



# ARBOL

**ESTRUCTURAS DE DATOS  
y ALGORITMOS  
LCC – LSI - TUPW**

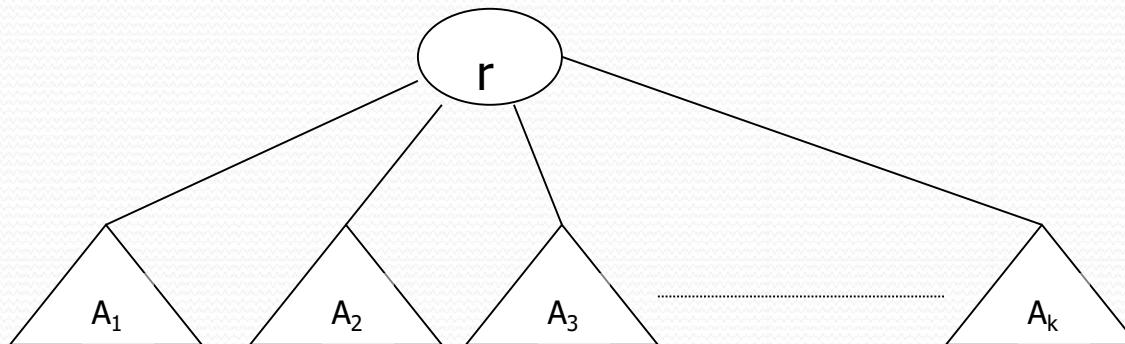
# Objetivos

- Conocer distintos tipos de árboles y sus aplicaciones habituales.
- Realizar un análisis comparativo entre los mismos, a partir de la evaluación de costo de ejecución de los algoritmos que los manipulan.
- Construir los TADs: Árbol BB, AVL, B y Montículo Binario.

# ARBOL

Un **ÁRBOL** es una colección o conjunto de nodos. La colección puede:

- 1 - Estar vacía
- 2 - Si no está vacía, consiste de un nodo distinguido  $r$ , conocido como *raíz*, y cero o más (sub)árboles  $A_1, A_2, \dots, A_k$ , cada uno de los cuales tiene su raíz conectada a  $r$  por medio de una *arista* (o *rama*) dirigida.



# ARBOL

- La raíz de cada subárbol es un **hijo** o **descendiente directo** de r, y r es el **padre** o **antecesor directo** de cada raíz de los subárboles.
- **Camino de un nodo ni a otro nk:** secuencia de nodos  $n_1, n_2, \dots, n_k$ , tal que  $n_i$  es el padre de  $n_{i+1}$  para  $1 < i < k$ . En un árbol existe solamente un camino desde la raíz a cada nodo.
- Si hay un camino entre los nodos  $n_1$  y  $n_2$ , entonces  $n_1$  es **antecesor** de  $n_2$  y  $n_2$  es **descendiente** de  $n_1$ .
- **Longitud de camino de un nodo ni a otro nk:** Número de aristas que forman el camino, o número de nodos menos 1 que forman la secuencia. Existe un camino de longitud cero desde cada nodo a sí mismo.
- **Nivel de un nodo:** si el nodo x está en el nivel i, entonces sus descendientes directos están en el nivel  $i+1$ . La raíz de un árbol, se define como localizada en el nivel 1.
- **Profundidad o Altura del árbol:** máximo de los niveles de todos los nodos del árbol.
- **Grado de un nodo:** Número de descendientes directos de un nodo.
- **Grado del árbol:** Grado máximo en todos los nodos.
- **Nodo hoja:** nodo de grado 0.
- **Nodo Interior:** Nodo no hoja.
- **Árbol Ordenado:** Árbol en el que las ramas de cada nodo están ordenadas.

# Árbol Binario

Un árbol ordenado de grado 2 se denomina **Árbol Binario**.

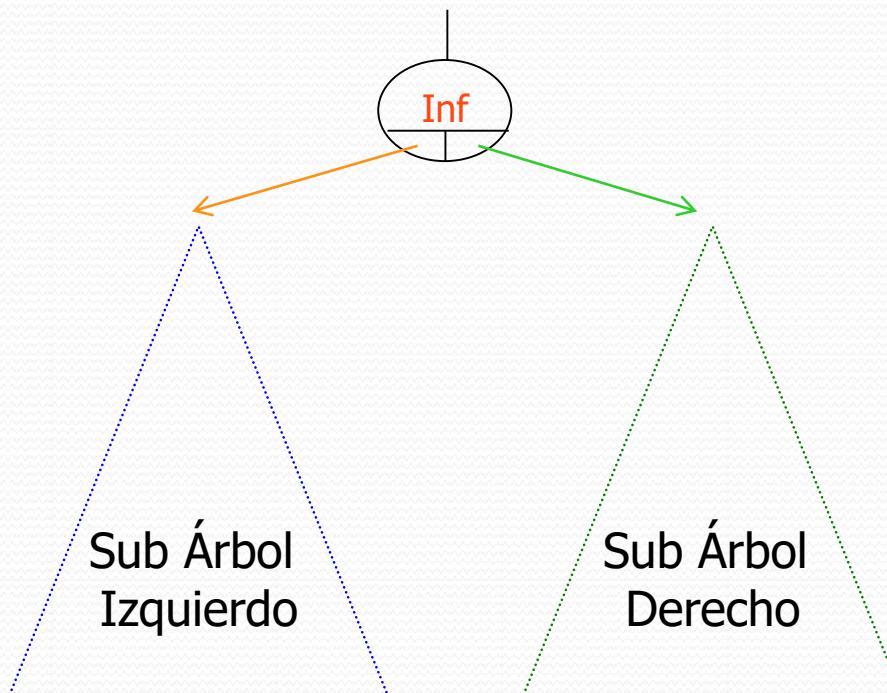
Este árbol se define como un conjunto finito de elementos (nodos) que:

- es vacío o
- consta de una raíz (nodo) con dos árboles binarios disjuntos llamados **subárbol izquierdo** y **subárbol derecho** de la raíz.

# Árbol Binario

## Representación

Representación enlazada



# Árbol Binario

## Aplicación

*Generación de Códigos de Huffman*

*Códigos de longitud variable*

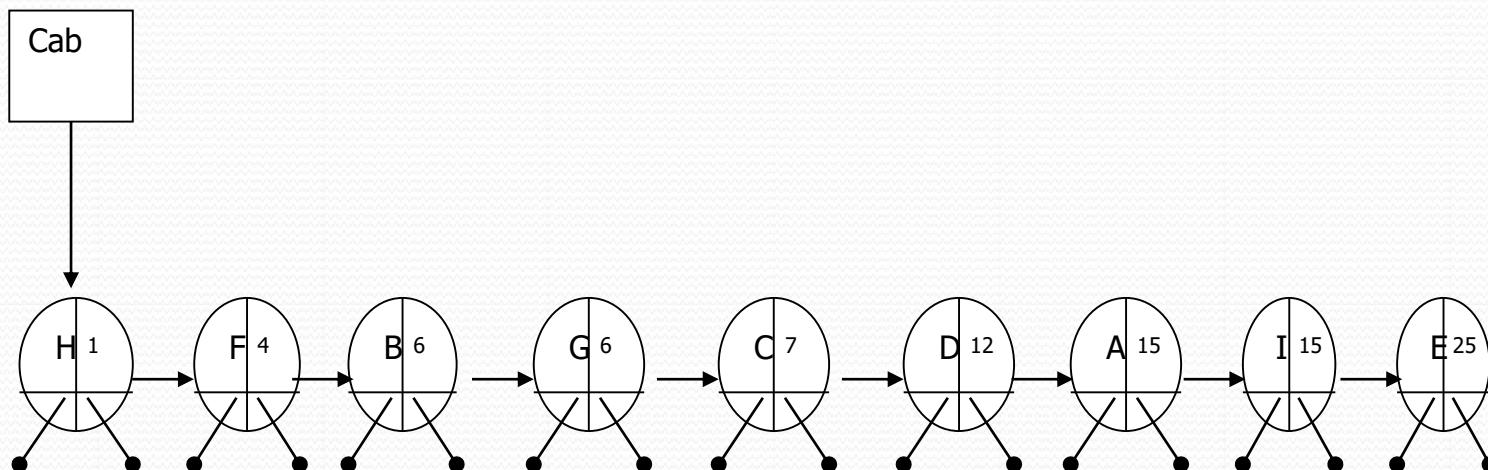
<i>Carácter</i>	A	B	C	D	E	F	G	H	I
<i>Frecuencia</i>	15	6	7	12	25	4	6	1	15

# Árbol Binario

## Aplicación

### Generación de Códigos de Huffman

**Paso 1:** Generar una lista de árboles binarios. Cada raíz, inicialmente, contiene un carácter y su frecuencia de ocurrencia.  
La lista está ordenada en forma creciente por las frecuencias.

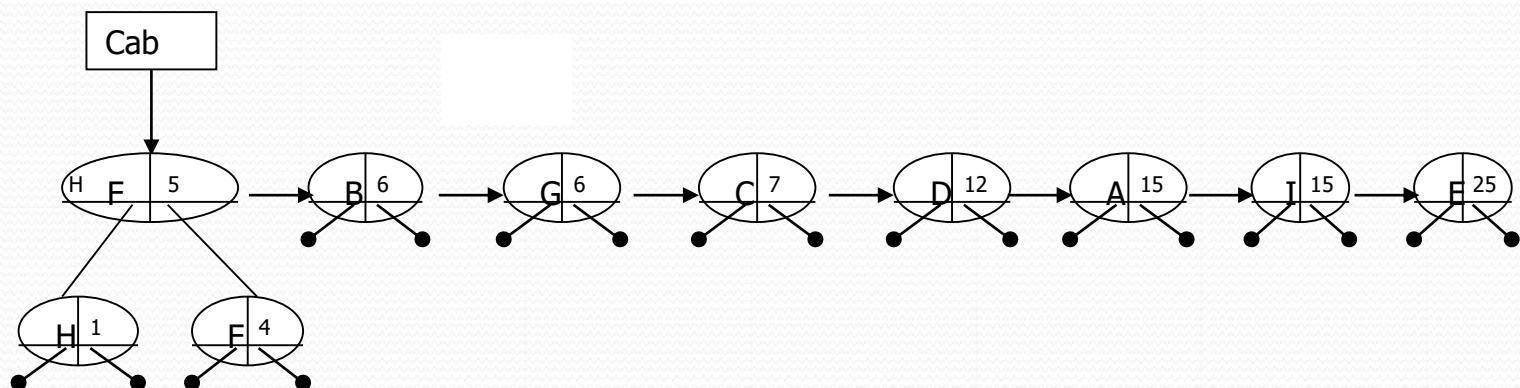


# Árbol Binario

## Aplicación

### Generación de Códigos de Huffman

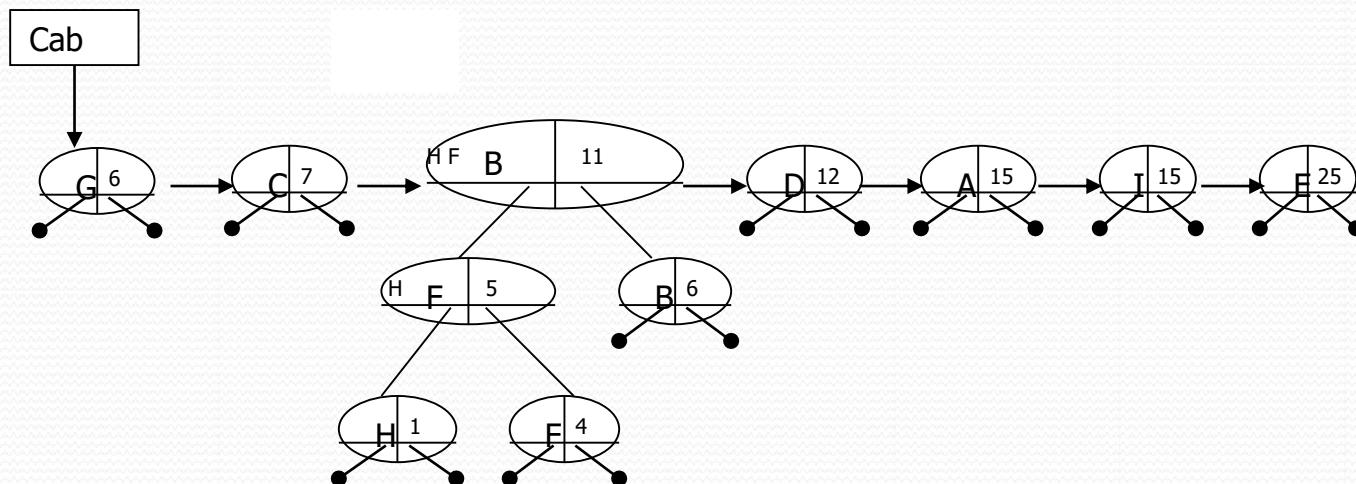
- **Paso 2:** Generar repetidamente árboles binarios a partir de aquellos con frecuencias menores, e insertarlos nuevamente en la lista ordenada. La frecuencia asociada a la raíz de cada nuevo árbol es la *suma* de las frecuencias de sus subárboles.



# Árbol Binario

## Aplicación

### Generación de Códigos de Huffman

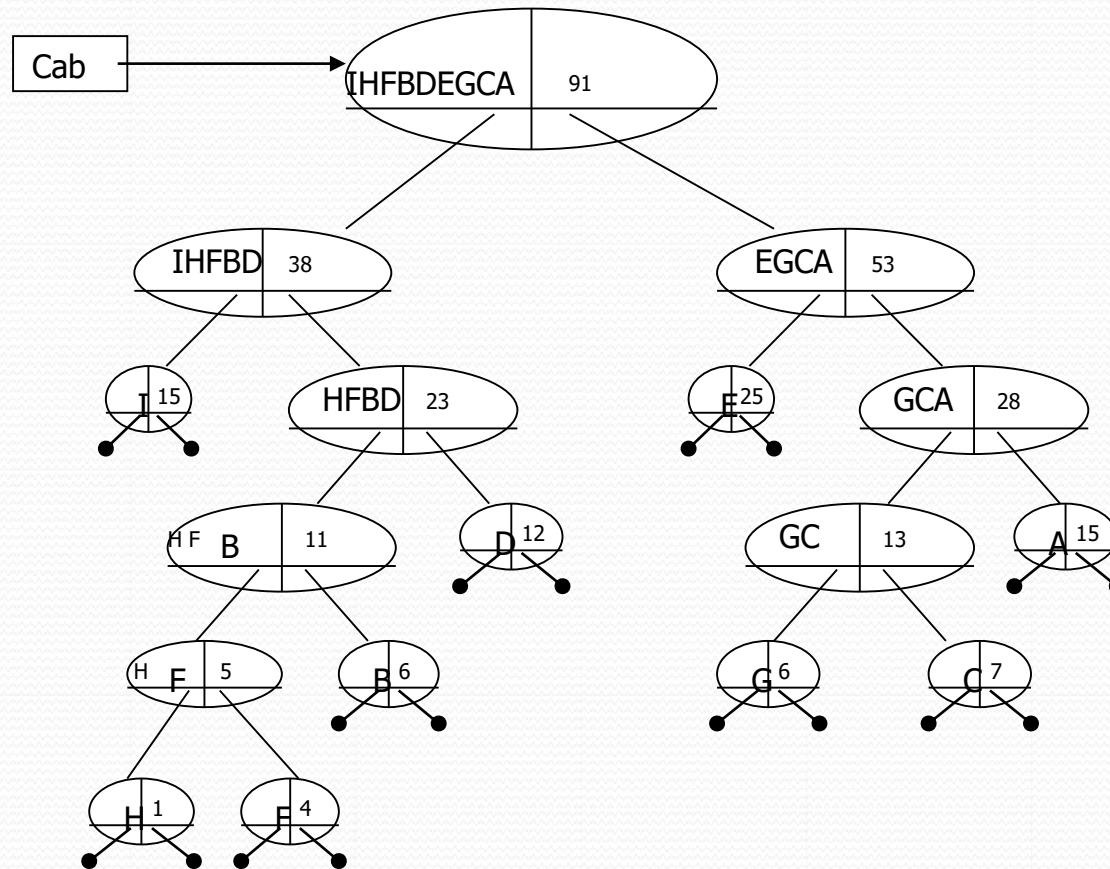


Esta secuencia de supresiones, síntesis e inserciones se realiza hasta el momento en que en la lista queda una sola celda, que corresponde a la raíz del árbol binario que tiene a todos los caracteres del diccionario de entrada como nodos hoja.

# Árbol Binario

## Aplicación

### Generación de Códigos de Huffman

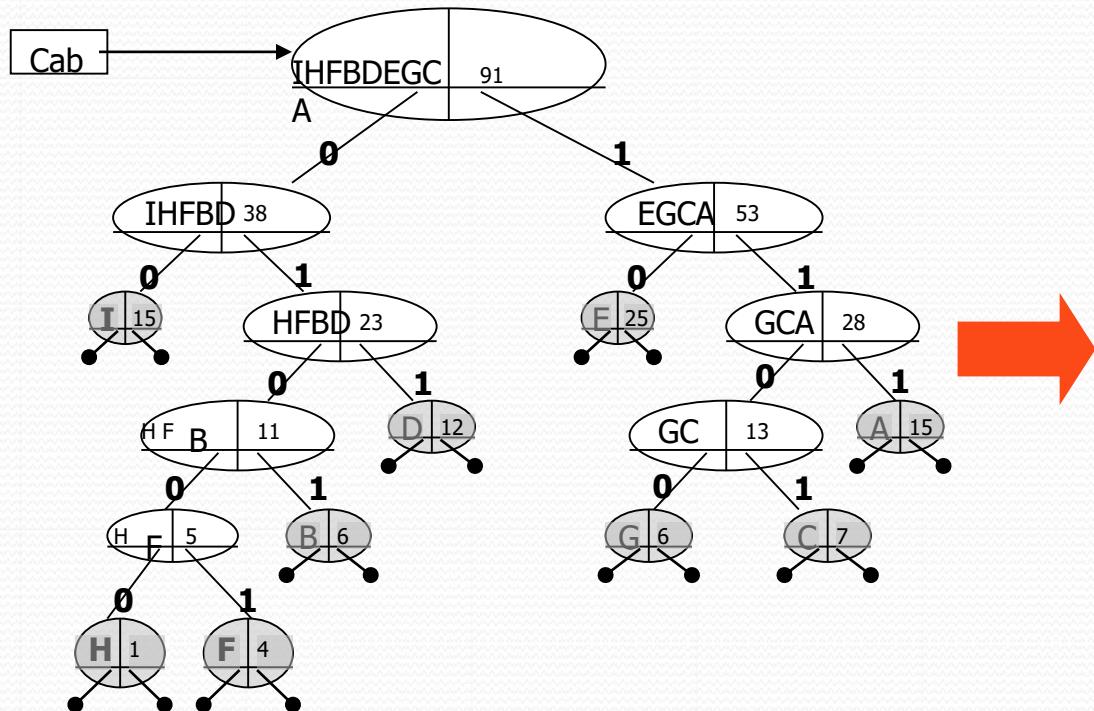


# Árbol Binario

## Aplicación

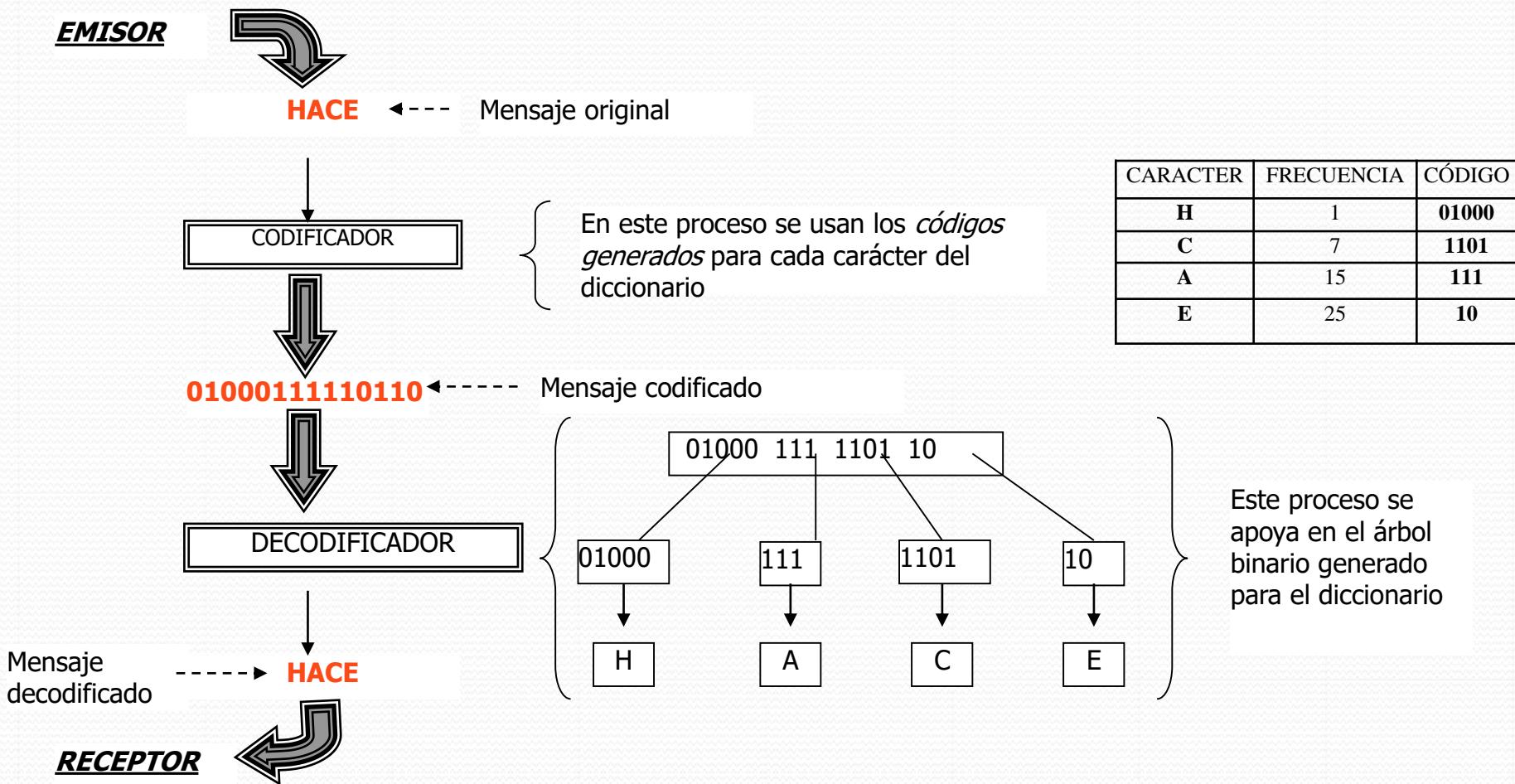
### Generación de Códigos de Huffman

**Paso 3:** Generar el código de longitud variable correspondiente a cada carácter, hoja del árbol, concatenando ceros y unos, según se atraviese en el árbol una rama izquierda o una rama derecha.



CARACTER	FRECUENCIA	CÓDIGO
H	1	01000
F	4	01001
B	6	0101
G	6	1100
C	7	1101
D	12	011
A	15	111
I	15	00
E	25	10

# Codificación-Decodificación



# T.A.D. Árbol Binario de Búsqueda

## Especificación (1)

Un **Árbol Binario de Búsqueda –ABB-** es un Árbol Binario en el que:

- Cada nodo contiene un campo clave (lo identifica de forma única dentro del árbol).
- Los nodos del árbol están ordenados de tal forma que para cualquier nodo  $x$  del árbol,
  - si  $z$  es un nodo cualquiera del subárbol izquierdo de  $x$ , entonces  $\text{clave}[z] < \text{clave}[x]$  y
  - si  $z$  es un nodo cualquiera del subárbol derecho de  $x$ , entonces  $\text{clave}[x] < \text{clave}[z]$ .

Esta condición es conocida como ***Propiedad del Árbol Binario de Búsqueda.***

# T.A.D. ABB

## Especificación(2)

### Operaciones Abstractas

Sean A: Árbol; X,Z : claves

NOMBRE	ENCABEZADO	FUNCION	ENTRADA	SALIDA
Crear	Crear (A)	Inicializa el árbol A	A	A=()
Insertar	Insertar(A, X)	Ingresa el elemento X en el árbol A, manteniéndolo como árbol binario de búsqueda.	A, X	A con X como hoja, si X $\notin$ A; Error en caso contrario
Suprimir	Suprimir(A, X)	Elimina el elemento X del árbol A, manteniéndolo como árbol binario de búsqueda.	A, X	A sin el nodo que contenía a X, si X $\in$ A; Error en caso contrario
Buscar	Buscar(A,X)	Localiza el nodo con clave X en el árbol A	A,X	Reporta los datos asociados con X, si X $\in$ A; Error en caso contrario
Nivel	Nivel(A, X)	Calcula el nivel del nodo con clave X	A, X	Reporta el nivel del nodo con clave X si X $\in$ A; Error en caso contrario
Hoja	Hoja(A, X)	Evalúa si el nodo con clave X es hoja	A, X	Verdadero si el nodo con clave X es hoja, Falso en caso contrario.

# T.A.D. ABB

## Especificación(3)

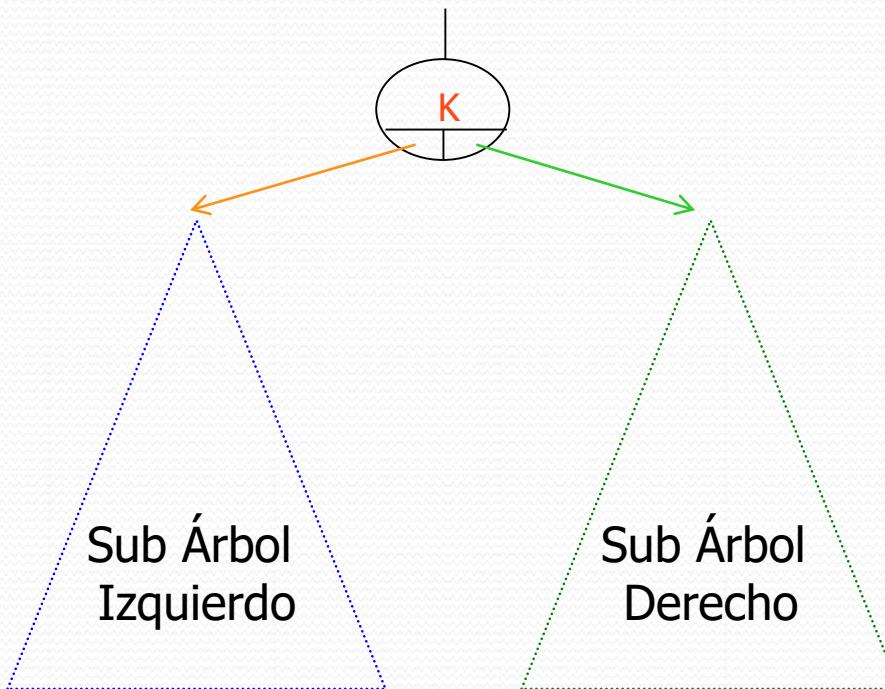
Operaciones Abstractas

Sean A: Árbol; X,Z : claves

<i>NOMBRE</i>	<i>ENCABEZADO</i>	<i>FUNCION</i>	<i>ENTRADA</i>	<i>SALIDA</i>
Hijo	Hijo(A, X, Z)	Evalúa si X es hijo (descendiente directo) de Z	A, X, Z	Verdadero si el nodo X es hijo del nodo Z, <b>Falso en caso contrario.</b>
Padre	Padre(A, X, Z)	Recíproca de la operación Hijo	A, X, Z	...
Camino	Camino(A, X, Z)	Recupera el camino del nodo con clave X al nodo con clave Z	A, X, Z	Reporta el camino de X a Z, si X es ancestro de Z; <b>Error en caso contrario.</b>
Altura	Altura(A)	Evalúa la altura de A	A	Reporta la altura de A.
InOrden	InOrden(A)	Procesa A en Inorden	A	Está sujeta al proceso que se realice sobre los elementos de A
PreOrden	PreOrden(A)	Procesa A en Preorden	A	Está sujeta al proceso que se realice sobre los elementos de A
PostOrden	PostOrden(A)	Procesa A en Postorden	A	Está sujeta al proceso que se realice sobre los elementos de A

# T.A.D. ABB

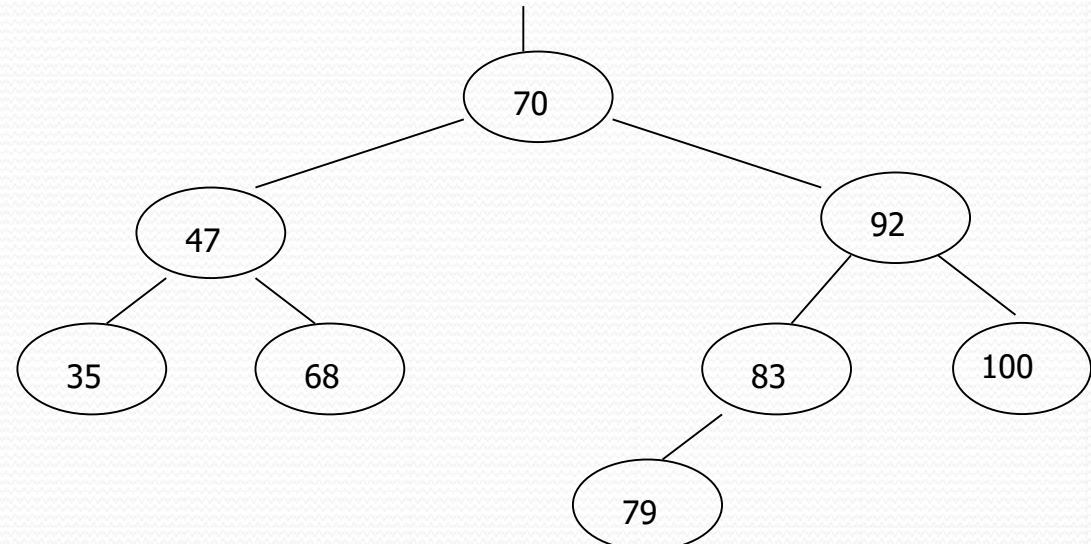
## Representación



# T.A.D. ABB

## Construcción de Operaciones Abstractas (1)

**Buscar(A,X)**



**Si** (sub)árbol vacío

**entonces Error- Elemento Inexistente**

**sino**

**Si** Clave [X] =Clave [R] (R:nodo raíz de (sub)árbol)

**entonces Éxito- Elemento existente**

**sino**

**Si** Clave [X] < Clave [R] (R:nodo raíz de (sub)árbol)

**entonces Buscar ((sub)árbol izquierdo( R), X)**

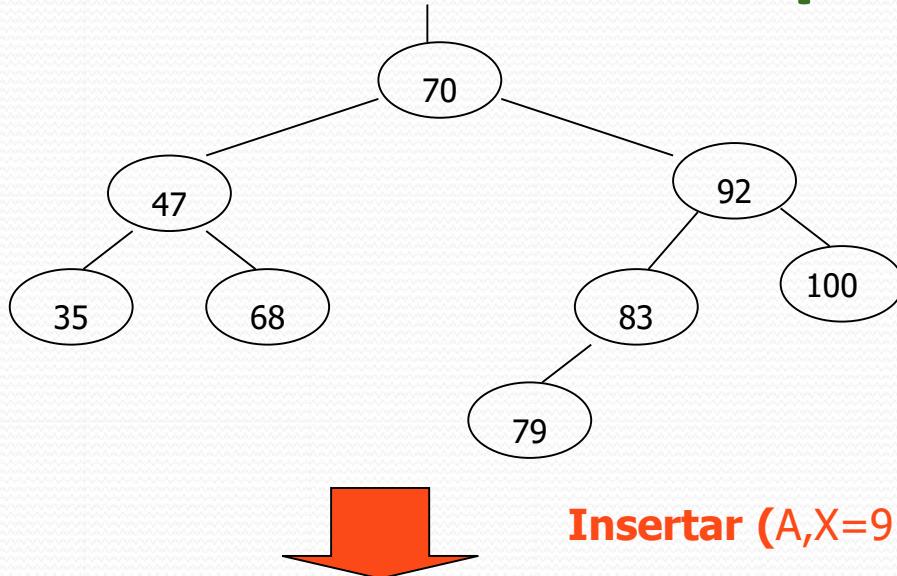
**sino**

**Si** Clave[X] >Clave [R] (R: nodo raíz de (sub)árbol)

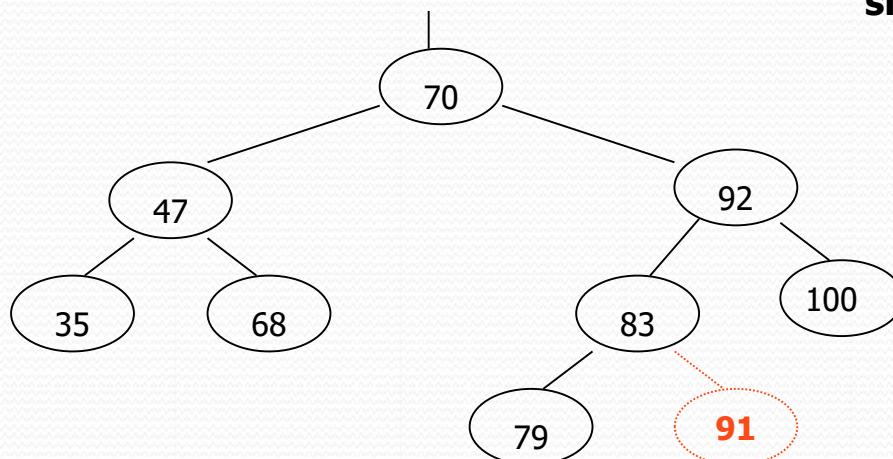
**entonces Buscar ((sub)árbol derecho(R), X)**

# T.A.D. ABB

## Construcción de Operaciones Abstractas (2)

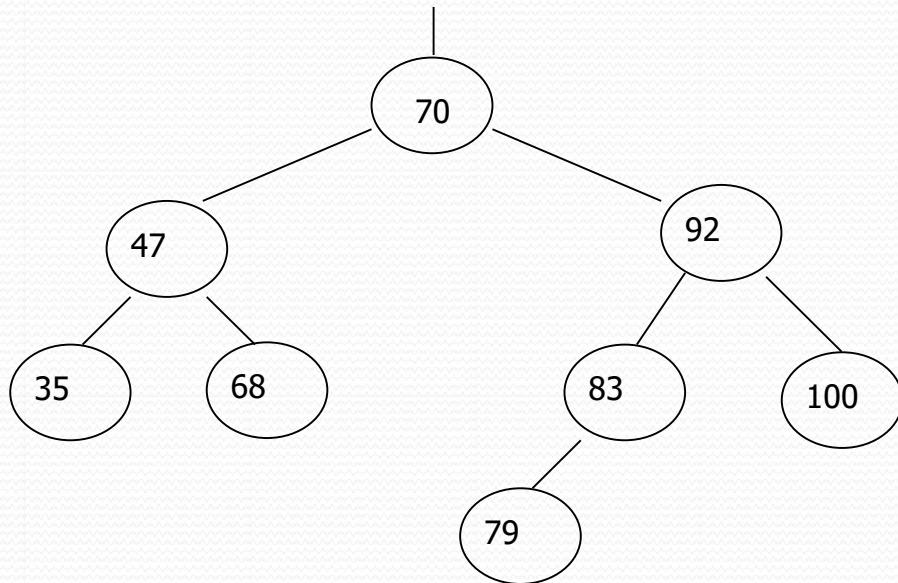


**Si** (sub)árbol vacío  
**entonces** Éxito- X nueva hoja  
**sino**  
    **Si** Clave [X] = Clave [R]  
    **entonces** Error- Elemento ya existente  
    **sino**  
        **Si** Clave [X] < Clave [R]  
        **entonces** Insertar((sub)árbol izquierdo( R),X)  
        **sino**  
            **Si** Clave[X] >Clave [R]  
            **entonces** Insertar ((sub)árbol derecho(R), X)

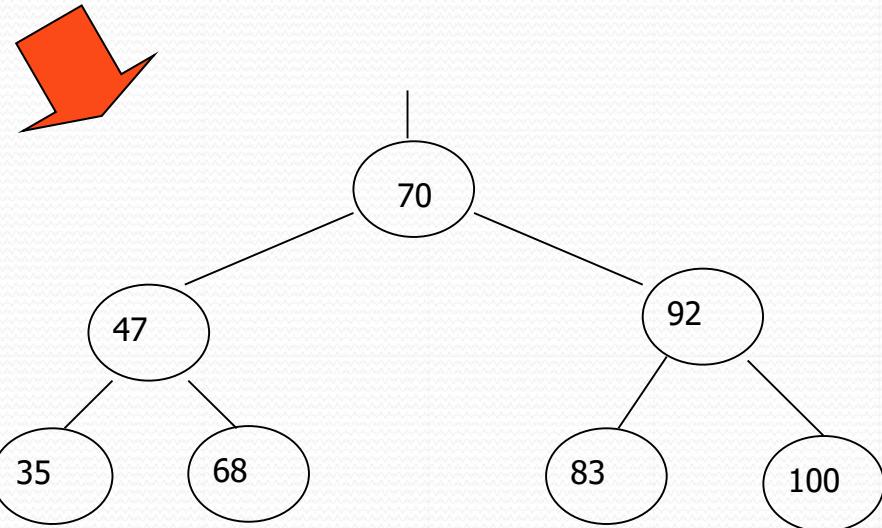


# T.A.D. ABB

## Construcción de Operaciones Abstractas (3)

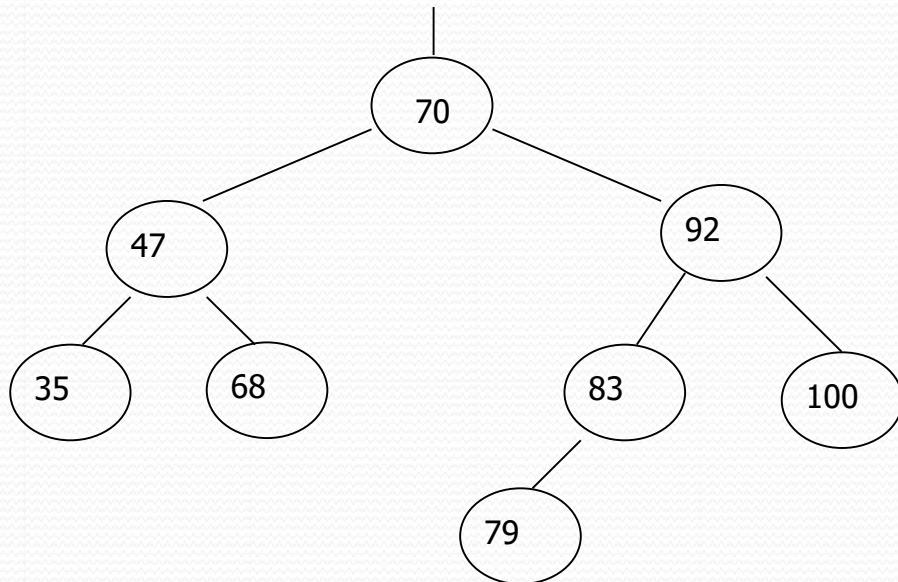


Suprimir (A , X=79)

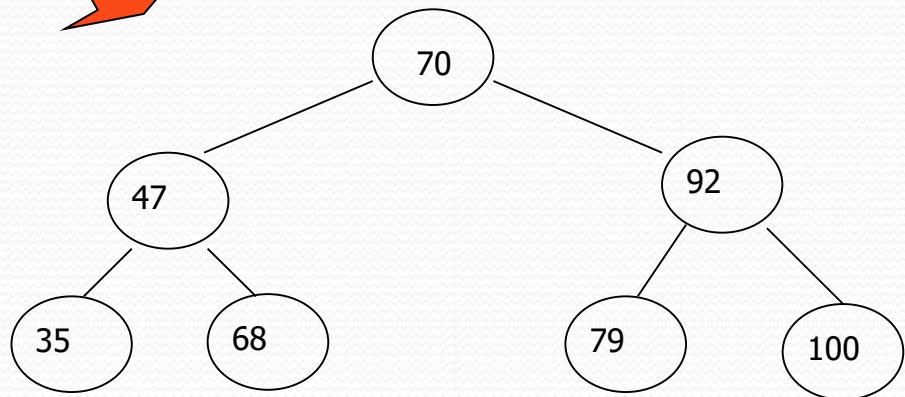


# T.A.D. ABB

## Construcción de Operaciones Abstractas (4)

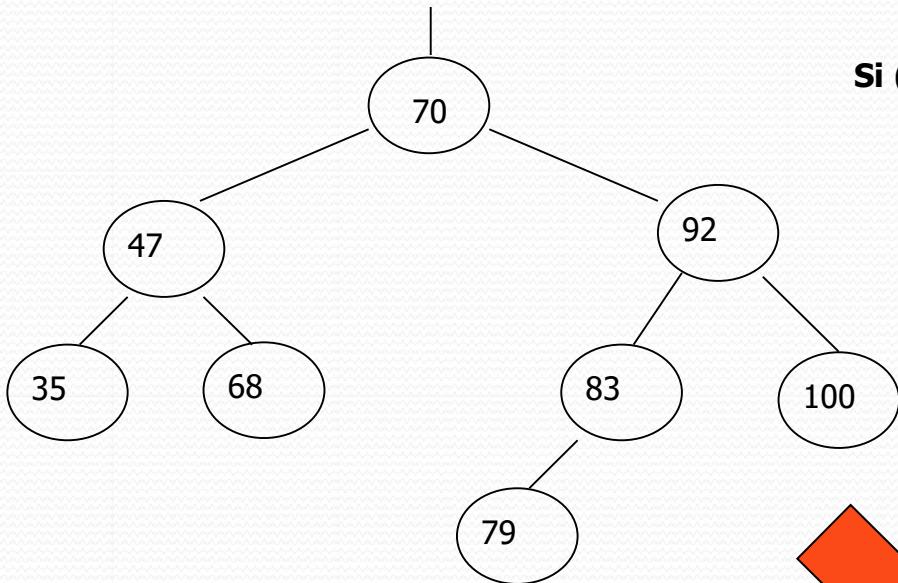


Suprimir (A , X=83)



# T.A.D. ABB

## Construcción de Operaciones Abstractas (5)



**Si** (sub)árbol vacío

**entonces Error** -elemento inexistente

**sino**

**Si** Clave [X] = Clave [R] (R: nodo raíz de (sub)árbol)

**entonces**

**Si** Grado(R) = 0

**entonces** eliminar nodo R

**Si** Grado(R) = 1

**entonces** Padre(R)  $\leftarrow$  Hijo(R)

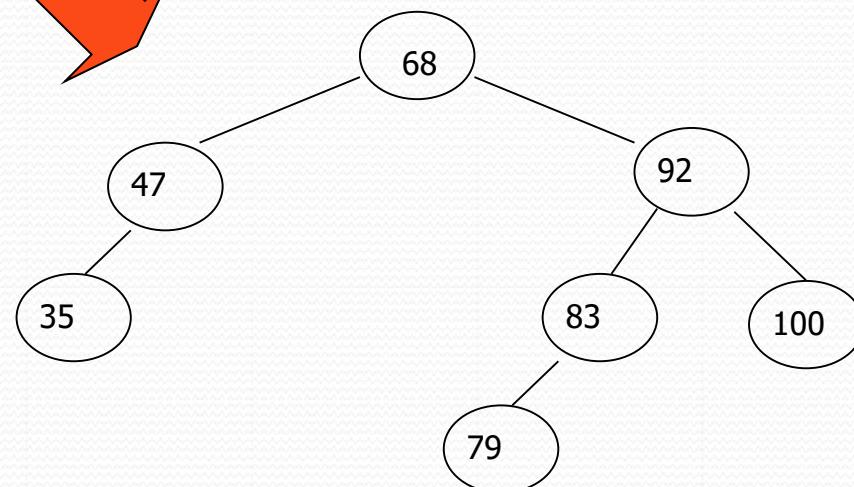
**Si** Grado(R) = 2

**entonces** R  $\leftarrow$  Máximo ((sub)árbol-izquierdo (R))

Eliminar (Máximo((sub)árbol-izquierdo(R)))



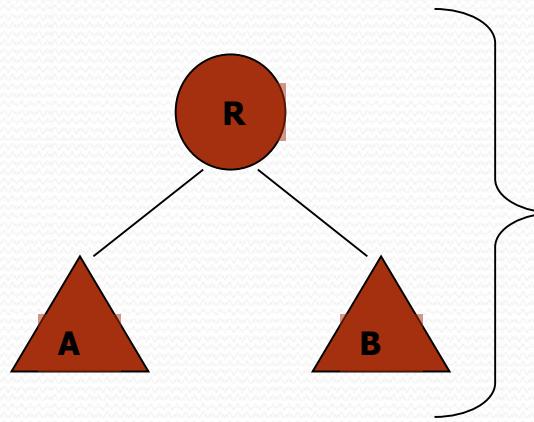
**Suprimir (A , X=70)**



# Árbol Binario

## Construcción de Operaciones Abstractas-Recorridos (6)

Recorridos



R: raiz

A: Sub Árbol izq de R

B: Sub Árbol der de R

**Preorden** (Árbol)

Si (Árbol no vacío)  
entonces

**Procesar nodo R**

**Preorden** ((Sub)Árbol izq  
de R)

**Preorden** ((Sub)Árbol der  
de R)

**Inorden** (Árbol)

Si (Árbol no vacío)  
entonces

**Inorden**((Sub) Árbol izq  
de R)

**Procesar nodo R**

**Inorden**((Sub) Árbol der  
de R)

**Postorden** (Arbol)

Si (Árbol no vacío)  
entonces

**Postorden**((Sub) Árbol izq  
de R)

**Postorden**((Sub) Árbol der  
de R)

**Procesar nodo R**

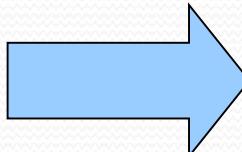
# T.A.D. ABB

## Aplicación

### Índice de referencias cruzadas

ENTRADA

- 1 . `El editor de lenguaje C
- 2 . presenta un entorno
- 3 . similar al entorno
- 4 . de lenguaje Pascal`



SALIDA

al	3
C	1
de	1, 4
editor	1
entorno	2, 3
el	1
Lenguaje	1, 4
Pascal	4
presenta	2
similar	3
un	2

# Árbol Binario

## Tipos

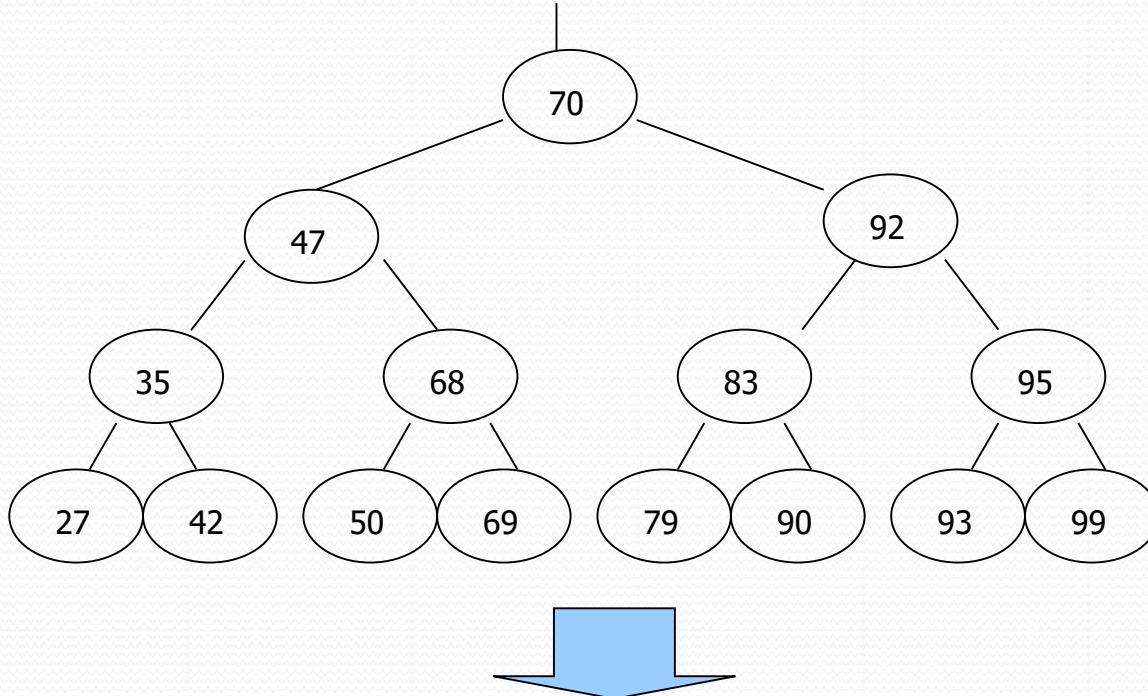
¿Cuántas comparaciones se realizan en una **búsqueda**, en el peor de los casos, en un **ABB** ?

**Las operaciones Insertar, Suprimir y Buscar se realizan en  $h$  accesos como máximo, siendo  $h$  la altura del árbol binario.**

- *Árbol Binario Relleno:* Es aquel árbol en el que todo nodo o bien es una hoja, o tiene dos hijos.
- *Árbol Binario Completo:* Es un árbol binario lleno en el que todas las hojas están en el mismo nivel.

¿Cuántas comparaciones se realizan en una **búsqueda**, en el peor de los casos, en un **ABB Completo**?

# ABB Completo

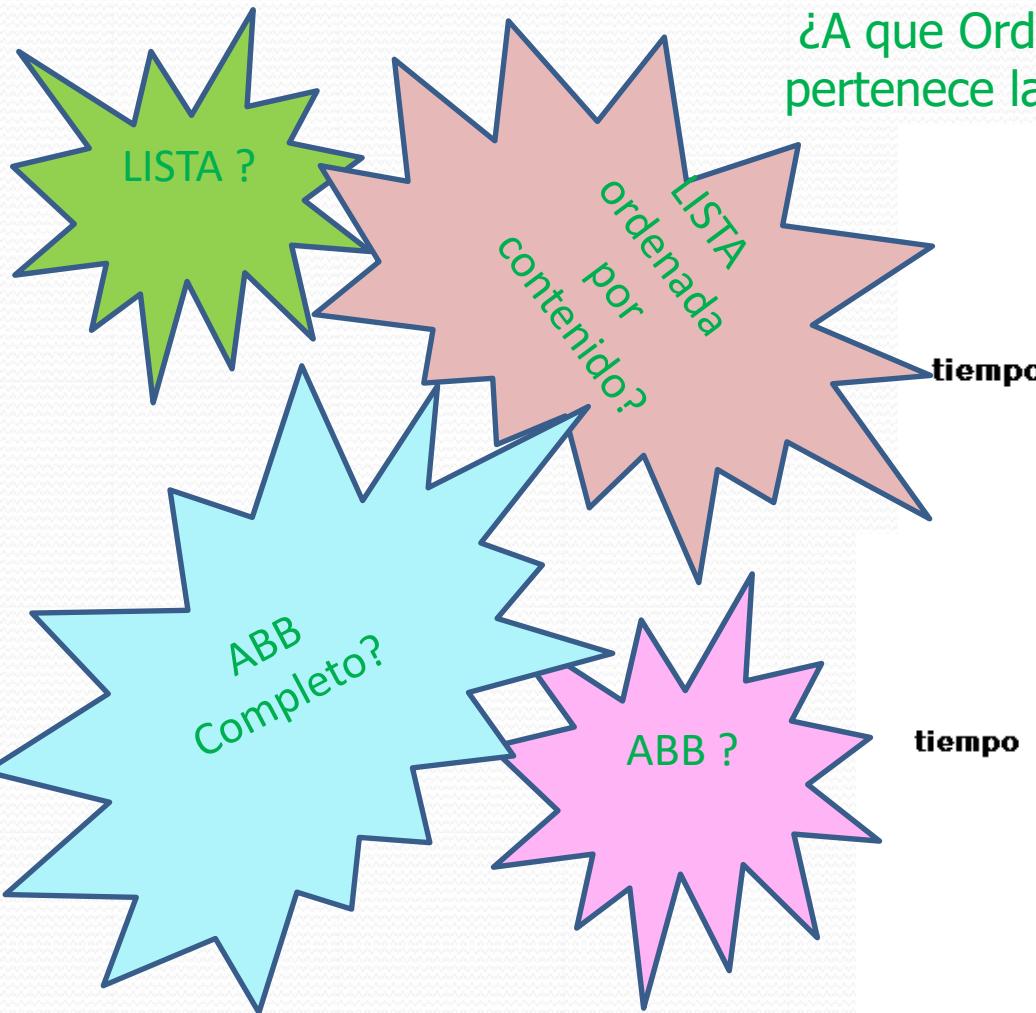


$$\begin{aligned}N &= 2^C - 1 \Rightarrow N + 1 = 2^C \\&\Rightarrow \log_2(N+1) = C (\log_2 2) \\&\Rightarrow \log_2(N+1) = C\end{aligned}$$

N	1	3	7	15
C	1	2	3	4

$t(\text{Buscar (ABB Completo)}) \in O(\log n)$

# Operaciones Abstractas: BUSCAR



¿A que Orden de Complejidad pertenece la operación Buscar?

