  $\geq$  M(HD))

        M(P)  $\leftarrow$  U

        Montón  $\leftarrow$  verdad

    Sino

        Si (M(HD)  $\leq$  M(HI))

            M(P)  $\leftarrow$  M(HI)

            P  $\leftarrow$  HI

        Sino

            M(P)  $\leftarrow$  M(HD)

            P  $\leftarrow$  HD

    FinSi

    HI  $\leftarrow$  2\*P

FinSi

FinMientras

Si ( $\neg$ Montón)

    Si(HI = N  $\wedge$  U < M(HI))

        M(P)  $\leftarrow$  M(HI)

        P  $\leftarrow$  HI

    FinSi

    M(P)  $\leftarrow$  U

FinSi

Fin

# ÁRBOL EN MONTÓN

## CREACIÓN DE UN ÁRBOL EN MONTÓN

Se tiene una fila  $F$  de enteros. Se quiere construir una aplicación para crear un árbol en montón teniendo como entrada la fila  $F$  de enteros.

**Entrada:**  $F$  fila secuencial no vacía

**Salida:**  $M$  árbol en montón

### Procedimiento:

1.  $\text{Primer}(F)$  abre la fila  $F$
2.  $\text{Tomar}(F, V)$  toma el primer elemento de  $F$  y lo guarda en  $V$ .  
Luego asignar  $V$  a  $M(1)$
3.  $\text{Tomar}(F, V)$  toma el siguiente elemento de  $F$  y lo guarda en  $V$
4. El ciclo Mientras se ejecuta mientras existan elementos en la fila. El ciclo consiste en:

Insertar el elemento tomado de  $F$  en el árbol en Montón  $M$

Tomar el siguiente elemento de  $F$ .

# ÁRBOL EN MONTÓN

**Acción Crea\_Árbol\_Montón(F, M)**

Inicio

Primer(F)

Tomar(F,V)

$N \leftarrow 1$

$M(N) \leftarrow V$

Tomar(F,V)

Mientras( $\neg \text{Último}(F)$ )

**Inserción\_Montón(M, N, V)**

// inserta V en el vector M con N elementos

Tomar(F, V)

FinMientras

Fin

# ÁRBOL EN MONTÓN

## ORDENAMIENTO EN MONTÓN

Sea  $M$  un árbol en montón de  $N$  elementos, lo que quiere decir que el nodo raíz corresponde al mayor valor del árbol. Si extraemos la raíz y la colocamos en una lista, y reemplazamos la raíz por el último nodo, es posible que el árbol resultante no sea un montón, en este caso lo reamontonamos para convertirlo en un árbol en montón.

Nuevamente eliminamos la raíz y la adicionamos en la lista, y así proseguimos hasta que el árbol este vacío, y todos sus elementos se hayan pasado a la lista en orden descendente.

# ÁRBOL EN MONTÓN

## ORDENAMIENTO EN MONTÓN

**Entrada:** árbol en montón

**Salida:** lista de elementos en orden descendente.

### **Procedimiento:**

Mientras el árbol en montón no sea vacío, repetir los pasos:

- 1 Eliminar la raíz, y colocar el elemento en la lista
- 2 Reemplazar la raíz con el último nodo del árbol. Si el árbol resultante no es un montón, entonces se reamontonará para convertirlo en un montón, luego se continuará en el paso 1.

### **Acción Ordena\_Montón(M, N)**

Inicio

Cab  $\leftarrow$  Nulo

Mientras (N > 0)

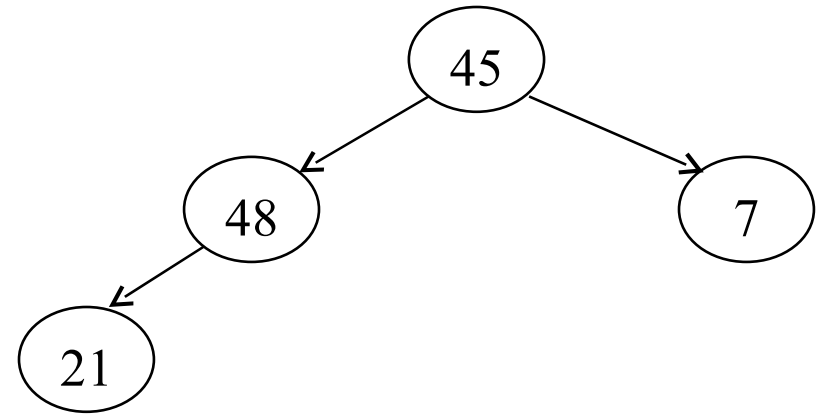
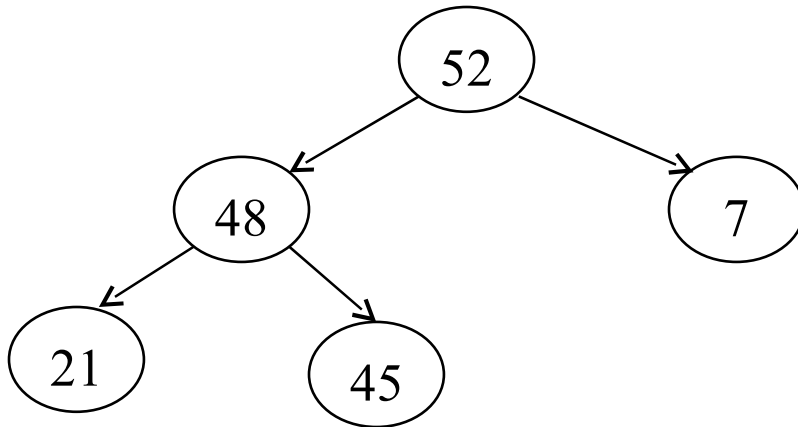
**Elimina\_Raíz(M, N, V)**

Adiciona(Cab, V, U) // adiciona V en la lista

FinMientras

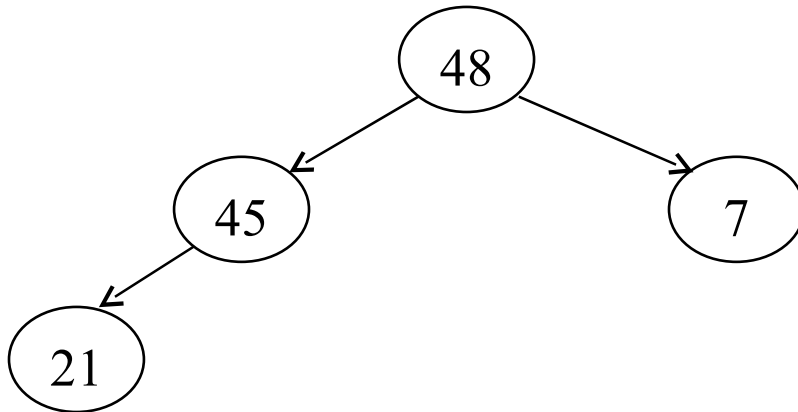
Fin

Eliminamos la raíz : 52

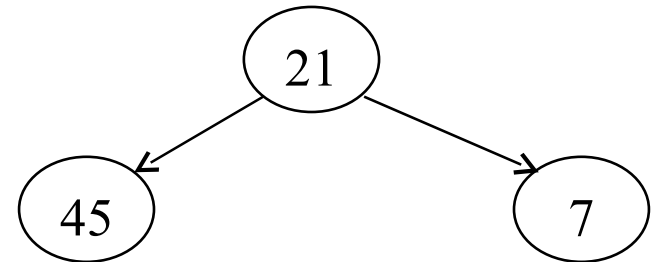


L: 52

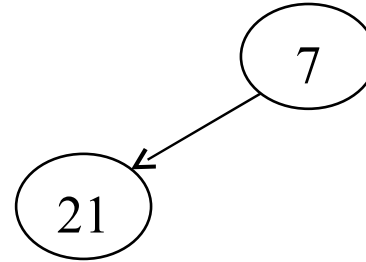
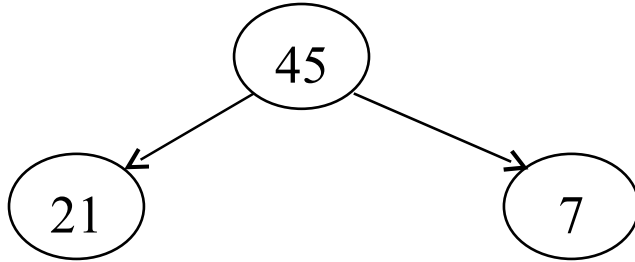
Eliminamos la raíz : 48



L: 52, 48

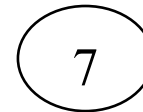
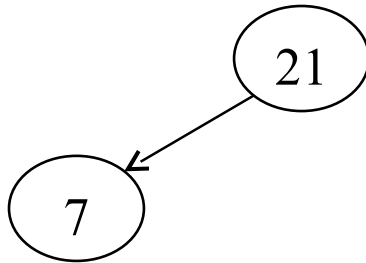


Eliminamos la raíz : 45



L: 52, 48, 45

Eliminamos la raíz : 21



L: 52, 48, 45, 21

Eliminamos la raíz : 7

L: 52, 48, 45, 21, 7

# ÁRBOL EN MONTÓN

## COMPLEJIDAD DE LA ORDENACION POR MONTÓN

En la construcción del árbol en montón  $M$ , el número de comparaciones para encontrar el sitio adecuado para un nuevo elemento  $V$  no puede exceder a la profundidad del árbol. Como  $M$  es un árbol completo, su profundidad está limitada por  $\log_2 m$ , donde  $m$  es el número de elementos del árbol  $M$ . De este modo, el número de comparaciones  $g(n)$  para insertar los  $n$  elementos del árbol está limitado por:

$$g(n) \leq n \log_2 n$$

Para eliminar los  $n$  elementos del árbol  $M$ , dado que el árbol es completo con  $m$  elementos y que los subárboles izquierdo y derecho de  $M$  son montones, se elimina la raíz y para reamontonar se efectúan cuatro comparaciones para mover el nuevo nodo raíz (el último del árbol) un paso abajo en el árbol. Como la profundidad del árbol no excede a  $\log_2 m$ , al reamontonar se efectúan a lo mas  $4 \log_2 m$  comparaciones para encontrar el lugar adecuado del nuevo nodo raíz. Si  $h(n)$  es el número total de comparaciones para eliminar los  $n$  elementos de  $M$ , lo que requiere reamontonar  $n$  veces, está limitado por:

$$h(n) \leq 4n \log_2 n$$

El tiempo de ejecución para la ordenación es proporcional a  $n \log_2 n$ , o sea,  $f(n) = O(n \log_2 n)$ . El tiempo de ejecución de la ordenación por el método de la burbuja es  $O(n^2)$ . El tiempo medio de ejecución de la ordenación rápida es  $O(n \log_2 n)$  como la del montón, pero para el peor caso es proporcional a  $O(n^2)$ .



# ÁRBOL DE HUFFMAN

Recordemos que un árbol binario extendido o árbol-2 es un árbol binario en el que cada nodo tiene 0 o 2 hijos. Los nodos con 0 hijos se llaman nodos externos y los nodos con 2 hijos se llaman nodos internos. Los nodos internos están representados por círculos y los nodos externos están representados por cuadrados.

En cualquier árbol-2 el número  $N_E$  de nodos externos es el número  $N_I$  de nodos internos mas 1; o sea,  $N_E = N_I + 1$

**La longitud de camino externo  $L_E$**  de un árbol-2 es la suma de todas las longitudes de camino obtenidas sobre cada camino desde la raíz  $R$  hasta un nodo externo.

**La longitud de camino interno  $L_I$**  de un árbol-2 es la suma de todas las longitudes de camino obtenidas sobre cada camino desde la raíz  $R$  hasta un nodo interno.

Para cualquier árbol-2 con  $n$  nodos internos

$$L_E = L_I + 2n$$

# ÁRBOL DE HUFFMAN

Suponga que  $T$  es un árbol-2 con  $n$  nodos externos y suponga que cada nodo externo tiene asignado un peso (no negativo).

**La longitud de camino con peso (externo)  $P$**  del árbol  $T$  se define como la suma de las longitudes de camino con sus pesos

$$P = W_1 L_1 + W_2 L_2 + \dots + W_n L_n$$

donde  $W_i$  y  $L_i$  denotan respectivamente, el peso y la longitud del camino del nodo externo  $N_i$ .

Si consideramos la colección de árboles-2 con  $n$  nodos externos y a cada árbol-2 se le dan los mismos  $n$  pesos para sus nodos externos. Entonces no está claro que árbol tenga la mínima longitud de camino con peso.

El problema general que queremos resolver es el siguiente:

Suponga una lista de  $n$  pesos dada:

$$W_1, W_2, \dots, W_n$$

Entre todos los árboles-2 con  $n$  nodos externos y con los  $n$  pesos dado, encontrar un árbol  $T$  con una longitud de camino con peso mínimo. Huffman dio un algoritmo para encontrar ese árbol  $T$ .

# ÁRBOL DE HUFFMAN

## Algoritmo de Huffman

Se define recursivamente en función del número de pesos. La solución para un solo peso es simplemente el árbol con un solo nodo.

**Entrada:** S secuencia de nodos

**Salida:** A árbol Huffman

**Método:**

Repetir hasta que S contenga un solo nodo

1. Elegir de S dos nodos  $n_1$  y  $n_2$  con menor peso  $w_1$  y  $w_2$  respectivamente.
2. Eliminar  $n_1$  y  $n_2$  de S.
3. Adicionar un nuevo nodo con peso  $w = w_1 + w_2$

# ÁRBOL DE HUFFMAN

## Acción Huffman(S, n)

Inicio

Si  $n = 2$

Elegir  $n_1$  con peso  $w_1$

Elegir  $n_2$  con peso  $w_2$

$T \leftarrow \text{construye}(w_1, w_2)$

Retornar(T)

Sino

Elegir nodos  $n_i, n_j$  con menor peso  $w_i$  y  $w_j$

$X \leftarrow n_i$

$Y \leftarrow n_j$

Adicionar nodo Z con peso  $w = w_i + w_j$

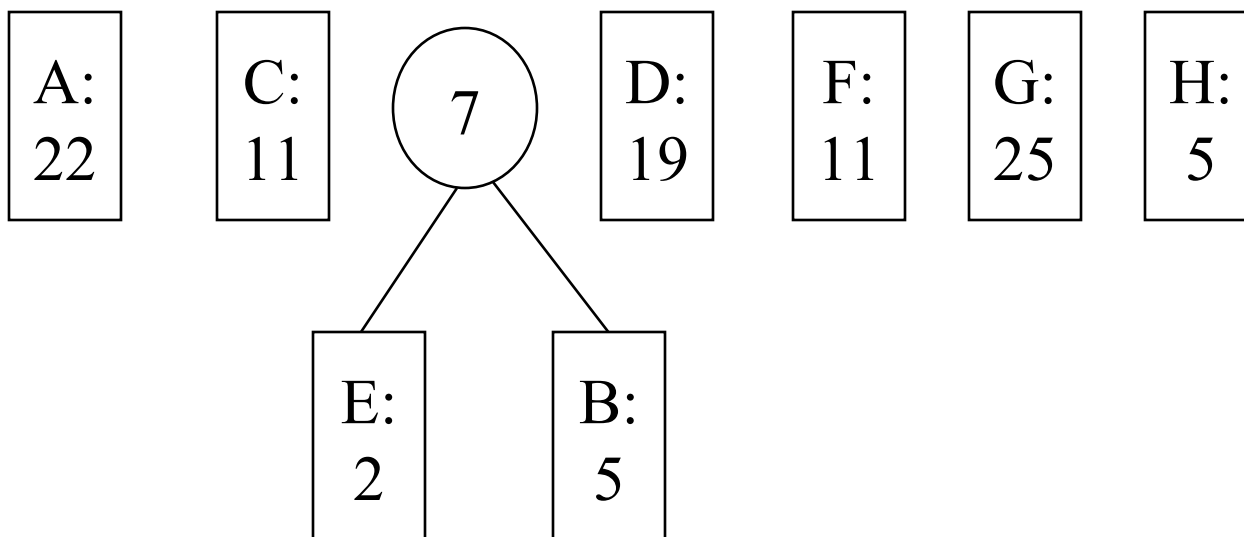
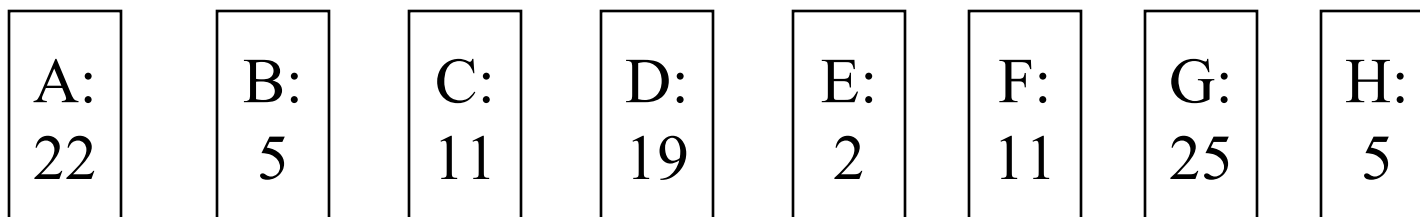
Enlazar X como hijo izquierdo de Z

Enlazar Y como hijo derecho de Z

$T \leftarrow \text{Huffman}(S, n-1)$

Retornar(T)

FinSi



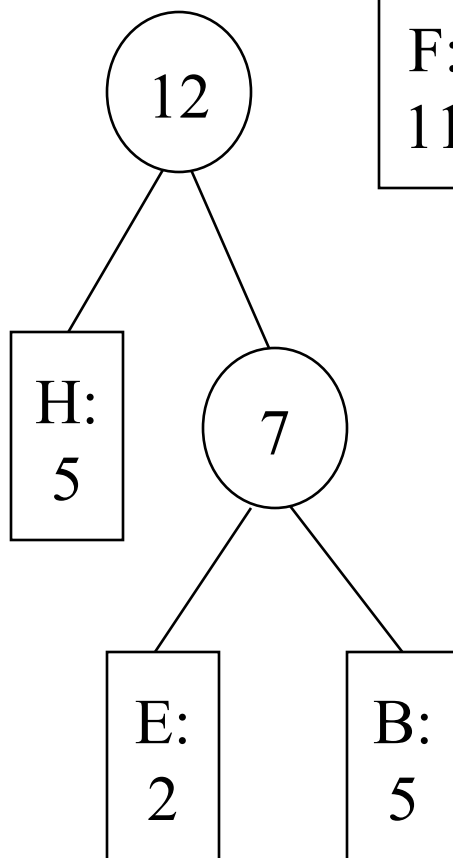
A:  
22

C:  
11

D:  
19

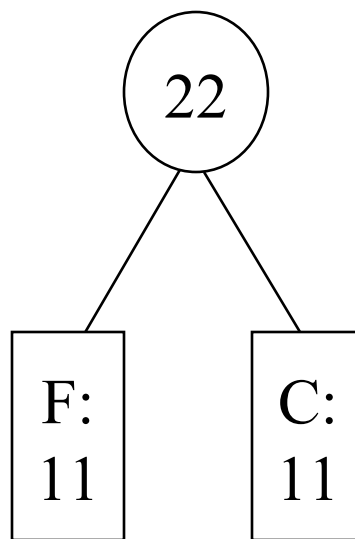
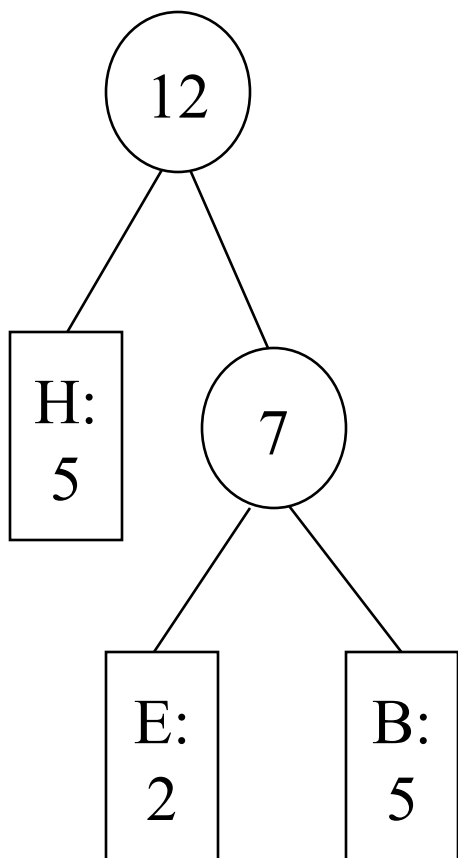
F:  
11

G:  
25



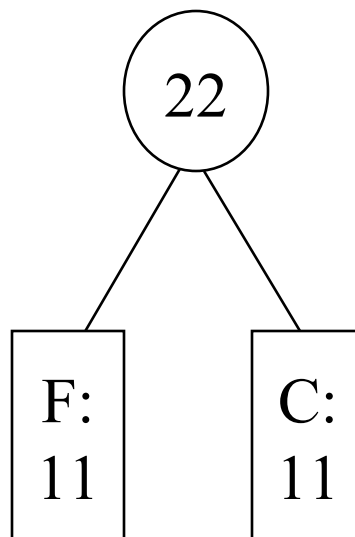
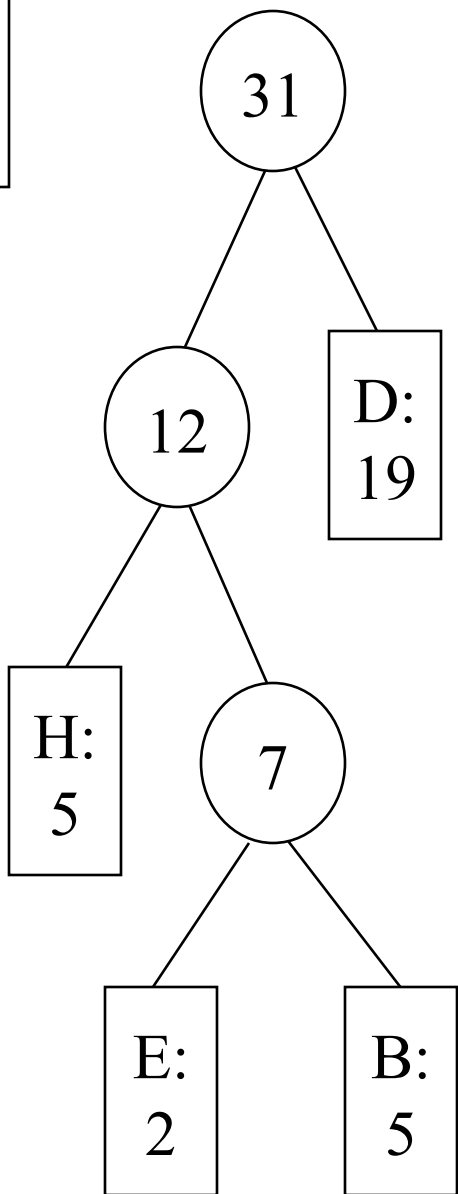
A:  
22

D:  
19



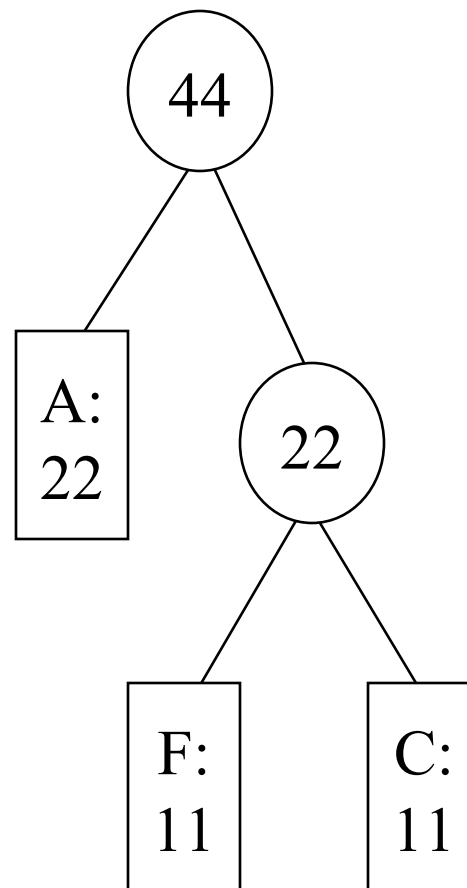
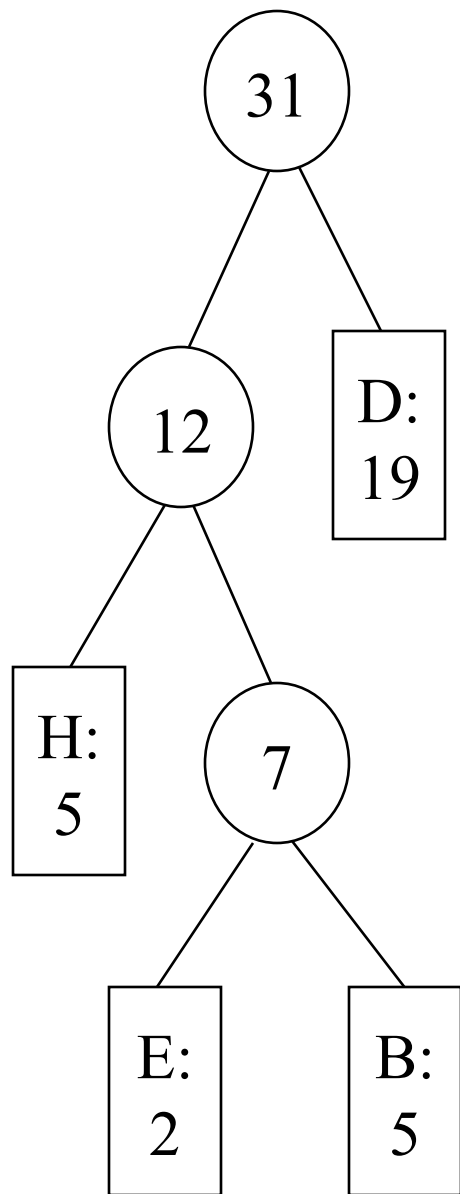
G:  
25

A:  
22

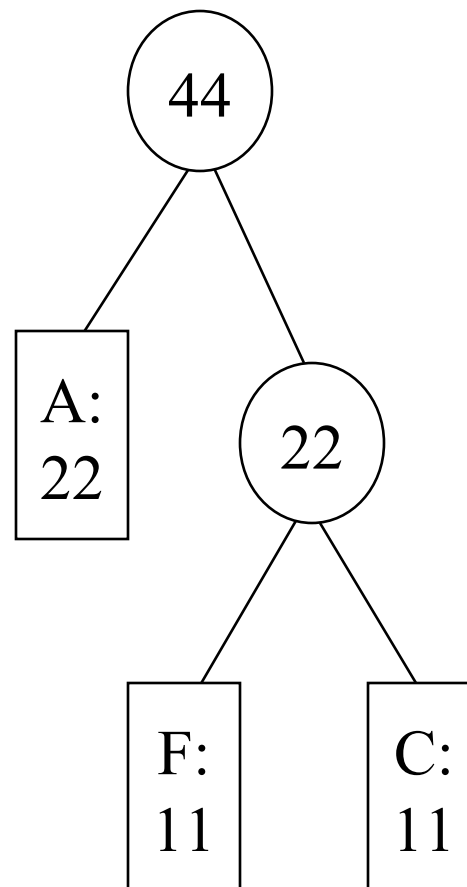
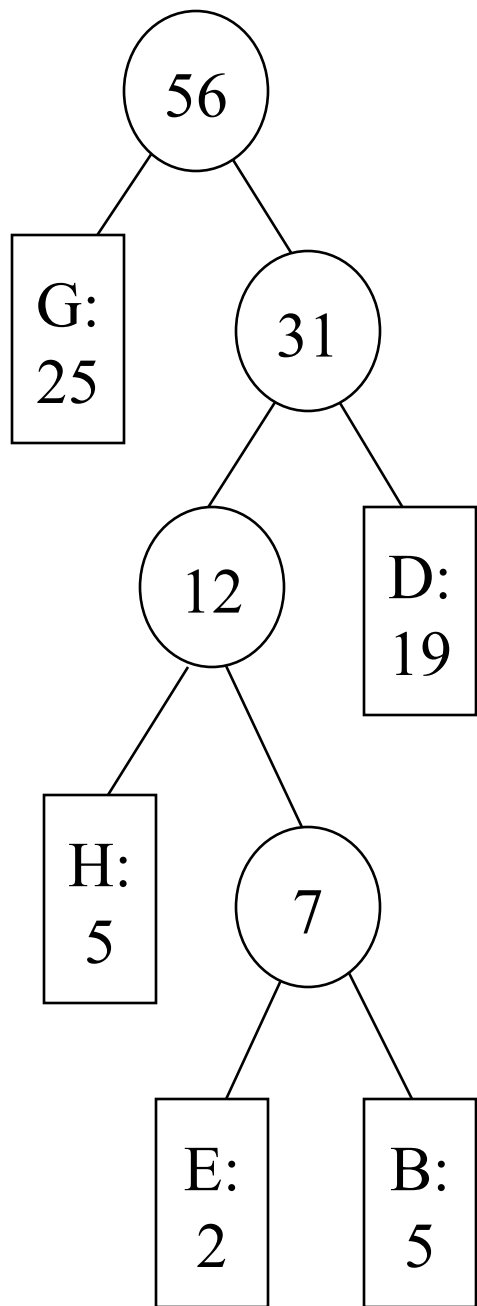


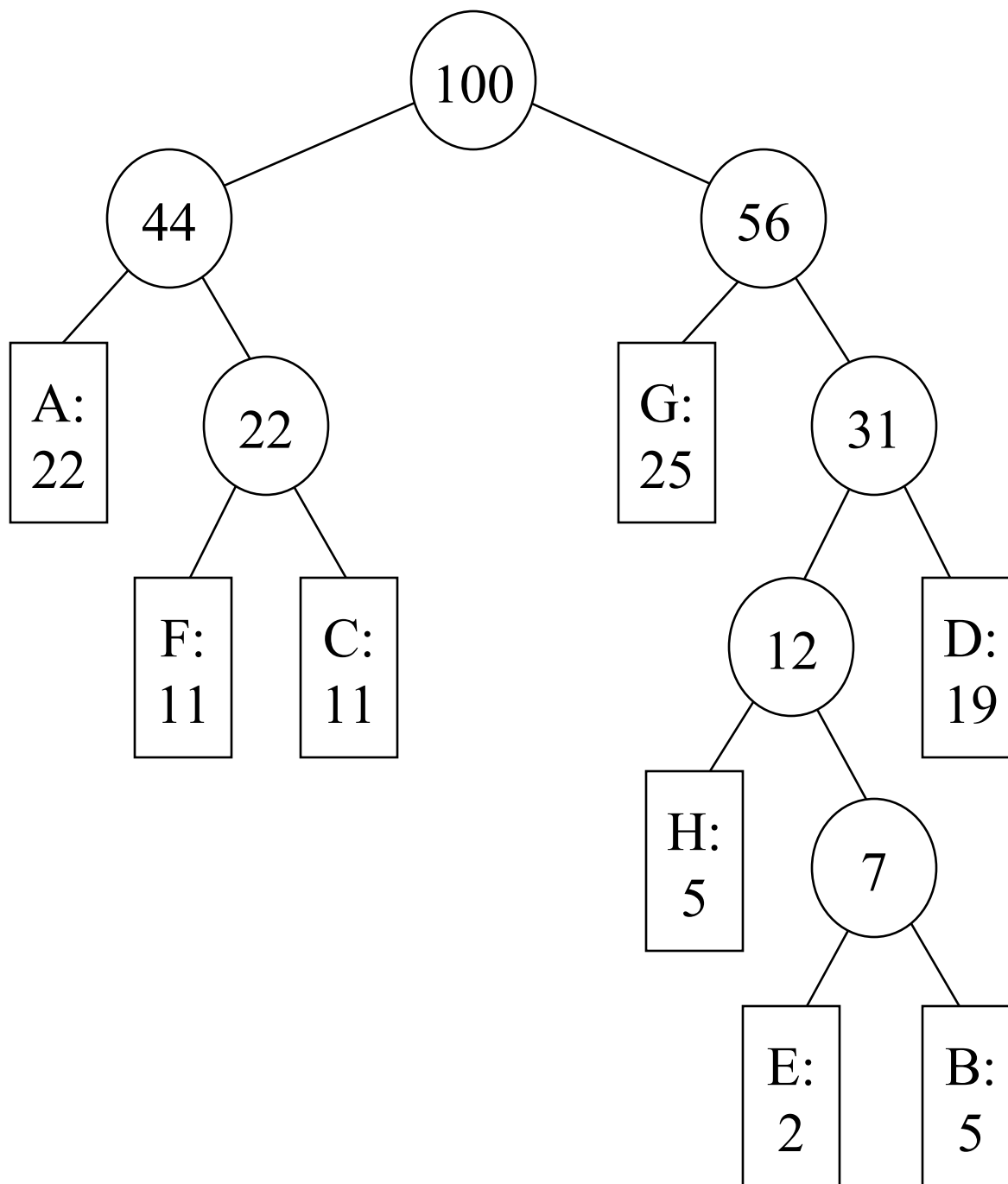
G:  
25





G:  
25





# IMPLEMENTACIÓN DEL ÁRBOL DE HUFFMAN

Se crea un bosque de árboles. Cada árbol corresponde a un nodo de una lista que contiene información acerca del valor del nodo, puntero al hijo izquierdo, puntero al hijo derecho, un campo para el peso (probabilidad o frecuencia) y un puntero al siguiente árbol del bosque.

Registro Nodo

carácter Valor

Nodo \*HI // puntero al hijo izquierdo

Nodo \*HD // puntero al hijo derecho

Nodo \*Hermano // puntero al siguiente árbol del bosque

Numérico Peso // peso del nodo

FinRegistro

Nodo \*Raíz

# IMPLEMENTACIÓN DEL ÁRBOL DE HUFFMAN

**Precondición** La lista enlazada está ordenada por peso y contiene más de dos nodos

**Postcondición** El bosque contiene solo un árbol (de Huffman)

**Método:** los nodos se van agrupando de dos en dos hasta que solo un árbol es el bosque (de Huffman)

**Acción Huffman(R)**      // en forma recursiva

Inicio

HI  $\leftarrow$  Seleccionar\_Menor(R)

HD  $\leftarrow$  Seleccionar\_Menor(R)

P  $\leftarrow$  Construye\_Padre(HI, HD)

Inserta\_Padre(R, P)

Si ( $R \rightarrow \text{Hermano} = \text{Nulo}$ )

Retornar P

Sino

P  $\leftarrow$  Huffman (R)

Retornar P

FinSi

Fin

# IMPLEMENTACIÓN DEL ÁRBOL DE HUFFMAN

**Acción Huffman(R)      // en forma iterativa**

Inicio

    Mientras( $R \rightarrow \text{Hermano} \neq \text{Nulo}$ )

$HI \leftarrow \text{Seleccionar\_Menor}(R)$

$HD \leftarrow \text{Seleccionar\_Menor}(R)$

$P \leftarrow \text{Construye\_Padre}(HI, HD)$

$\text{Inserta\_Padre}(R, P)$

    FinMientras

    Retornar R

Fin

**Acción Seleccionar\_Menor(R)**

Inicio

$P \leftarrow R$

$R \leftarrow P \rightarrow \text{Hermano}$

$P \rightarrow \text{Hermano} \leftarrow \text{Nulo}$

    Retornar P

Fin

# IMPLEMENTACIÓN DEL ÁRBOL DE HUFFMAN

La acción Construye\_Padre crea un nodo ficticio  $n$  y le asigna como hijos los dos nodos seleccionados previamente.

## Acción Construye\_Padre( $P, Q$ )

Inicio

$n \leftarrow$  nuevo nodo

$n \rightarrow \text{Peso} \leftarrow P \rightarrow \text{Peso} + Q \rightarrow \text{Peso}$

$n \rightarrow \text{HI} \leftarrow P$

$n \rightarrow \text{HD} \leftarrow Q$

$n \rightarrow \text{Hermano} \leftarrow \text{Nulo}$

Retornar  $n$

Fin

La acción Inserta\_Padre inserta el nuevo nodo N construido en la lista con cabecera R en el lugar que le corresponde según su peso.

**Acción Inserta\_Padre(R, N)**

Inicio Si (R = Nulo)

R  $\leftarrow$  N

Sino

P  $\leftarrow$  R

Si (P  $\rightarrow$  Peso > N  $\rightarrow$  Peso)

N  $\rightarrow$  Hermano  $\leftarrow$  R

R  $\leftarrow$  N

Sino

Mientras(N  $\rightarrow$  Peso > P  $\rightarrow$  Peso  $\wedge$  P  $\rightarrow$  Hermano  $\neq$  Nulo)

Ant  $\leftarrow$  P

P  $\leftarrow$  P  $\rightarrow$  Hermano

FinMientras

Si (P  $\rightarrow$  Peso  $\geq$  N  $\rightarrow$  Peso)

N  $\rightarrow$  Hermano  $\leftarrow$  P

Ant  $\rightarrow$  Hermano  $\leftarrow$  N

Sino

N  $\rightarrow$  Hermano  $\leftarrow$  P  $\rightarrow$  Hermano

P  $\rightarrow$  Hermano  $\leftarrow$  N

FinSi

FinSi

FinSi

Fin



# APLICACIÓN EN CODIFICACIÓN

Suponga que un conjunto de  $n$  datos  $A_1, A_2, \dots, A_n$ , a ser codificados mediante cadena de  $r$  bits donde

$$2^{r-1} < n \leq 2^r$$

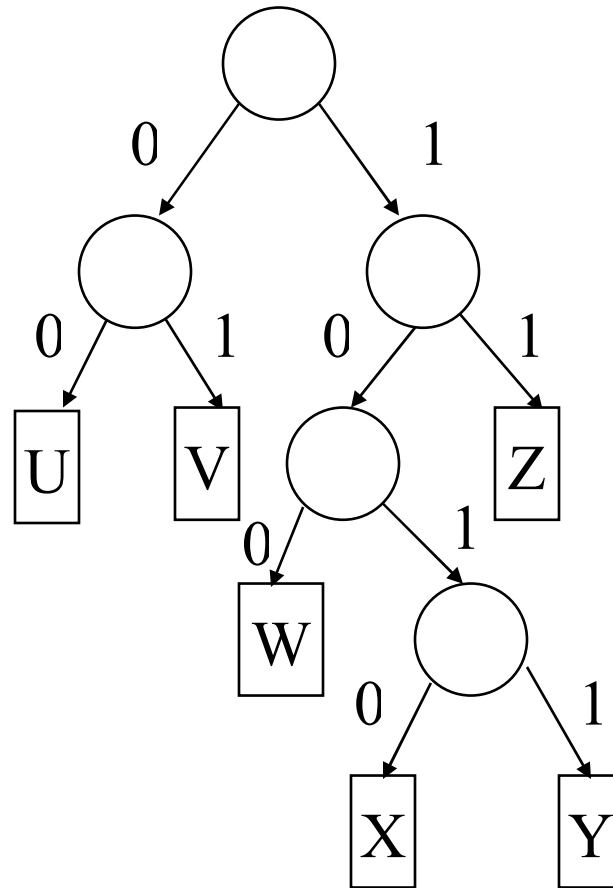
Suponga que los elementos no se dan con la misma probabilidad. Entonces se puede preservar espacio de memoria usando cadenas de longitud variable, de forma que a los elementos que aparezcan más frecuentemente se les asigna cadenas de menor longitud, quedando las de mayor longitud para aquellos datos menos frecuentes.

Considere el árbol-2  $T$  cuyos nodos externos son los elementos  $U, V, W, X, Y$  y  $Z$  de la figura siguiente. Observe que cada arista de un nodo interno hacia su hijo izquierdo esta etiquetada con un 0 y cada arista derecha con un 1. El código Huffman asigna a cada nodo externo la secuencia de bits desde la raíz al nodo. El árbol  $T$  de Huffman determina un código para los nodos externos;

U: 00            V: 01      W: 100   X: 1010   Y: 1011   Z: 11

Este código tiene la propiedad “prefija”; esto es, el código de cualquier elemento no es una subcadena inicial del código de otro elemento. Esto significa que no existe ambigüedad al decodificar cualquier mensaje que usa el código Huffman.

# APLICACIÓN EN CODIFICACIÓN



U:00    V:01    W:100    X:1010    Y:1011    Z:11

# GRAFOS NO DIRIGIDOS

En muchos problemas derivados de la ciencia de la computación, matemáticas, ingeniería y muchas áreas se presentan relaciones entre objetos de datos. Las relaciones entre objetos de datos de un conjunto pueden presentarse en forma de grafos, los cuales a su vez pueden representarse mediante matrices de adyacencias.

Un grafo, corresponde a una relación matemática o una matriz en álgebra, por tanto las propiedades que pueden aplicarse a uno de los tres modelos puede ser aplicado a cualquiera de los otros dos. Si decimos que una relación es simétrica, decimos que el grafo asociado a la relación también es simétrica, la matriz asociada también es simétrica.

En muchas aplicaciones de computación e informática se estudian relaciones simétricas correspondientes a grafos no dirigidos.

# GRAFOS NO DIRIGIDOS

## Definición

Un grafo  $G$  es un modelo matemático definido como el par  $(V, A)$  donde:

$V$ : es el conjunto finito de vértices o nodos

$A$ : es el conjunto finito de aristas o arcos

$V = \{v_i / v_i \text{ es un vértice o nodo, } v_i \in V\}$

$A = \{a_i / a_i \text{ es una arista o camino, } a_i \in V \times V\}$

Podemos definir  $G$  como una relación de la siguiente forma:

$A = \{(u, v) / u, v \in V\}$

El conjunto  $A$  corresponde a la relación  $R$  definida de la siguiente forma:

$u R v$  si y solo si existe una arista  $a = (u, v)$

Si el par  $(u, v)$  es ordenado decimos que  $G$  es grafo dirigido (dígrafo), si no lo es decimos que es un grafo no dirigido.

# GRAFOS NO DIRIGIDOS

## Definición

Si existe la arista  $(u, v)$ , decimos que la arista es **incidente** sobre los vértices  $u, v$ . Decimos que el vértice  $v$  es adyacente al vértice  $u$ .

El vértice  $u$  es el vértice origen, y  $v$  es el vértice destino o terminal. Si un vértice no tiene aristas incidentes, decimos que es un vértice aislado o solitario.

## Definición

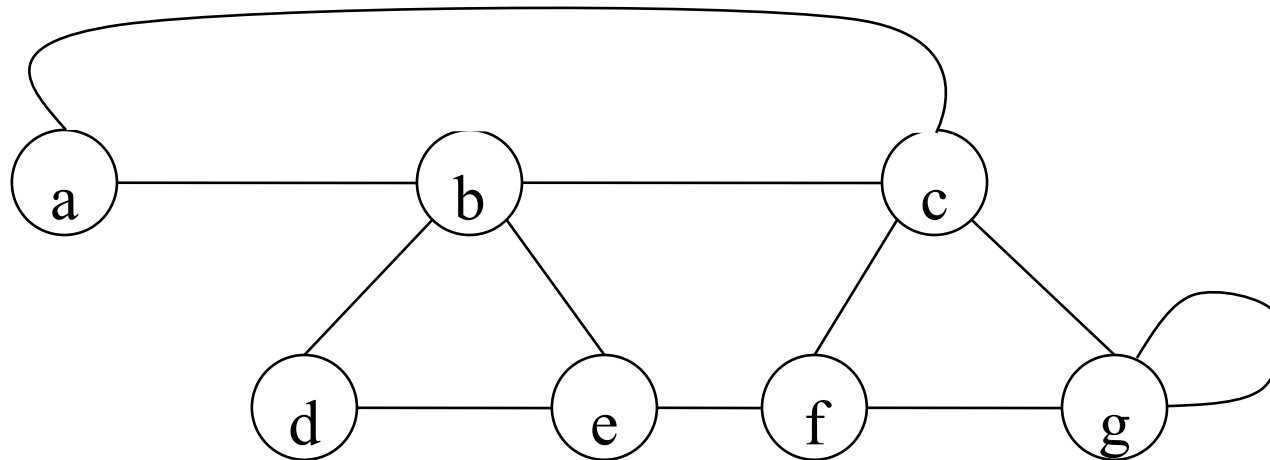
Un grafo no dirigido corresponde a una relación simétrica, pues  $u R v$ , podemos decir también que  $v R u$ .

Como  $R$  es simétrica, decimos que  $u$  y  $v$  son vértices adyacentes, porque  $v$  es adyacente a  $u$ , y  $u$  es adyacente a  $v$ .

# GRAFOS NO DIRIGIDOS

## Grado de un vértice

Sea  $G = (V, A)$  un grafo no dirigido. Para cualquier vértice  $v \in V$ , el grado de  $v$ , se denota por **grad(v)**, es el número de aristas que son incidentes con  $v$



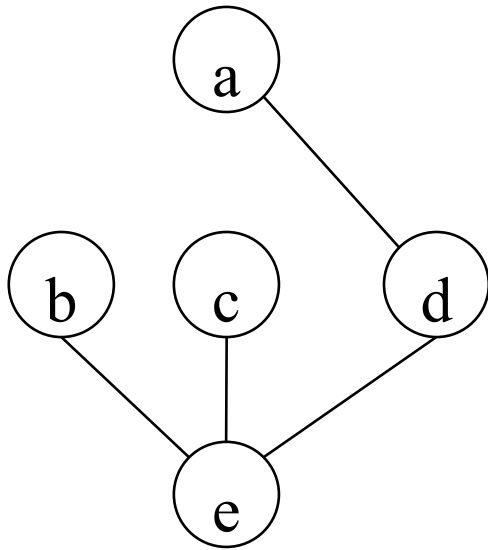
$$\text{Grad}(a) = 2$$

$$\text{Grad}(c) = 4$$

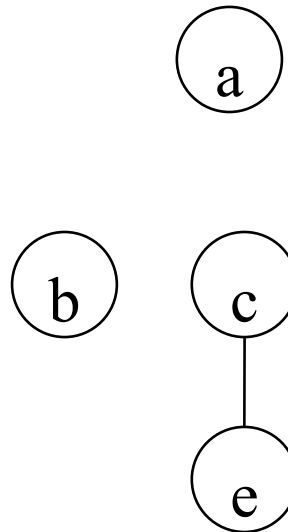
$$\text{Grad}(g) = 4$$

# SUBGRAFOS

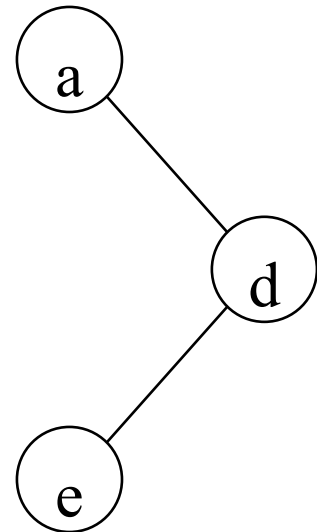
Sea  $G = (V, A)$  un grafo. El grafo  $G' = (V', A')$  es un subgrafo de  $G$ , si  $A' \subseteq A$ , y cada arista de  $A'$  es incidente con los vértices de  $V'$



(a)



(b)



(c)

El grafo (b) y (c) son dos subgrafos del grafo (a)

# SUBGRAFOS INDUCIDOS

Sea  $G = (V, A)$  un grafo

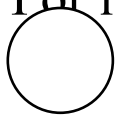
Sea  $G' = (V', A')$  un subgrafo de  $G$

Si  $A'$  consta de todas las aristas  $(u, v) \in A$  tal que  $u$  y  $v \in V'$  entonces decimos que  $G'$  es un subgrafo inducido

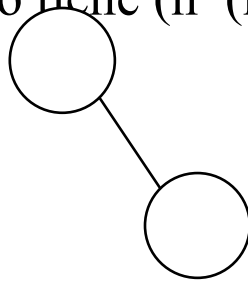
Por ejemplo: El grafo (c) es inducido pero el grafo (b) no lo es porque falta la arista (b, e).

## Grafos completos

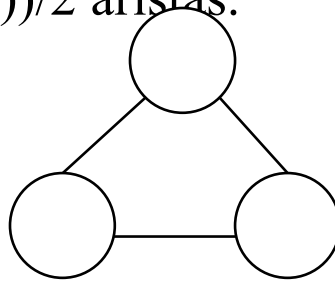
Sea  $G = (V, A)$  un grafo. Si  $|V| = n$ , el grafo completo sobre  $V$ , se denota  $G_n$  es un grafo no dirigido sin lazos tal que para todo  $a, b \in V$  donde  $a \neq b$ , existe una arista  $(a, b)$ . Es decir deben existir aristas entre todos los vértices. Por lo tanto tiene  $(n*(n-1))/2$  aristas.



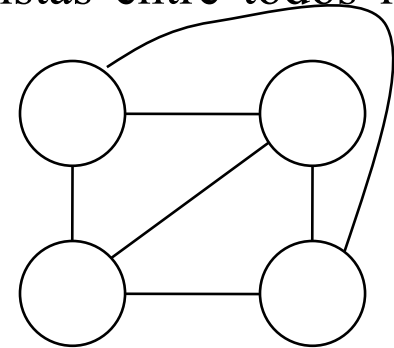
$G_1$



$G_2$



$G_3$



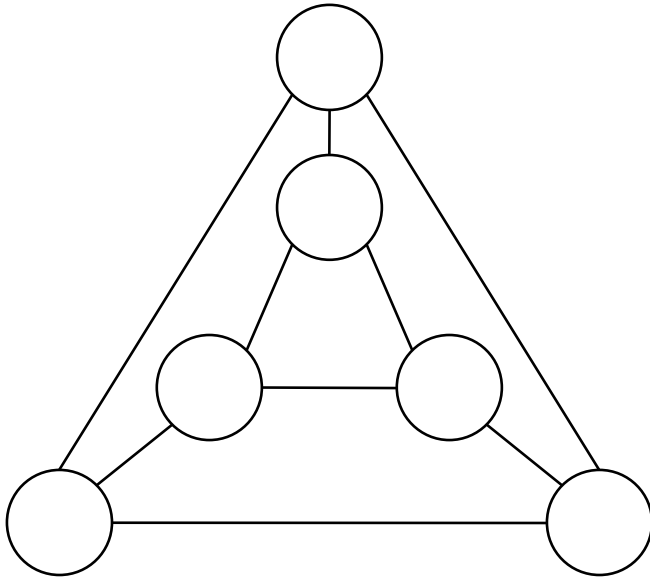
$G_4$

$G_1, G_2, G_3$  y  $G_4$  son grafos completos



# GRAFO PLANO

Un grafo es plano si podemos dibujar  $G$  en el plano de modo que sus aristas se intersecten solo en los vértices de  $G$ . Este dibujo se conoce como una inmersión de  $G$  en el plano.



El grafo es plano  
Cada vértice tiene grado 3  
Ningún par de aristas se  
Intersectan, excepto en los  
vértices

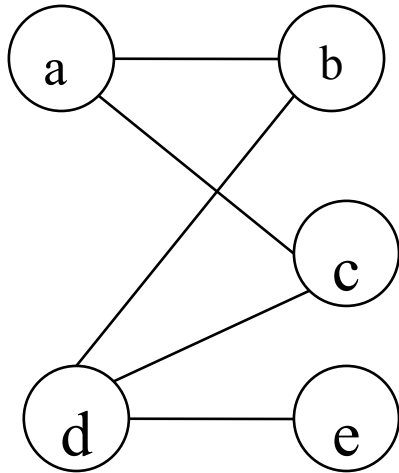
Un mapa de carreteras y autopistas puede representarse mediante un grafo plano. Generalmente las carreteras se intersectan en los puntos de confluencia o en poblaciones. Excepcionalmente, en los pasos a desnivel las carreteras se intersectan.

# GRAFO BIPARTITO

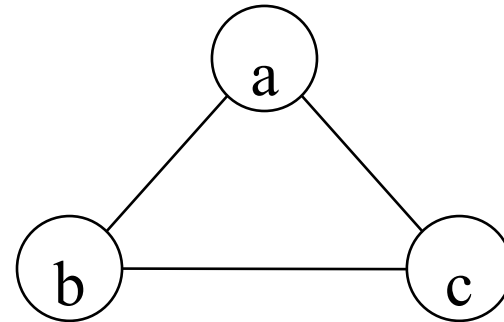
## Definición

Sea  $G = (V, A)$  un grafo,  $G$  se denomina bipartito, si  $V$  se puede descomponer en dos subconjuntos  $V_1$  y  $V_2$  donde

$V = V_1 \cup V_2$  tal que no existen dos vértices adyacentes en  $V_1$  ni tampoco en  $V_2$



(a)



(b)

El grafo (a) es bipartito pues  $V$  se puede dividir en  $\{a, d\}$  y  $\{b, c, e\}$ . El grafo (b) no es bipartito.

# GRAFO BIPARTITO COMPLETO Y GRAFOS ISOMORFOS

## Definición

Sea  $G = (V, A)$  un grafo,  $G$  se denomina bipartito, si  $V$  se puede descomponer en dos subconjuntos  $V_1$  y  $V_2$  donde

$V = V_1 \cup V_2$  y  $V_1 \cap V_2 = \emptyset$ . Si cada vértice de  $V_1$  está unido con los vértices de  $V_2$ , se tiene un grafo bipartito completo.

Si  $|V_1| = m$  y  $|V_2| = n$  El grafo se denota como  $G_{mn}$

## Definición

Sean los grafos no dirigidos  $G_1 = (V_1, A_1)$  y  $G_2 = (V_2, A_2)$ .

Una función

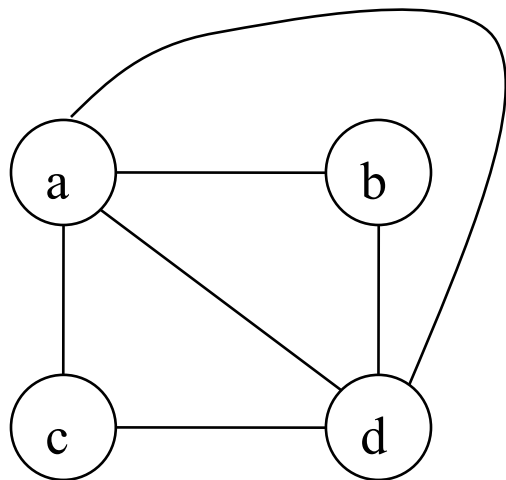
$\Phi: V_1 \rightarrow V_2$  es un isomorfismo entre  $G_1$  y  $G_2$ , si

a)  $\Phi$  es inyectiva y sobreyectiva

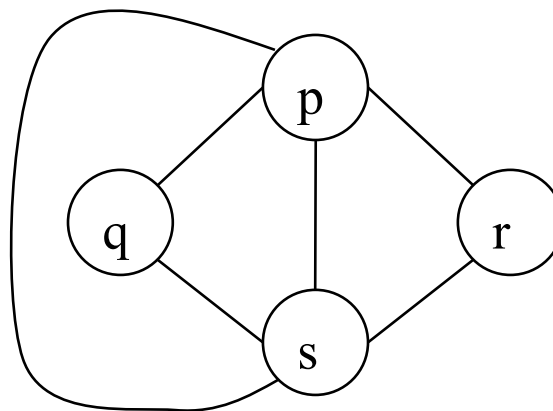
b)  $\forall a, b \in V_1$ ,  $(a, b) \in A_1$  si y solo si  $(\Phi(a), \Phi(b)) \in A_2$

Cuando existe la función  $\Phi$ , decimos que  $G_1$  y  $G_2$  son isomorfos

Un isomorfismo es una correspondencia entre cada elemento de un



(a)



(b)

Los grafos (a) y (b) son isomorfos, porque existe una correspondencia entre los nodos

$$\Phi(a) = p,$$

$$\Phi(b) = r,$$

$$\Phi(c) = q \text{ y}$$

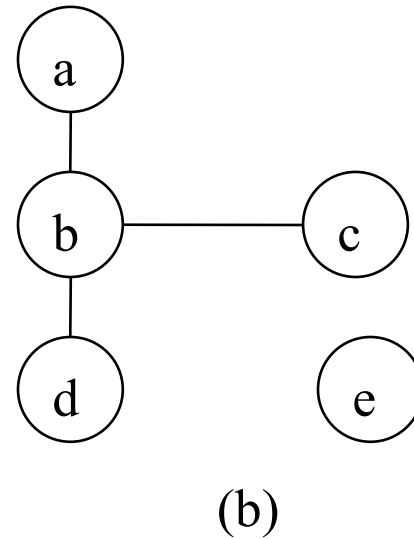
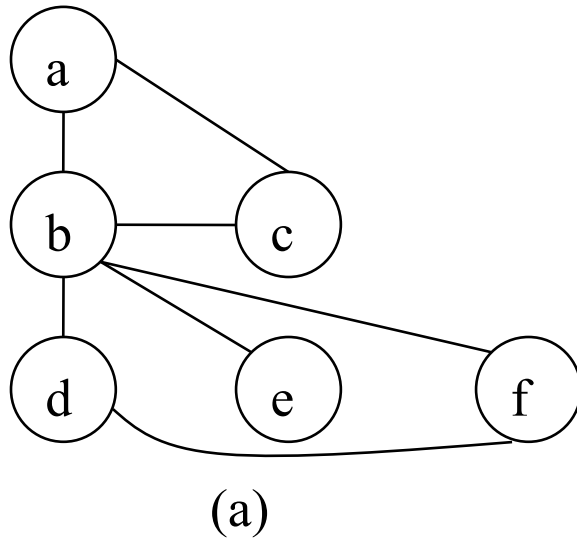
$$\Phi(d) = s$$

La correspondencia entre los vértices de un isomorfismo mantiene las adyacencias.

# GRAFOS CONEXOS

## Definición

Sea  $G = (V, A)$  un grafo no dirigido,  $G$  es conexo si existe un camino simple entre cualquier par de vértices de  $G$ . Si un grafo no es conexo, decimos que es desconexo.

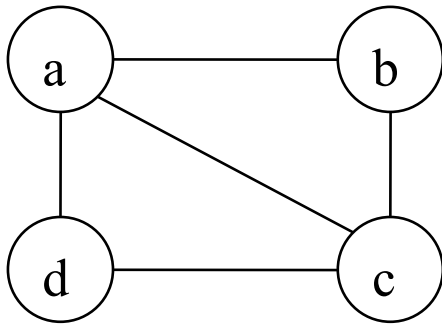


El grafo (a) es conexo. El grafo (b) no es conexo, e es un nodo aislado.

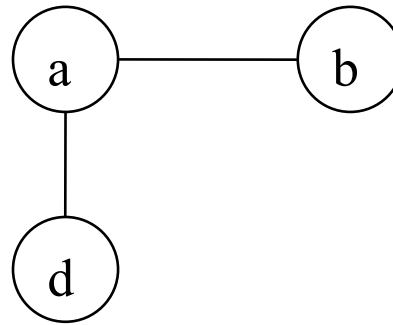
# GRAFOS CONEXOS

## Componentes conexos en un grafo no dirigido

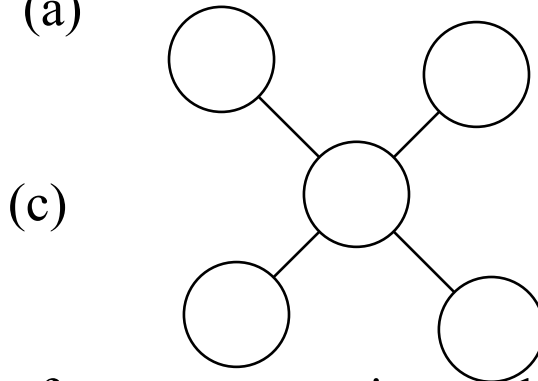
Un componente conexo de un grafo  $G$  es un subgrafo conexo inducido maximal, es decir un grafo conexo inducido que por si mismo no es un subgrafo propio de ningún otro subgrafo conexo de  $G$ .



(a)



(b)



(c)

- (a) Es un grafo conexo que tiene solo un componente conexo, el mismo
- (b) Es uno de los subgrafos inducidos de (a)
- (c) Es un grafo que contiene dos componentes conexos

# REPRESENTACIÓN DE GRAFOS NO DIRIGIDOS

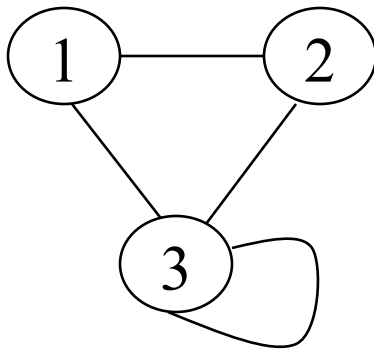
Un grafo no dirigido puede representarse de igual forma que un grafo dirigido, mediante una matriz de adyacencia simétrica o una lista de adyacencia. Aunque es necesario entender que una arista no dirigida  $(u, v)$ , puede representarse como dos aristas dirigidas, una  $(u, v)$  y otra  $(v, u)$ .

Un grafo dirigido puede representarse como una relación simétrica, haciendo uso de estructuras de datos:

- Arreglos bidimensionales (matriz de adyacencia simétrica)
- Listas enlazadas (lista de adyacencia)

# REPRESENTACIÓN MEDIANTE MATRIZ DE ADYACENCIA

Un grafo no dirigido puede representarse mediante una matriz de adyacencia, en donde se representen las aristas desde el vértice  $i$  al vértice  $j$  con un 1 en la entrada  $[i, j]$ , y un cero cuando no exista una arista. La matriz de adyacencia juega un rol muy importante, pues a partir de ella pueden encontrarse otras matrices que relacionan los caminos de longitud mayor a uno, y a partir de ella encontrar la matriz de caminos.



0	1	1
1	0	1
1	1	1

**Matriz de  
Adyacencia  
simétrica**

$$\text{Grad}(1) = 2 \quad \text{Grad}(2) = 2 \quad \text{Grad}(3) = 4$$



A partir de la matriz de adyacencia se puede obtener el grado de un vértice.

Entrada : A matriz de dimensión  $N \times N$

Salida :  $\text{grado}(u)$ , u el vértice a verificar

La función Obtener  $\text{grado}(u)$  suma los elementos de la fila u. Cuando encuentra un 1 en la columna j, significa que j es adyacente a u. Después de sumar toda la fila correspondiente al vértice que se esta verificando, se suma el elemento  $A[i, i]$  que significa 1 si es un bucle o 0 si no hay un bucle.

**Función Obtener\_Grado(A, N, u) : Entero**

Inicio

$i \leftarrow u$

$\text{grado} \leftarrow 0$

Para j desde 1 hasta N

$\text{grado} \leftarrow \text{grado} + A[i, j]$

FinPara

$\text{grado} \leftarrow \text{grado} + A[i, i]$  // suma el bucle si lo hubiese

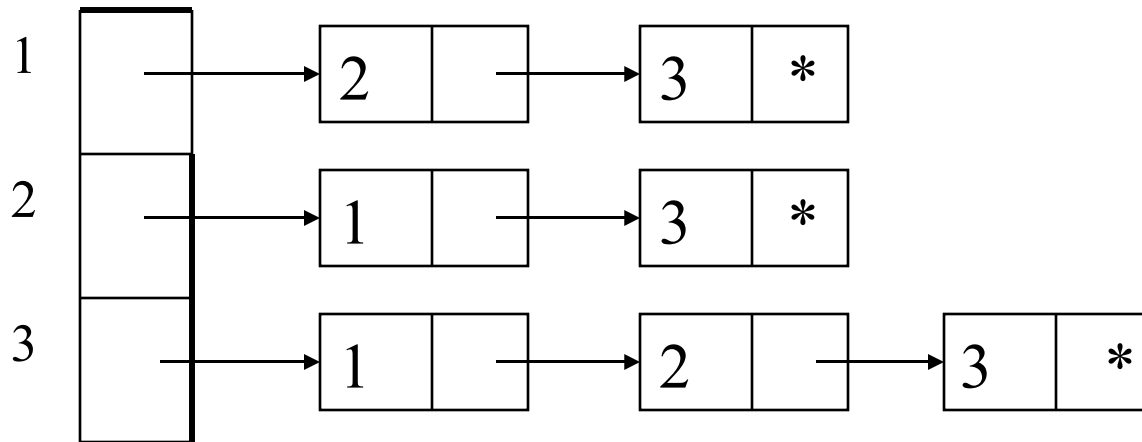
Retornar grado

Fin

# REPRESENTACIÓN MEDIANTE LISTAS ENLAZADAS

## Lista de adyacencia

Una lista de adyacencia es una lista de los vértices. Cada uno de los vértices apunta a su vez a una lista de los vértices adyacentes.



# **RECORRIDO DE UN GRAFO NO DIRIGIDO**

## **Recorrido en profundidad REP en un grafo no dirigido**

La idea general del recorrido en profundidad comenzando en un nodo  $v$  es la siguiente: Primero examinamos el nodo inicial. Luego examinamos un nodo adyacente (vecino) a  $v$ , luego un adyacente del adyacente de  $v$  y así sucesivamente hasta llegar a un punto muerto, o sea al final del camino. Al volver si hay otro nodo adyacente a  $v$  no visitado, se toma como punto de partida este nodo y se realiza el mismo procedimiento. Cuando estén examinados (visitados) todos los nodos adyacentes a  $v$ , el recorrido que se inicio con  $v$  ha finalizado. Si queda algún nodo sin visitar, tomamos cualquiera de ellos y procedemos como con el nodo  $v$ . El procedimiento sigue hasta que estén examinados (visitados) todos los nodos del grafo.

# RECORRIDO DE UN GRAFO NO DIRIGIDO

## Recorrido en profundidad REP en un grafo no dirigido

**Entrada:** G grafo, v vértice inicial, que será la raíz del árbol

**Salida:** A árbol abarcador

**Precondición:** La búsqueda empieza en el nodo v

### Acción Recorrido()

Inicio

Para cada  $v \in V$

Visita(v)  $\leftarrow$  0 // inicializa los nodos como no visitados

FinPara

Para cada  $v \in V$

Si (Visita(v) = 0)

REP(v) // llama a procedimiento de búsqueda

FinSi

FinPara

Fin

# RECORRIDO DE UN GRAFO NO DIRIGIDO

## Recorrido en profundidad REP en un grafo no dirigido

### Acción REP(v)

Inicio

Visita(v)  $\leftarrow$  1 // cuando se visita el nodo v su estado cambia a 1

Para cada nodo w adyacente a v

Si (Visita(w) = 0)

Inserta(A, v, w) // inserta la arista (v, w) en A

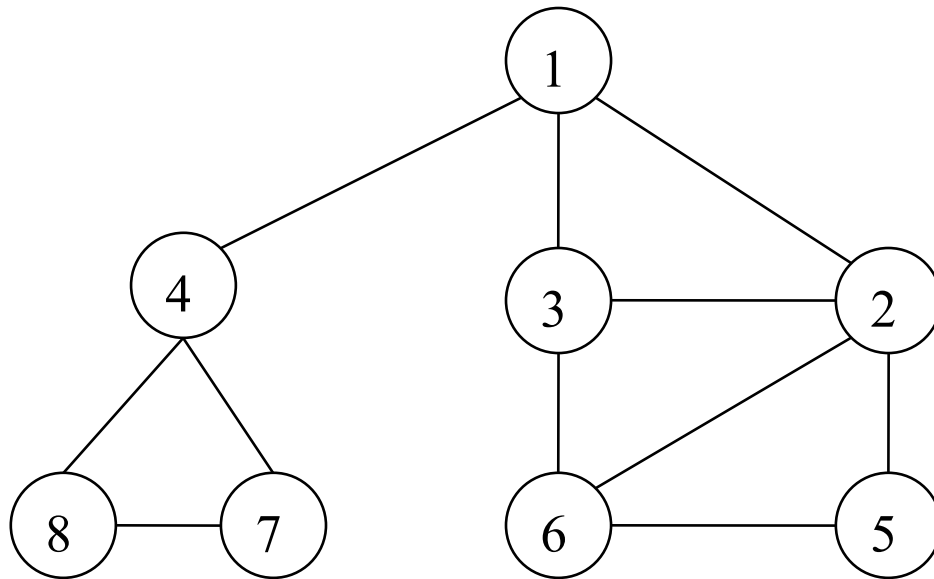
**REP(w)** // llama a procedimiento REP para w

FinSi

FinPara

Fin

La recursión sólo se detiene cuando la exploración del grafo se ve bloqueada (en un punto muerto) y no se puede proseguir.



## REP = recorrido en profundidad

2BP(1)

llamada inicial

3 BP(2)

llamada recursiva

4 BP(3)

llamada recursiva

5 BP(6)

llamada recursiva

6 BP(5)

llamada recursiva, no se puede continuar

7 BP(4)

no se ha visitado el nodo 4 vecino de 1

8 BP(7)

llamada recursiva

9 BP(8)

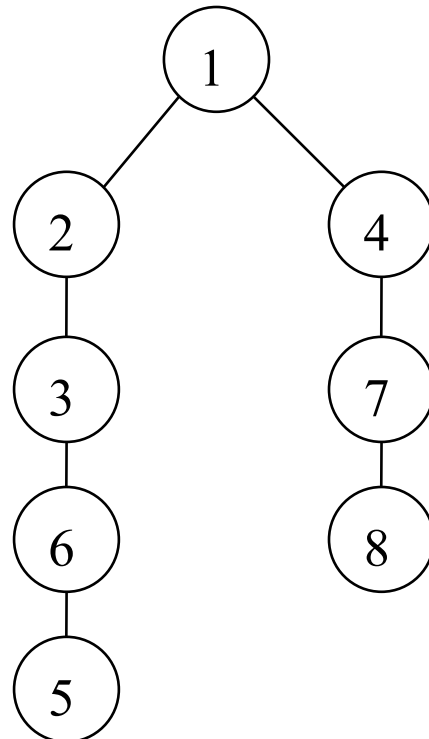
llamada recursiva, no se puede continuar

10 No quedan nodos por visitar

Cada nodo de  $G$  se visita solo una vez. Por lo tanto el algoritmo recursivo se invoca  $n$  veces. El recorrido en profundidad de un grafo conexo asocia un árbol abarcador (recubridor), en donde el punto de partida elegido pasa a ser la raíz del árbol.

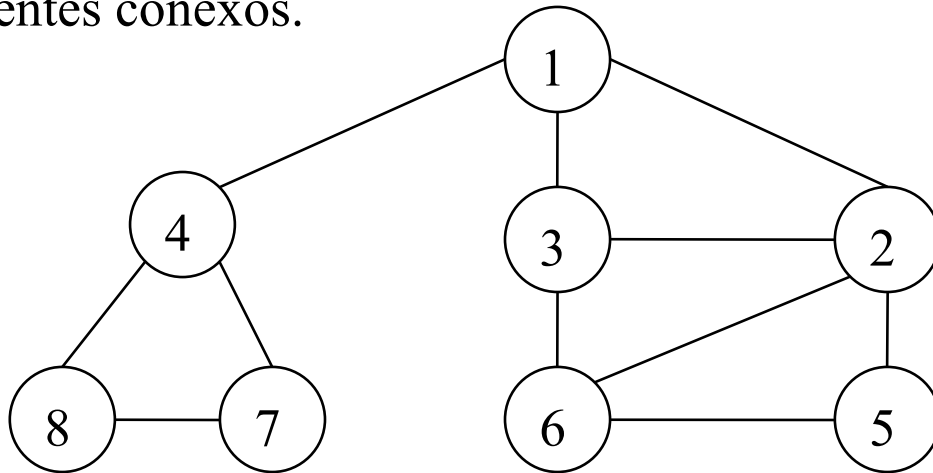
Si el grafo no es conexo, entonces un recorrido en profundidad, asocia al grafo un bosque de árboles abarcadores. Uno para cada componente conexo.

En el ejemplo anterior se construye el árbol abarcador:

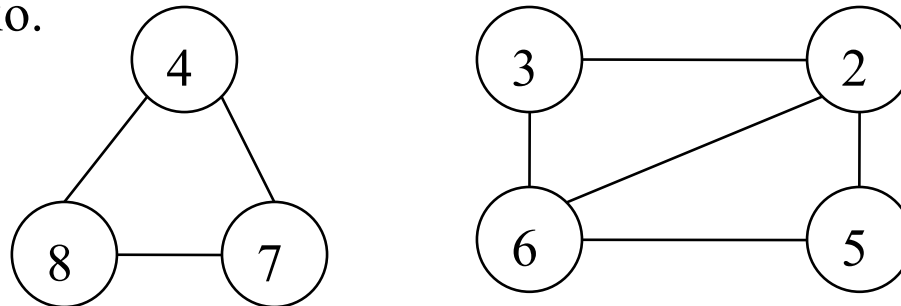


## Puntos de articulación

En un grafo conexo, puede existir un vértice que sirva de articulación entre dos componentes conexos. En el ejemplo el vértice 1 es de articulación, ya que si se elimina el vértice 1 y las aristas que inciden en él, se obtendría un subgrafo que ya no es conexo. Quedarían dos componentes conexos.



Si se elimina el vértice 1 y las aristas que inciden en él, se obtendría un grafo que ya no es conexo.



Un grafo se denomina biconexo si carece de puntos de articulación.



# RECORRIDO DE UN GRAFO NO DIRIGIDO

## Recorrido en amplitud REA en un grafo no dirigido

En un recorrido en amplitud, el recorrido se inicia con un nodo de partida  $v$ . A continuación se visitan los nodos adyacentes al nodo inicial  $v$  (nodos a una distancia 1 del nodo  $v$ ), luego se visitan los nodos que están a una distancia 2 y así sucesivamente hasta que no quede ningún nodo por visitar.

El algoritmo de recorrido en amplitud no es recursivo

**Entrada:**  $G$  grafo,  $v$  vértice inicial, que será la raíz del árbol

**Salida:** A árbol abarcador

**Precondición:** La búsqueda empieza en el nodo  $v$

La cola  $C$  debe estar vacía.

# RECORRIDO DE UN GRAFO NO DIRIGIDO

## Acción REA(v)

Inicio

Init(C) // cola vacía

Visita(v)  $\leftarrow$  1 // v es visitado

Encolar(C, v)

Mientras ( $\neg$ Vacíó(C))

Decolar(C, v)

Mientras exista w adyacente a v

Si (Visita(w) = 0)

Visita(w)  $\leftarrow$  1

Inserta(A, v, w) // inserta la arista (v, w) en el árbol

A

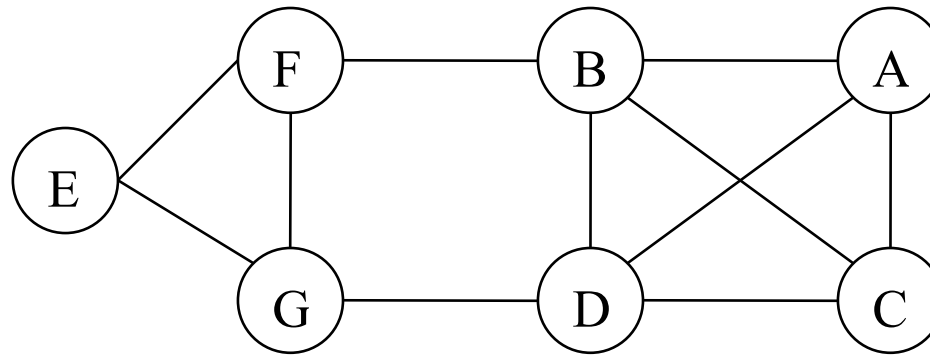
Encolar(C,w)

FinSi

FinMientras

FinMientras

Fin



Si se comienza con A, se visitan los nodos

Origen	A	
primero	B, C, D	distancia 1
luego	F, G	distancia 2
Finalmente	E	distancia 3

Si se comienza con E, se visitan los nodos

Origen	E	
primero	F, G	distancia 1
luego	B, D	distancia 2
finalmente	A, C	distancia 3