ÁRBOLES

CAPÍTULO 6

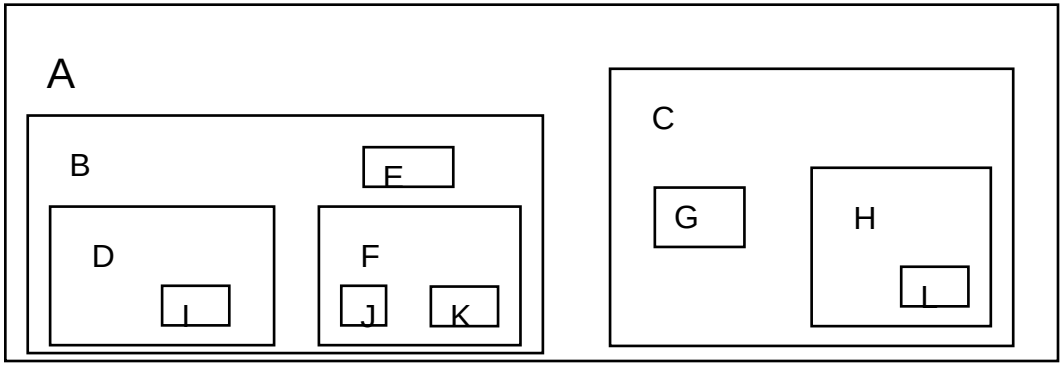
ÁRBOLES

- Desde el punto de vista conceptual, un árbol es un objeto que comienza con una raíz (root) y se extiende en varias ramificaciones o líneas (edges), cada una de las cuales puede extenderse en ramificaciones hasta terminar, finalmente en una hoja.
- Los árboles representan las estructuras no-lineales y dinámicas de datos más importantes en computación. Dinámicas, puesto que la estructura árbol puede cambiar durante la ejecución de un programa. No-lineales, puesto que a cada elemento del árbol pueden seguirle varios elementos.

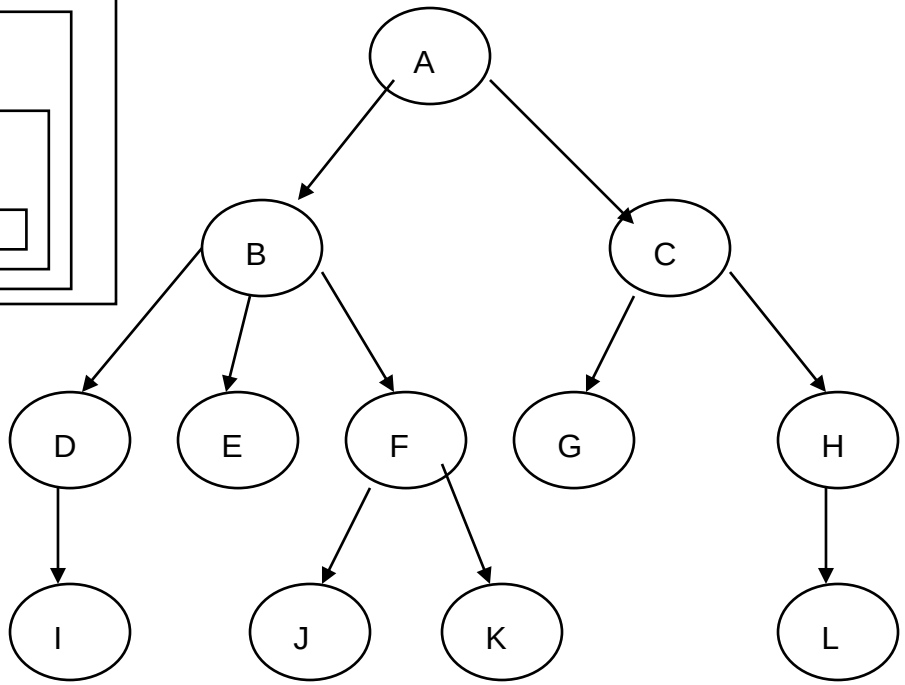
Propiedades

- En la ciencia de la computación definimos un **árbol como un conjunto de nodos y líneas**. Un **nodo es un elemento de información** que reside en el árbol. Una **línea es un par de nodos ordenados <u,v>**, y a la **secuencia de líneas se le denomina ruta (path).**
- Además, los árboles tienen las siguientes propiedades:
- Tienen un nodo al que se le llama raíz del árbol.
- Todos los nodos, excepto la raíz, tienen una sola línea de entrada (el nodo raíz no tiene ninguna).
- Existe una ruta única del nodo raíz a todos los demás nodos del árbol.
- Si hay una ruta <a,b>, entonces a ‘b’ se le denomina ‘hijo’ de ‘a’ y es el nodo raíz de un subárbol.

Gráficamente puede representarse una estructura árbol de diferentes maneras y todas ellas equivalentes:



Árbol por medio de diagramas de Venn



Árbol mediante grafo.

CARACTERÍSTICAS Y PROPIEDADES DE LOS ÁRBOLES.

1. * **NODO** indica un elemento, o ítem, de información.
2. * Todo árbol que no es vacío, tiene **un único nodo raíz.**
3. * Un nodo X es descendiente directo de un nodo Y, si el nodo X es apuntado por el nodo Y. **X es hijo de Y.**
4. * Un nodo X es antecesor directo de un nodo Y, si el nodo X apunta al nodo Y. **X es padre de Y.**
5. *Se dice que todos los nodos que son descendientes directos (hijos) de un mismo nodo (padre), son **hermanos.**
6. * Todo nodo que no tiene ramificaciones (hijos), se conoce con el nombre de **terminal u hoja.**
7. * Todo nodo que no es raíz, ni terminal u hoja se conoce con el nombre de **interior.**
8. *** Grado** es el número de descendientes directos de un determinado nodo. **Grado del árbol** es el máximo grado de todos los nodos del árbol.
9. * **Nivel** es el número de arcos que deben ser recorridos para llegar a un determinado nodo. Por definición, la raíz tiene nivel 1.
10. ***Altura** del árbol es el máximo número de niveles de todos los nodos del árbol.

Ejemplo

A es la raíz del árbol

B es hijo de A

A es padre de B

B y C son hermanos

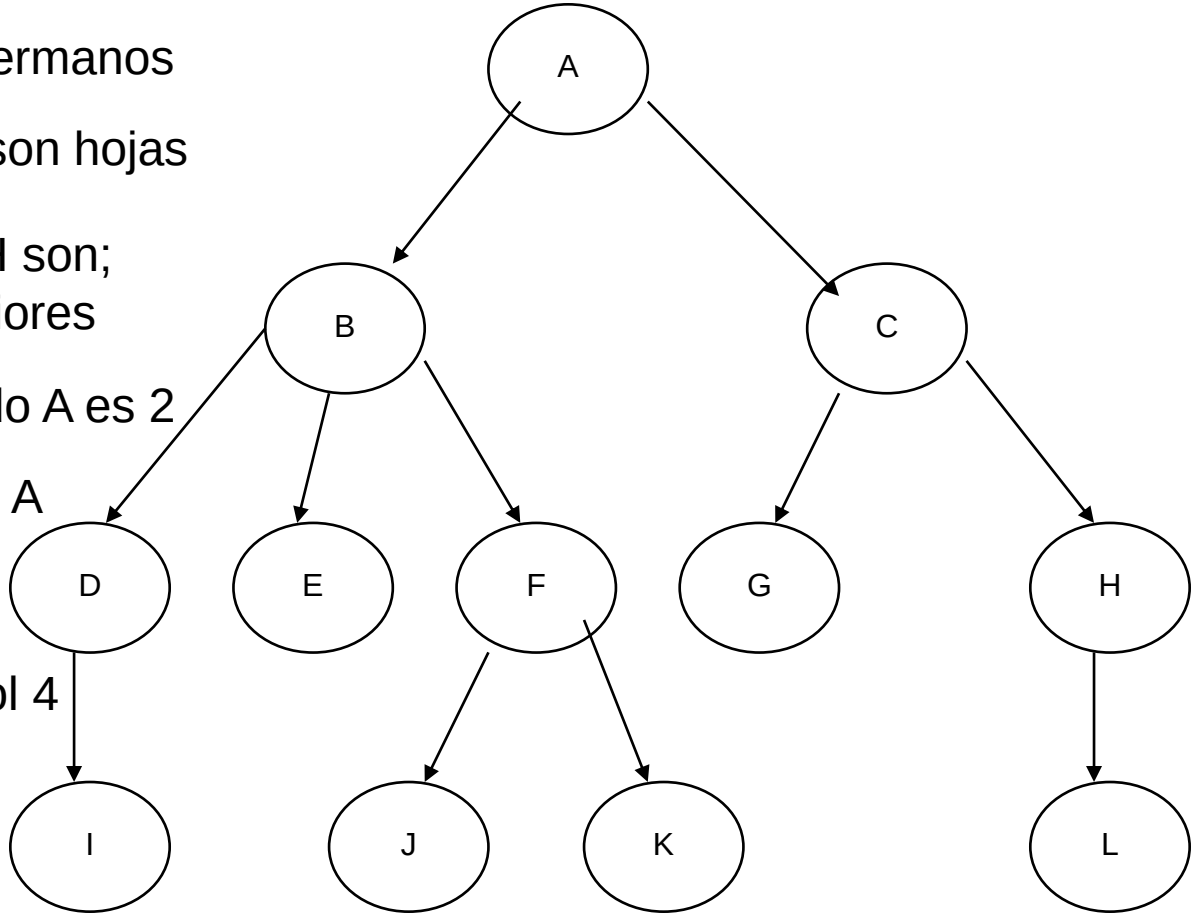
I,E,J,K,G,L son hojas

B, D, F, C, H son; nodos interiores

El grado de nodo A es 2

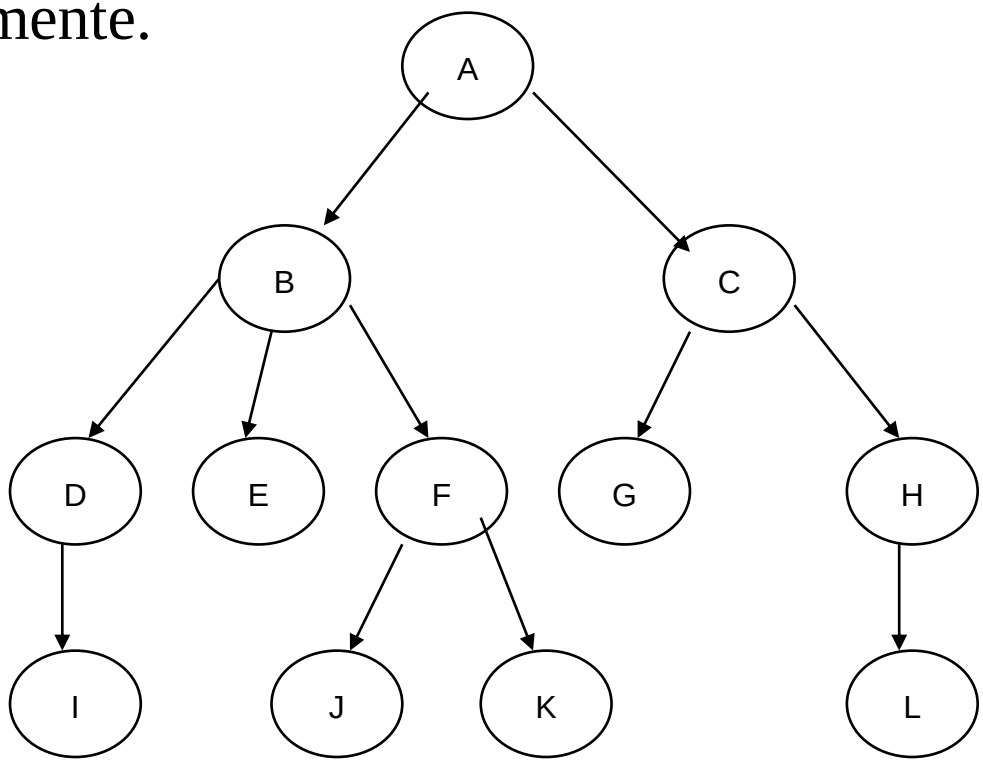
Nivel del nodo A es 1 (def)
Nivel B es 2

Altura del árbol 4



LONGITUD DE CAMINO INTERNO Y EXTERNO.

- Se define la longitud de camino X como el número de arcos que deben ser recorridos para llegar desde la raíz al nodo X. Por definición la raíz tiene longitud de camino 1, sus descendientes directos longitud de camino 2 y así sucesivamente.



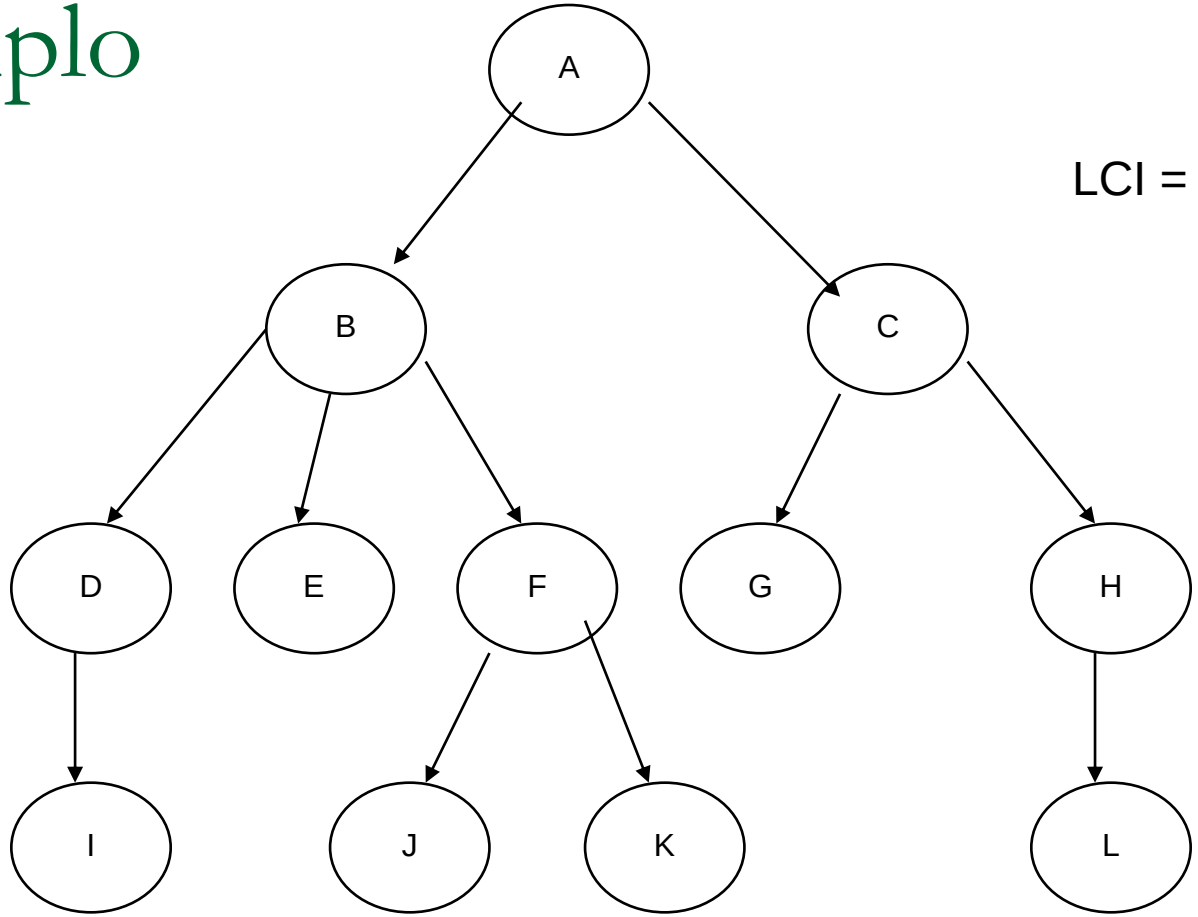
LONGITUD DE CAMINO INTERNO.

- La longitud de camino interno es la suma de las longitudes de camino de todos los nodos del árbol. Es importante por que permite conocer los caminos que tiene el árbol. Puede calcularse por medio de la siguiente fórmula:

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

- donde ‘i’ representa el nivel del árbol, ‘h’ su altura y ‘ni’ el número de nodos en el nivel ‘i’.

Ejemplo



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

La LCI del árbol anterior es: LCI= 1*1 + 2*2 + 5*3 + 4*4 = 36
h=4

MEDIA DE LA LONGITUD DE CAMINO INTERNO (LCIM)

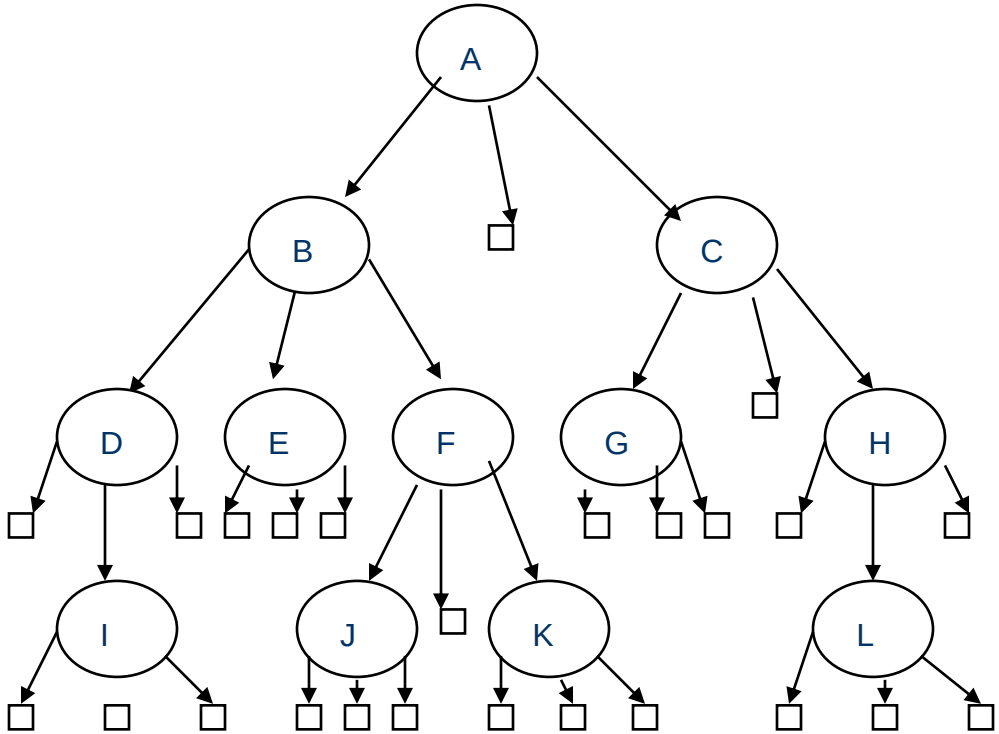
- Se calcula dividiendo la LCI entre el número de nodos del árbol (n).
- $$LCIM = LCI / n$$
- Y significa el número de arcos que deben ser recorridos en promedio para llegar, partiendo de la raíz, a un nodo cualquiera del árbol.
- La LCIM del árbol anterior es:
  - LCIM = 36 / 12 = 3

LONGITUD DE CAMINO EXTERNO.

- Primero definiremos los conceptos de:
- **Árbol extendido** es aquel en el que el número de hijos de cada nodo es igual al grado del árbol. Si alguno de los nodos del árbol no cumple con esta condición entonces debe incorporársele al mismo nodos especiales; tantos como sea necesario para satisfacer la condición.
- Los **nodos especiales** tienen como objetivo reemplazar las ramas vacías o nulas, no pueden tener descendientes y normalmente se representan con la forma de un cuadrado.

Ejemplo

El número de nodos especiales de este árbol es 25, del árbol de grado 3.



Definición LCE

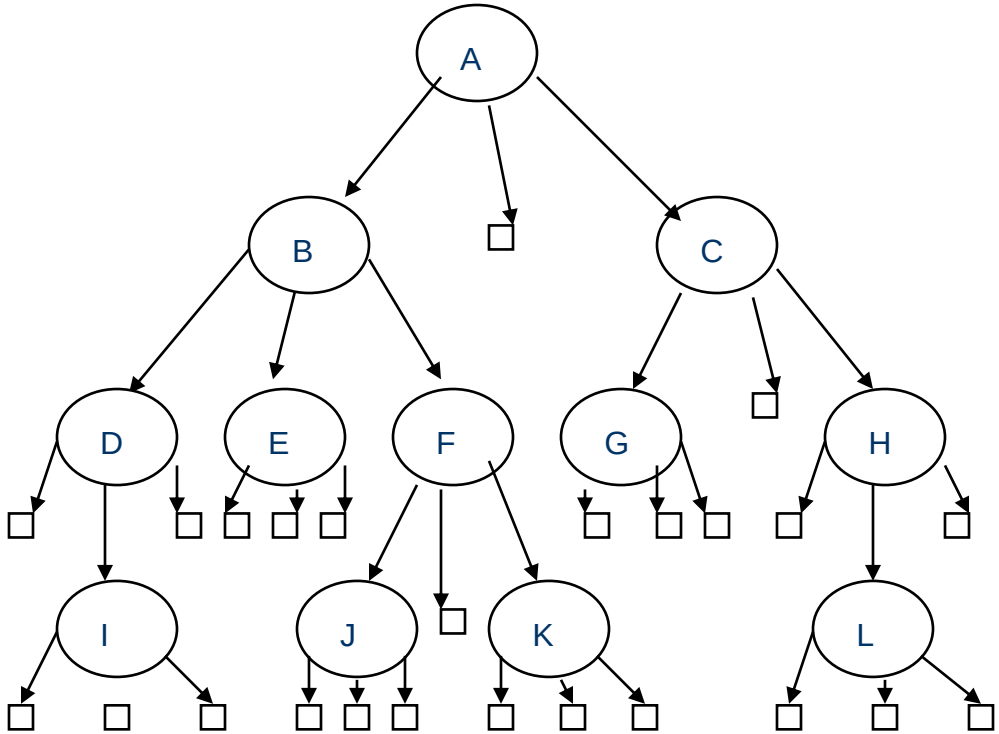
- Se puede definir ahora la longitud de camino externo como la suma de las longitudes de camino de todos los nodos especiales del árbol. Se calcula por medio de la siguiente fórmula:

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

- en donde ‘i’ representa el nivel del árbol, ‘h’ su altura y ‘nei’ el número de nodos especiales en el nivel ‘i’.

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

La LCE del árbol anterior es:
LCE = 1*2 + 1*3 + 11*4 + 12*5 = 109

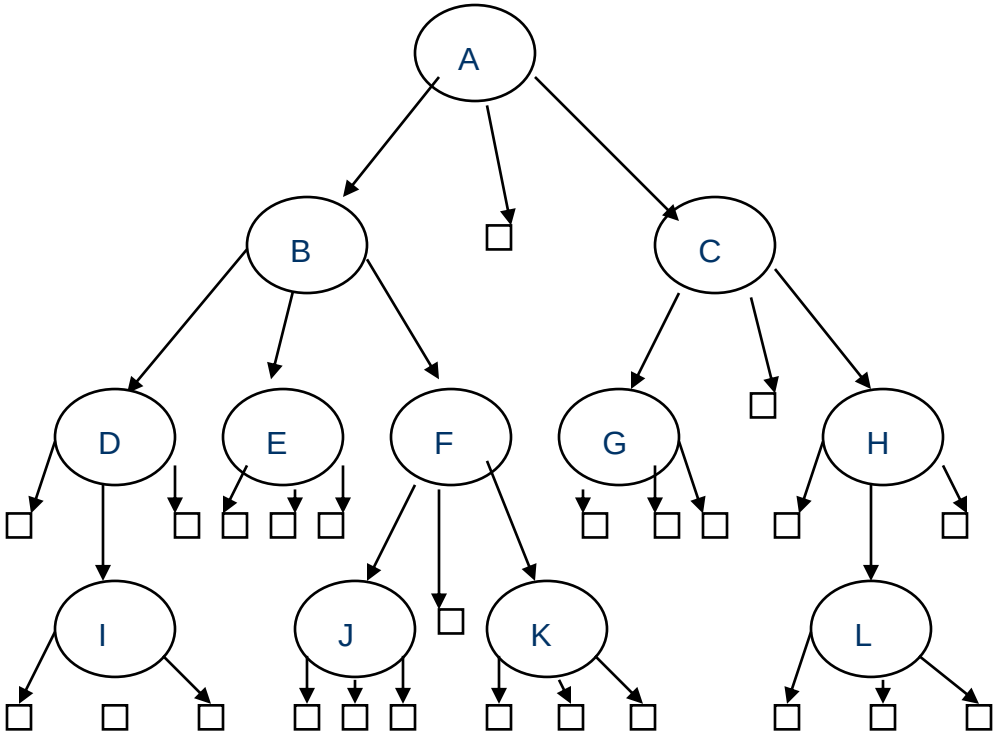


La media de la longitud de camino externo (LCEM)

- Ahora bien, se calcula dividiendo LCE entre el número de nodos especiales del árbol (ne).
- $$LCEM = LCE / ne$$
- Y significa el número de arcos que deben ser recorridos en promedio para llegar, partiendo desde la raíz, a un nodo especial cualquiera del árbol.

Ejemplo…

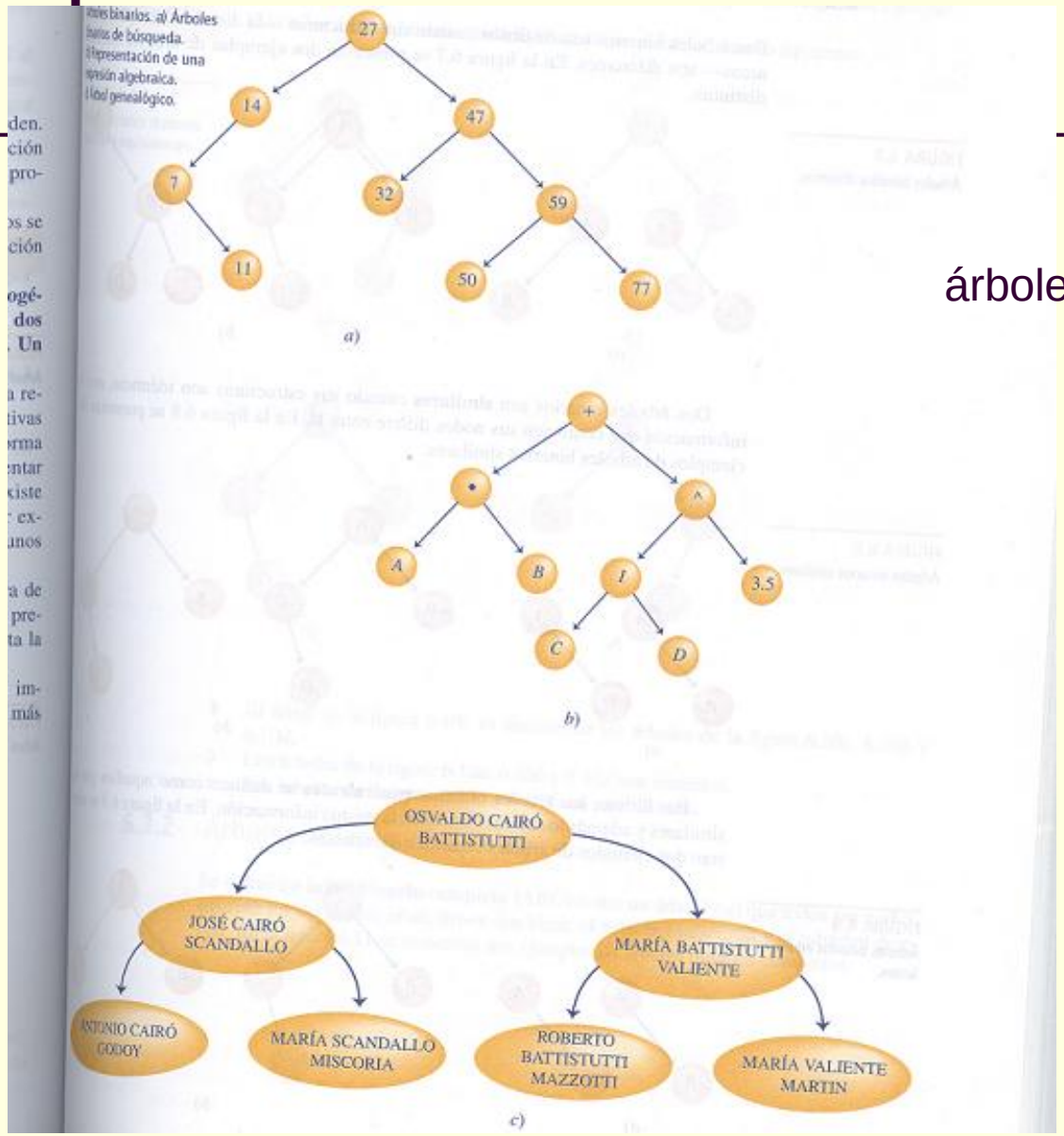
- Para nuestro árbol anterior,
  LCEM = 109 / 25 = 4.36



ÁRBOLES BINARIOS

- Un árbol ordenado es aquel en el cual la distribución de las ramas sigue cierto orden. Los árboles ordenados de grado 2 son de especial interés puesto que representan una de las estructuras de datos más importante en computación, conocida como árboles binarios.
- En un árbol binario cada nodo puede tener como máximo dos subárboles; y siempre es necesario distinguir entre el subárbol izquierdo y el subárbol derecho

Aplicaciones de árboles binarios



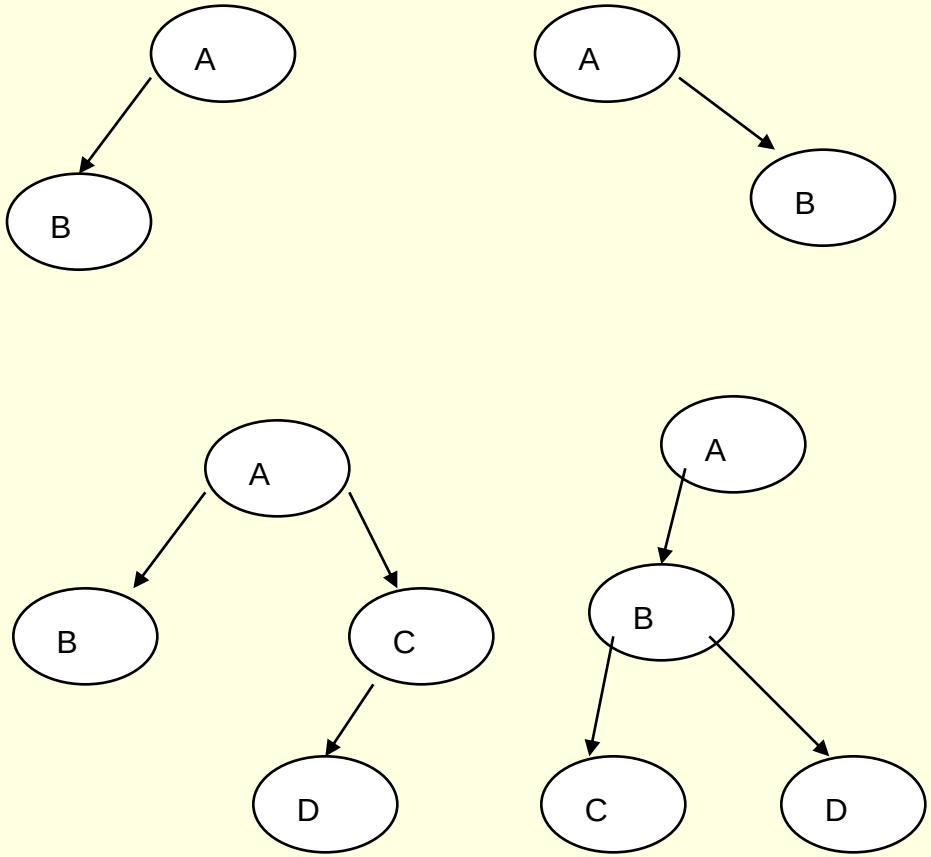
árboles binarios de búsqueda

representación de una expresión algebraica

árbol genealógico

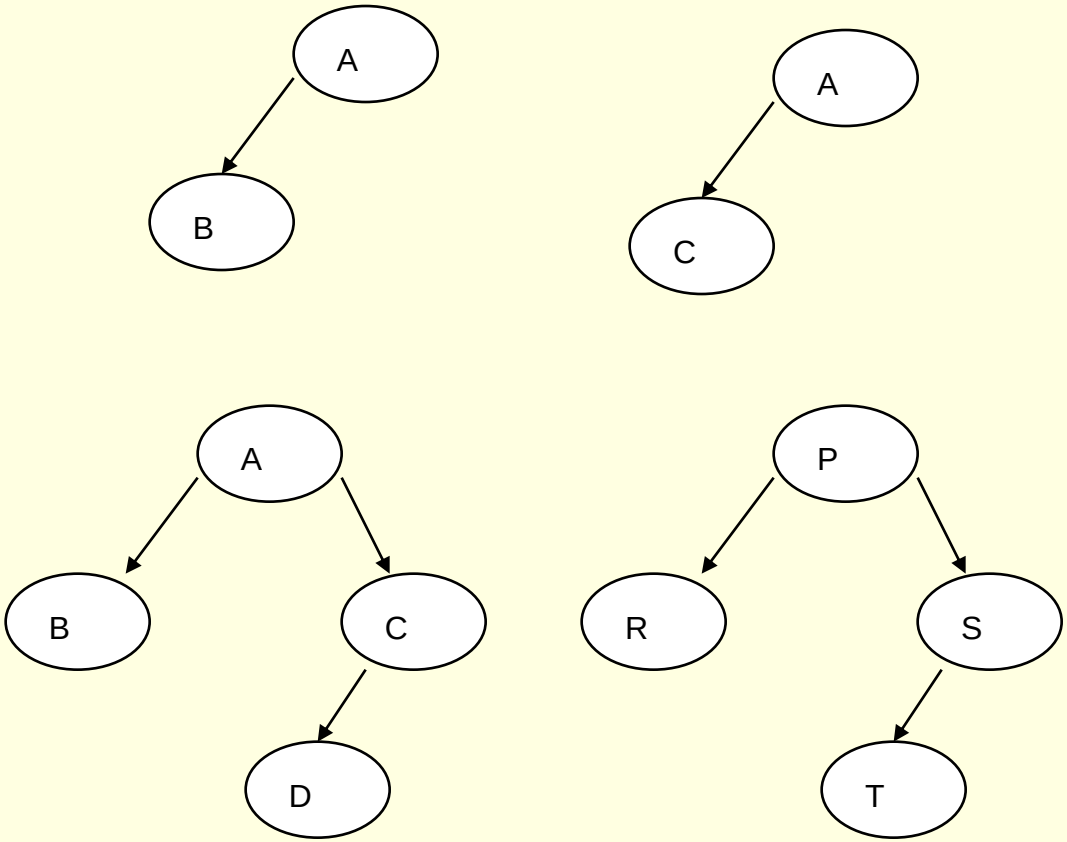
ÁRBOLES BINARIOS DISTINTOS, SIMILARES Y EQUIVALENTES.

- Dos árboles binarios son **distintos** cuando sus estructuras son diferentes. Ejemplo:



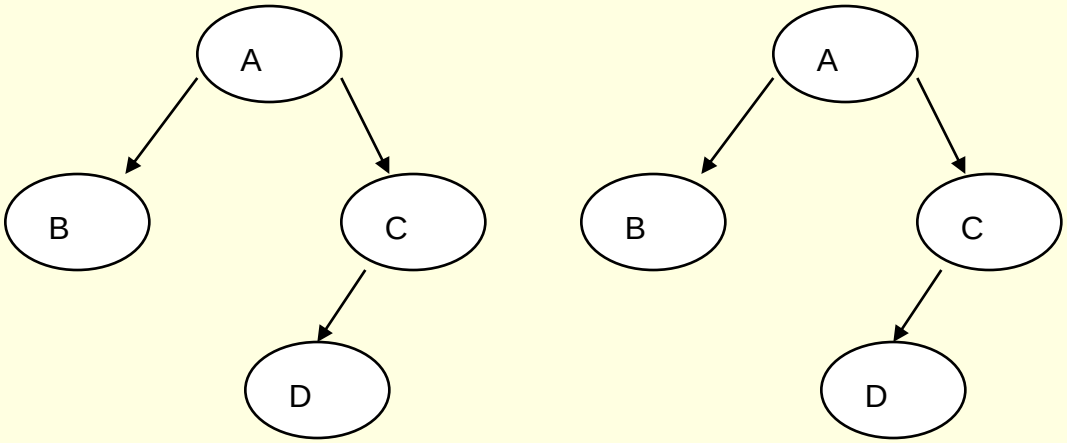
Similares

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



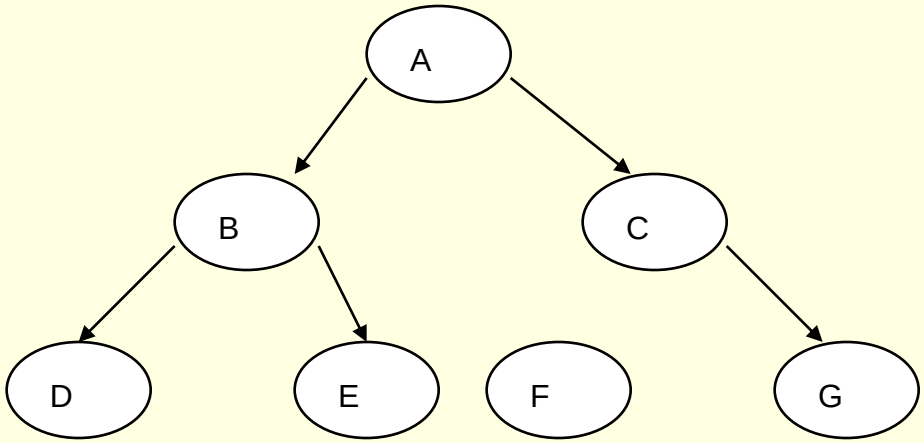
Equivalentes

- Los árboles binarios **equivalentes** se definen como aquellos que son similares y además los nodos contienen la misma información.



ÁRBOLES BINARIOS COMPLETOS

- Se define un árbol binario **completo** como un árbol en el que todos sus nodos, excepto los de último nivel, tienen dos hijos; el subárbol izquierdo y el subárbol derecho



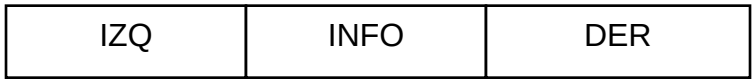
Árbol binario completo.

ÁRBOLES BINARIOS COMPLETOS

- Cabe aclarar que existen algunos autores que definen un árbol binario completo de otra forma; y otros que utilizan el término lleno para referirse a completo.
- Se puede calcular el número de nodos de un árbol binario completo de altura ‘h’, aplicando la siguiente fórmula:
- $$\text{NÚMERO DE NODOS}_{ABC} = 2^h - 1$$
- Donde ABC significa árbol binario completo, y ‘h’ la altura del árbol.

REPRESENTACIÓN DE ÁRBOLES BINARIOS EN MEMORIA.

- Existen dos formas de representar un árbol binario en memoria:
  - Por medio de punteros
  - Por medio de arreglos
- Aquí lo veremos por medio de punteros.
- Los nodos del árbol binario serán representados como registros, que contendrán como mínimo tres campos. En un campo se almacenará la información del nodo. Los dos restantes se utilizarán para apuntar a los subárboles izquierdo y derecho respectivamente del subnodo en cuestión.



- IZQ: Campo donde se almacena la dirección del subárbol izquierdo del nodo T.
- INFO: Campo donde se almacena la información de interés del nodo.
- DER: Campo donde se almacena la dirección del subárbol derecho del nodo T.

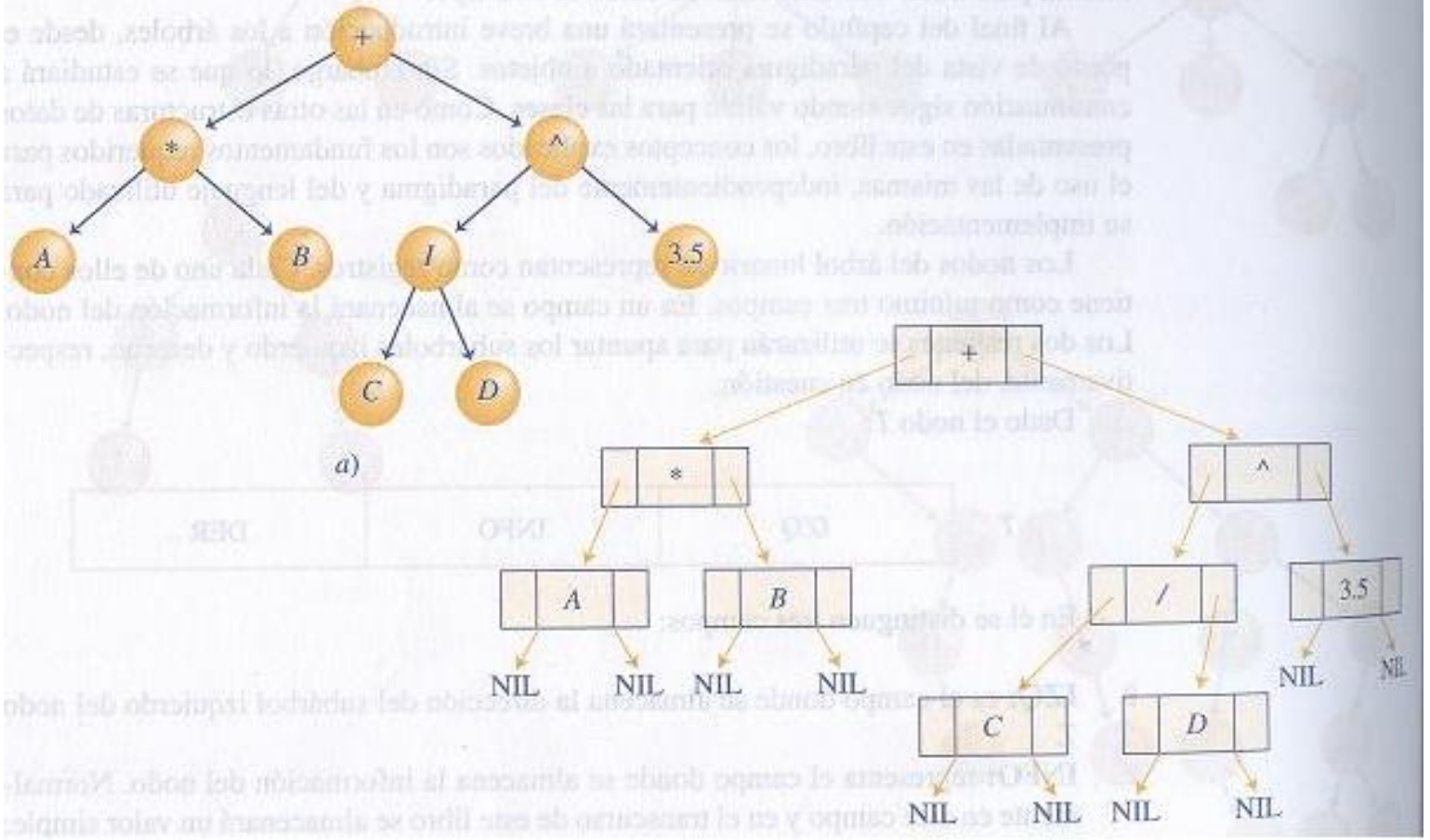
La definición de un árbol binario en lenguaje algorítmico:

- ENLACE= ^NODO
- NODO= REGISTRO
- IZQ: tipo ENLACE
- INFO: tipo de dato
- DER: tipo ENLACE
- Fin

Ejemplo

FIGURA 6.19

Representación de un árbol binario en memoria. *a*) Árbol binario. *b*) Su representación en memoria.



OPERACIONES EN ÁRBOLES BINARIOS

- Una de las operaciones básicas de un árbol binario es la creación del mismo en memoria.
- Algoritmo Crea árbol

Crea_árbol (APNODO)

{El algoritmo crea un árbol binario en memoria. APNODO es una variable de tipo ENLACE —puntero a un nodo—. La primera vez APNODO se crea en el programa principal}
{INFO, IZQ y DER son campos del registro NODO. INFO es de tipo carácter. IZQ y DER son de tipo puntero. Las variables RESP y OTRO son de tipo carácter y de tipo ENLACE, respectivamente}

1. Leer APNODO^.INFO {Lee la información y se guarda en el nodo}
2. Escribir "¿Existe nodo por izquierda: 1(Sí) – 0(No)?"
3. Leer RESP
4. *Si* (RESP = "Sí")
   entonces
      Crear(OTRO) {Se crea un nuevo nodo}
      Hacer APNODO^.IZQ ← OTRO
      Regresar a Crea_árbol con APNODO^.IZQ {Llamada recursiva}
   si no
      Hacer APNODO^.IZQ ← NIL
5. {Fin del condicional del paso 4}
6. Escribir "¿Existe nodo por derecha: 1(Sí) – 0(No)?"
7. Leer RESP
8. *Si* (RESP = "Sí")
   entonces
      Crear(OTRO) {Se crea un nuevo nodo}
      Hacer APNODO^.DER ← OTRO
      Regresar a Crea_árbol con APNODO^.DER {Llamada recursiva}
   si no
      Hacer APNODO^.DER ← NIL
9. {Fin del condicional del paso 8}

Programación OO
La clase árbol

Árbol	}Nombre de la Clase
Raíz = puntero a un nodo	}Atributos
Regresa_Raíz() Recorre() Inserta() Elimina()	} Métodos

RECORRIDOS EN ÁRBOLES BINARIOS.

- Una de las operaciones más importantes a realizar en un árbol binario es el recorrido de los mismos. Recorrer significa visitar los nodos del árbol en forma sistemática; de tal manera que todos los nodos del mismo sean visitados una sola vez. Existen tres formas diferentes de efectuar el recorrido y todas ellas de naturaleza recursiva, éstas son:

Recorridos

- **Recorrido en preorden**
  - Visitar la raíz
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
- **Recorrido en inorden**
  - Recorrer el subárbol izquierdo
  - Visitar la raíz
  - Recorrer el subárbol derecho
- **Recorrido en postorden**
  - Recorrer el subárbol izquierdo
  - Recorrer el subárbol derecho
  - Visitar la raíz
- El termino visitar puede ser reemplazado por escribir la información el nodo.

Ejemplos

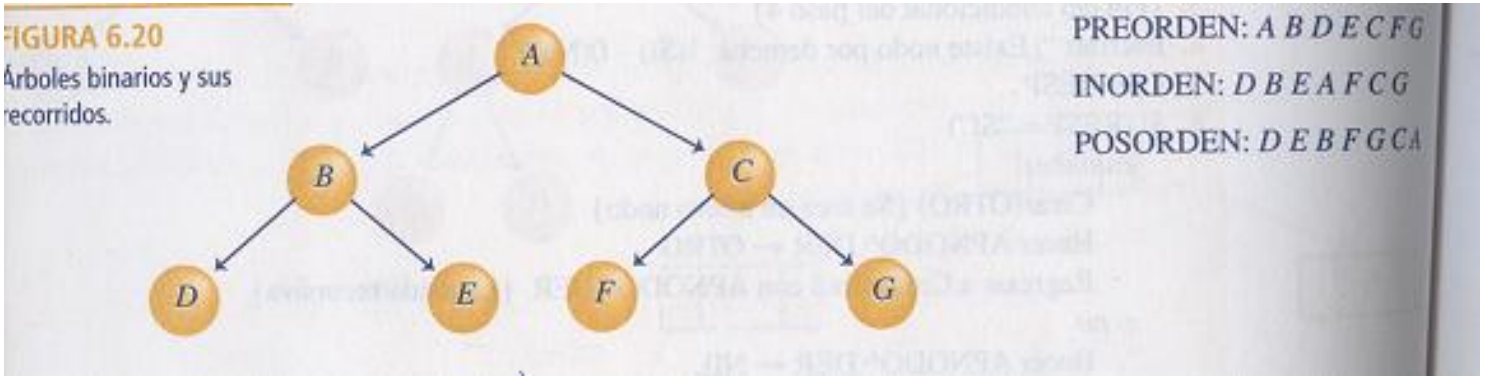
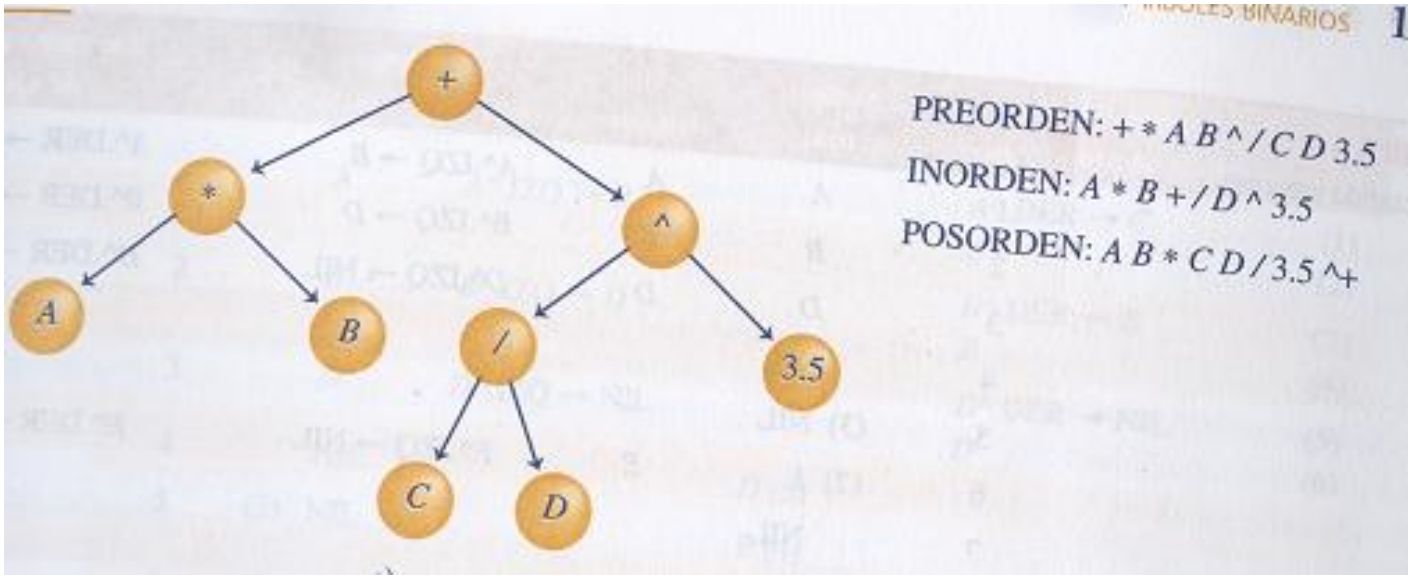


FIGURA 6.20
Árboles binarios y sus recorridos.
A
B
C
D
E
F
G
PREORDEN: A B D E C F G
INORDEN: D B E A F C G
POSORDEN: D E B F G C A


+
*
^
A
B
/
3.5
C
D
PREORDEN: + * A B ^ / C D 3.5
INORDEN: A * B + / D ^ 3.5
POSORDEN: A B * C D / 3.5 ^ +

ALGORITMO PREORDEN

- PREORDEN(NODO)
- {El algoritmo realiza el recorrido preorden en un árbol binario. NODO es un dato de tipo PUNTERO}
- {INFO, IZQ y DER son campos del registro NODO. INFO es una o más variables de información del nodo, IZQ y DER son variables de tipo puntero}
- Si NODO i= NILL entonces
- Visitar el NODO
  {Escribir la información NODO^.INFO}
  Regresa a PREORDEN con PREORDEN(NODO^.IZQ)
  {Llamada recursiva con la rama izquierda}
- Regresa a PREORDEN con PREORDEN(NODO^.DER)
  {Llamada recursiva con la rama derecha}
- Fin-si
- Fin-algoritmo

ALGORITMO INORDEN

- INORDEN(NODO)
- {El algoritmo realiza el recorrido inorden en un árbol binario. NODO es un dato de tipo PUNTERO}
- {INFO, IZQ y DER son campos del registro NODO. INFO es una o más variables de información del nodo, IZQ y DER son variables de tipo puntero}
- Si NODO ¡= NILL entonces
- INORDEN(NODO^.IZQ) {Llamada recursiva con la rama izquierda}
- Visitar el NODO {Escribir la información NODO^.INFO}
- INORDEN(NODO^.DER) {Llamada recursiva con la rama derecha}
- Fin-si
- Fin-algoritmo

ALGORITMO POSTORDEN

- POSTORDEN(NODO)
- {El algoritmo realiza el recorrido postorden en un árbol binario. NODO es un dato de tipo PUNTERO}
- {INFO, IZQ y DER son campos del registro NODO. INFO es una o más variables de información del nodo, IZQ y DER son variables de tipo puntero}
- Si NODO ¡= NILL entonces
- POSTORDEN(NODO^.IZQ) {Llamada recursiva con la rama izquierda}
- POSTORDEN(NODO^.DER) {Llamada recursiva con la rama derecha}
- Visitar el NODO {Escribir la información NODO^.INFO}
- Fin-si
- Fin-algoritmo.

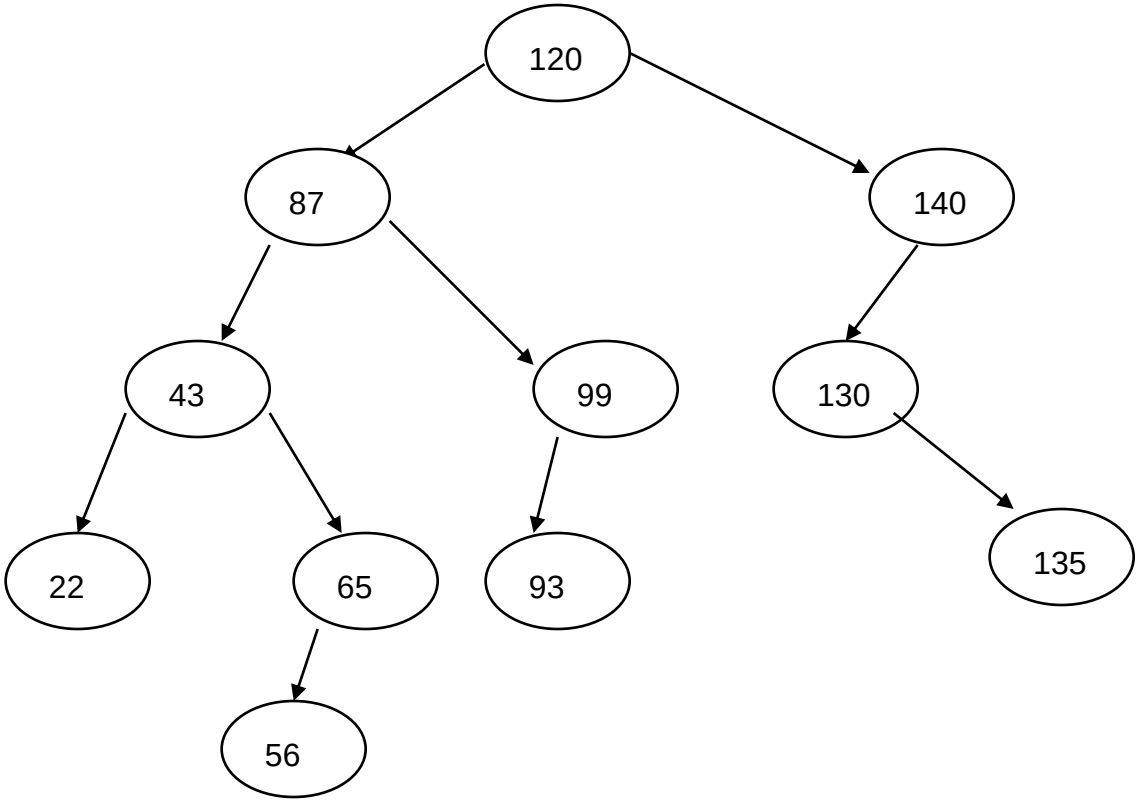
Debe recordarse que antes de recorrer un árbol, debe cargarse en memoria. Un algoritmo muy simple que cargue los nodos de un árbol binario en memoria, es el que sigue:

- **CARGA (NODO)**
- **{ El algoritmo carga los nodos de un árbol binario en memoria. NODO es una variable de tipo puntero. La primera vez NODO es creado en el programa principal}**
- **Leer información(INFOR)**
- **Hacer NODO^.INFO ← INFOR**
- **Escribir "¿Existe nodo por la izquierda ?"**
- **Leer respuesta**
- **Si respuesta == "si" entonces**
- **CREA(OTRO) {Crea un nuevo nodo, OTRO es de tipo puntero}**
- **Hacer NODO^.IZQ ← OTRO**
- **CARGA(NODO^.IZQ) {Llamada recursiva}**
- **Si no**
- **Hacer NODO^.IZQ ← NILL**
- **Fin si**
- **Escribir "¿Existe nodo por la derecha ?"**
- **Leer respuesta**
- **Si respuesta == "si" entonces**
- **CREA(OTRO) {Crea un nuevo nodo, OTRO es de tipo puntero}**
- **Hacer NODO^.DER ← OTRO**
- **CARGA(NODO^.DER) {Llamada recursiva}**
- **Si no**
- **Hacer NODO^.DER ← NILL**
- **Fin si**
- **Fin Algoritmo**

ÁRBOLES BINARIOS DE BÚSQUEDA.

- El árbol binario de búsqueda es una estructura sobre la cual se pueden realizar eficientemente las operaciones de búsqueda, inserción y eliminación.
- Formalmente se define un árbol binario de búsqueda de la siguiente manera: “Para todo nodo T del árbol debe cumplirse que todos los valores de los nodos del subárbol izquierdo de T deben ser menores o iguales al valor del nodo T. De forma similar, todos los valores de los nodos el subárbol derecho de T deben ser mayores o iguales al valor del nodo T”.

ÁRBOLES BINARIOS DE BÚSQUEDA.



Observemos que si se efectúa un recorrido in-orden sobre el árbol de búsqueda se obtendrá una clasificación de los nodos en forma ascendente.
El recorrido in-orden del árbol anterior produce el siguiente resultado:
22 43 56 65 87 93 99 120 130 135 140

Algoritmo de Búsqueda

- BÚSQUEDA (NODO, INFOR)
- {El algoritmo localiza un nodo en un árbol binario de búsqueda. NODO es una variable de tipo puntero que apunta a la raíz del árbol. INFOR es una variable que contiene la información que se desea localizar en el árbol. Cabe aclarar que la primera vez la variable NODO no puede ser vacía.}
- Si INFOR < NODO^.INFO entonces
- Si NODO^.IZQ = NILL entonces
- Escribir “El nodo no se encuentra en el árbol”
- Sino
- BÚSQUEDA (NODO^.IZQ, INFOR) {Llamada recursiva}
- Fin Si
- Sino
- Si INFOR > NODO^.INFO entonces
- Si NODO^.DER = NILL entonces
- Escribir “El nodo no se encuentra en el árbol”
- Sino
- BÚSQUEDA (NODO^.DER, INFOR) {Llamada recursiva}
- Fin Si
- Sino
- Escribir “El nodo se encuentra en el árbol”
- Fin Si
- Fin Si
- Fin del algoritmo

Otra forma de escribir el algoritmo de búsqueda:

- Algoritmo de Búsueda
- BÚSQUEDA1 (NODO, INFOR)
- {El algoritmo localiza un nodo en un árbol binario de búsqueda. NODO es una variable de tipo puntero que apunta a la raíz del árbol. INFOR es una variable que contiene la información que se desea localizar en el árbol.}
- Si NODO ¡= NILL entonces
- Si INFOR < NODO^.INFO entonces
- BÚSQUEDA (NODO^.IZQ, INFOR) {Llamada recursiva}
- Si no
- Si INFOR > NODO^.INFO entonces
- BÚSQUEDA (NODO^.DER, INFOR) {Llamada recursiva}
- Sino
- Escribir “El nodo se encuentra en el árbol”
- Fin Si
- Fin Si
- Si no
- Escribir “El nodo no se encuentra en el árbol”
- Fin si
- Fin del Algoritmo

Ejemplos

Supongamos que se desea localizar la clave 23 en el árbol binario. (P) pasos, (C) comparaciones

P	C	Preguntas y acciones
1	1	¿Es 93 < 120?
		Sí. ¿Es el subárbol izquierdo de 120 (87) = NIL?
	2	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 120 (87) e INFOR
	3	¿Es 93 < 87?
2	4	No. ¿Es 93 > 87?
		Sí. ¿Es el subárbol derecho de 87 (99) = NIL?
	5	No. Entonces se regresa a Búsqueda con el subárbol derecho de 87 (99) e INFOR
	6	¿Es 93 < 99?
3		Sí. ¿Es el subárbol izquierdo de 99 (93) = NIL?
	7	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 99 (93) e INFOR
	8	¿Es 93 < 93?
4	9	No. ¿Es 93 > 93?
		No. Entonces ÉXITO

Ejemplo 2
localiza la clave 123

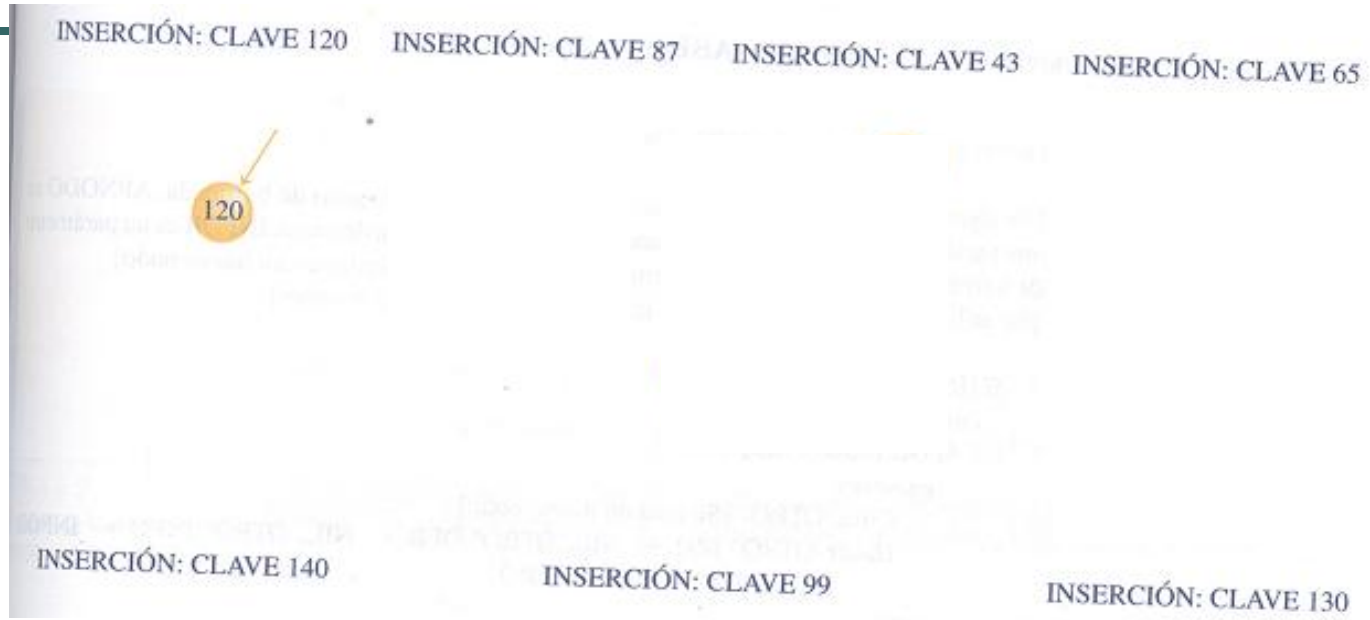
P	C	Preguntas y acciones
	1	¿Es 123 < 120?
1	2	No. ¿Es 123 > 120?
		Sí. ¿Es el subárbol derecho de 120 (140) = NIL?
	3	No. Entonces se regresa a Búsqueda con el subárbol derecho de 120 (140) e INFOR
	4	¿Es 123 < 140?
2		Sí. ¿Es el subárbol izquierdo de 140 (130) = NIL?
	5	No. Entonces se regresa a Búsqueda con el subárbol izquierdo de 140 (130) e INFOR
	6	¿Es 123 < 130?
3	7	Sí. ¿Es el subárbol izquierdo de 130 (NIL) = NIL?
		Sí. Entonces FRACASO

INSERCIÓN EN UN ÁRBOL BINARIO DE BÚSQUEDA.

- Los pasos que deben realizarse para insertar un elemento a un árbol binario de búsqueda son los siguientes:
- 1.- Debe compararse la clave a insertar con la raíz del árbol. Si es mayor, debe avanzarse hacia el subárbol derecho. Si es menor, debe avanzarse hacia el subárbol izquierdo.
- 2.- Repetir sucesivamente el paso 1 hasta que se cumpla alguna de las siguientes condiciones:
- 2.1 El subárbol derecho es igual a vacío, o el subárbol izquierdo es igual a vacío; en cuyo caso se procederá a insertar el elemento en el lugar que le corresponde.
- 2.2 La clave que quiere insertarse es igual a la raíz del árbol; en cuyo caso no se realiza la inserción.

Insertar las claves siguientes en un árbol binario de búsqueda;

120 – 87 – 43 – 65 – 140 – 99 – 130 – 22 -56



ALGORITMO INSERCIÓN1 (NODO, INFOR)

- { El algoritmo realiza la inserción de un elemento en un árbol binario de búsqueda. NODO es una variable de tipo puntero. INFOR contiene la información del elemento que se quiere insertar.}{Se utiliza como auxiliar la variable OTRO de tipo putero}
- Si INFOR < NODO^.INFO entonces
- SI (NODO^.IZQ==NIL)
- CREA(OTRO) {Crea un nuevo NODO)
- OTRO^.IZQ = NIL
- OTRO^.DER = NIL
- OTRO^.INFO = INFOR
- NODO^IZQ = OTRO
- SI NO
- INSERCIÓN1(NODO^.IZQ, INFOR) {Llamada recursiva}
- FIN SI
- Si no
- Si INFOR > NODO^.INFO entonces
- SI (NODO^.DER==NIL)
- CREA(OTRO) {Crea un nuevo NODO)
- OTRO^.IZQ = NIL
- OTRO^.DER = NIL
- OTRO^.INFO = INFOR
- NODO^DER = OTRO
- SI NO
- INSERCIÓN1(NODO^.DER, INFOR) {Llamada recursiva}
- FIN SI
- Si no
- Escribir “El nodo ya se encuentra en el árbol”
- Fin Si
- Fin Si
- Fin del Algoritmo

BORRADO EN UN ÁRBOL BINARIO DE BÚSQUEDA.

- Consiste en eliminar un nodo del árbol sin violar los principios que definen justamente un árbol binario de búsqueda. Se debe distinguir los siguientes casos:
- 1.- Si el elemento a borrar es terminal u hoja, simplemente se suprime.
- 2.- Si el elemento a borrar tiene un solo descendiente, entonces tiene que sustituirse por ese descendiente.
- 3.- Si el elemento a borrar tiene dos descendientes, entonces se tiene que sustituir por el nodo que se encuentra más a la izquierda en el subárbol derecho o por el nodo que se encuentra más a la derecha en el subárbol izquierdo.
- Además, debe recordarse que antes de eliminar un nodo, debe localizársele en el árbol.

Eliminación_ABB (APNODO, INFOR)

{El algoritmo realiza la eliminación de un elemento en un árbol binario de búsqueda. APNODO es una variable, por referencia, de tipo ENLACE. INFOR es un parámetro de tipo entero que contiene la información del nodo que se desea eliminar}
{AUX, AUXI y OTRO son variables auxiliares de tipo puntero. BO es una variable de tipo booleano}

```
1. Si (APNODO ≠ NIL)
     entonces
   1.1 Si (INFOR < APNODO^.INFO)
         entonces
           Regresar a Eliminación_ABB con APNODO^.IZQ e INFOR
         si no
   1.1.1 Si (INFOR > APNODO^.INFO)
           entonces
             Regresar a Eliminación_ABB con APNODO^.DER e INFOR
           si no
             Hacer OTRO ← NODO
         1.1.1.1 Si (OTRO^.DER = NIL)
                   entonces
                     Hacer APNODO ← OTRO^.IZQ
                   sino
               1.1.1.1.1 Si (OTRO^.IZQ = NIL)
                           entonces
                             Hacer APNODO ← OTRO^.DER
                           sino
                             Hacer AUX ← APNODO^.IZQ y BO ← FALSO
                     1.1.1.1.A Mientras (AUX^.DER ≠ NIL) Repetir
                                 Hacer AUX1 ← AUX, AUX ← AUX^.DER
                                      y BO ← VERDADERO
                     1.1.1.1.B {Fin del ciclo del paso 1.1.1.1.A}
                               Hacer APNODO^.INFO ← AUX^.INFO y
                                    OTRO ← AUX
                     1.1.1.1.C Si (BO = VERDADERO)
                                 entonces
                                   Hacer AUX1^.DER ← AUX^.IZQ
                                 sino
                                   Hacer APNODO^.IZQ ← AUX^.IZQ
                   1.1.1.1.D {Fin del condicional del paso 1.1.1.1.C}
               1.1.1.1.2 {Fin del condicional del paso 1.1.1.1.1}
           1.1.1.2 {Fin del condicional del paso 1.1.1.1}
                   Quitar (OTRO) {Se libera el espacio de memoria}
       1.1.2 {Fin del condicional del paso 1.1.1}
   1.2 {Fin del condicional del paso 1.1}
   sino
       Escribir "La información a eliminar no se encuentra en el árbol"
{Fin del condicional del paso 1}
```

Eliminación_ABB (APNODO, INFOR)

{El algoritmo realiza la eliminación de un elemento en un árbol binario de búsqueda. APNODO es una variable, por referencia, de tipo ENLACE. INFOR es un parámetro de tipo entero que contiene la información del nodo que se desea eliminar}
{AUX, AUXI y OTRO son variables auxiliares de tipo puntero. BO es una variable de tipo booleano}

```
1. Si (APNODO ≠ NIL)
     entonces
   1.1 Si (INFOR < APNODO^.INFO)
         entonces
           Regresar a Eliminación_ABB con APNODO^.IZQ e INFOR
         si no
   1.1.1   Si (INFOR > APNODO^.INFO)
             entonces
               Regresar a Eliminación_ABB con APNODO^.DER e INFOR
             si no
               Hacer OTRO ← NODO
     1.1.1.1   Si (OTRO^.DER = NIL)
                 entonces
                   Hacer APNODO ← OTRO^.IZQ
                 sino
       1.1.1.1.1   Si (OTRO^.IZQ = NIL)
                     entonces
                       Hacer APNODO ← OTRO^.DER
                     sino
                       Hacer AUX ← APNODO^.IZQ y BO ← FALSO
   1.1.1.1.1.A         Mientras (AUX^.DER ≠ NIL) Repetir
                         Hacer AUX1 ← AUX, AUX ← AUX^.DER
                           y BO ← VERDADERO
   1.1.1.1.1.B       {Fin del ciclo del paso 1.1.1.1.1.A}
                     Hacer APNODO^.INFO ← AUX^.INFO y
                       OTRO ← AUX
   1.1.1.1.1.C       Si (BO = VERDADERO)
                       entonces
                         Hacer AUX1^.DER ← AUX^.IZQ
                       sino
                         Hacer APNODO^.IZQ ← AUX^.IZQ
     1.1.1.1.1.D     {Fin del condicional del paso 1.1.1.1.1.C}
   1.1.1.1.2         {Fin del condicional del paso 1.1.1.1.1}
   1.1.1.2           {Fin del condicional del paso 1.1.1.1}
                     Quitar (OTRO) {Se libera el espacio de memoria}
   1.1.2             {Fin del condicional del paso 1.1.1}
 1.2                 {Fin del condicional del paso 1.1}
 sino
     Escribir "La información a eliminar no se encuentra en el árbol"
{Fin del condicional del paso 1}
```

ÁRBOLES BALANCEADOS.

- Con el objeto de mejorar el rendimiento en la búsqueda surgen los **árboles balanceados.** La idea central de éstos es la de realizar reacomodos o balanceos, después de inserciones o eliminaciones de elementos. Estos árboles también reciben el nombre de **árboles AVL** en honor a sus inventores, dos matemáticos rusos, G.M. Adelson Velskii y E. M. Landis.

- Formalmente se define un árbol balanceado como un árbol binario de búsqueda, en el cual se debe cumplir la siguiente condición: “Para todo nodo T del árbol, la altura de los subárboles izquierdo y derecho no debe diferir en más de una unidad”.

Árboles balanceados.