Árboles

DEFINICIÓN

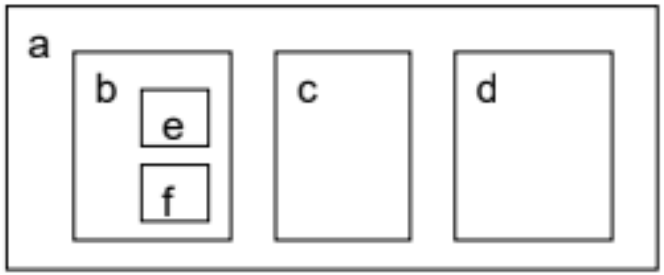
- Un árbol es una estructura no lineal, jerárquica, organizada y dinámica.
  - **Jerárquica** porque los componentes están a distinto nivel.
  - **Organizada** porque importa la forma en que esté dispuesto el contenido.
  - **Dinámica** porque su forma, tamaño y contenido pueden variar durante la ejecución.
- Un árbol puede ser:
  - Vacío,
  - Una raíz + subárboles.

UTILIDAD

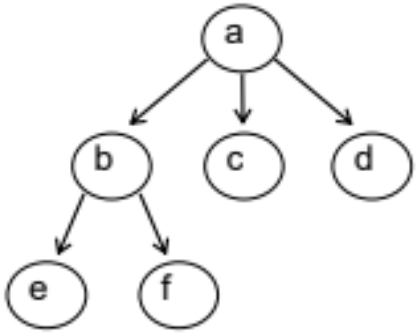
- Para representar fórmulas algebraicas.
- *Para organizar objetos en orden de tal forma que las búsquedas sean muy eficientes*
- En aplicaciones diversas tales como inteligencia artificial o algoritmos de cifrado.
- En diseño de compiladores
- En procesado de texto.

Representación Visual de un Árbol.

- Mediante diagramas de Venn



- Mediante círculos y flechas



- Mediante paréntesis anidados:

(a (b (e,f), c, d))

Conceptos Básicos (P-1)

- **Camino** es una secuencia de nodos en los que cada nodo es adyacente al siguiente.
- ***Padre*** es un nodo que tiene sucesores inmediatos.
- ***Hijo***, cualquiera de los nodos sucesores inmediatos de un nodo.
- ***Descendiente*** de un nodo, es cualquier sucesor de dicho nodo.
- ***Ascendiente*** de un nodo, son el padre y abuelos de dicho nodo.
- ***Hermano*** de un nodo, es otro nodo con el mismo padre.
- ***Generación***, es un conjunto de nodos con la misma profundidad.

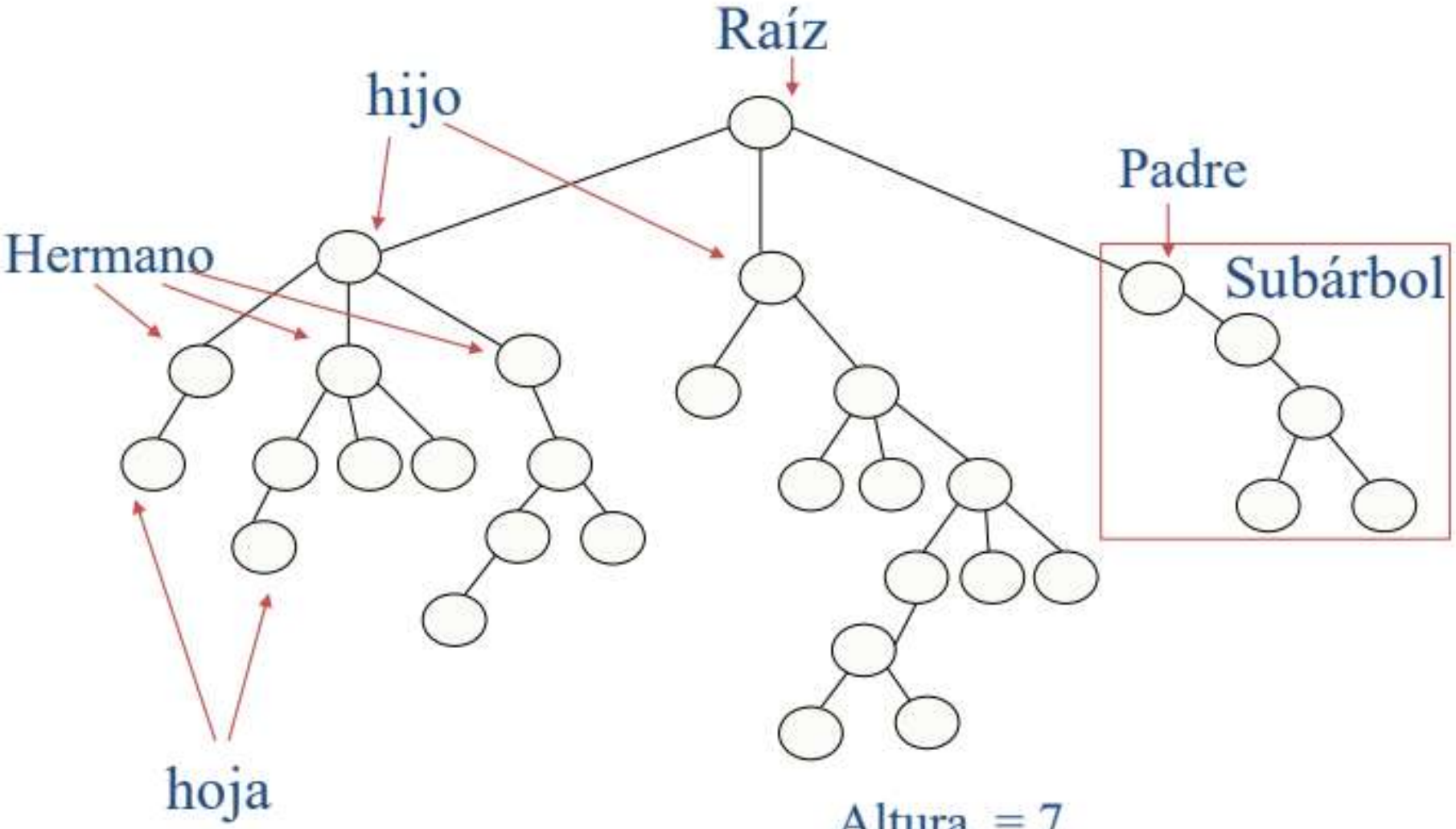
Conceptos Básicos (P-2)

- ***Raíz*** es el nodo que no tiene ningún predecesor (sin padre).
- ***Hoja*** es el nodo que no tiene sucesores (sin hijos) (Terminal). Los que tienen predecesor y sucesor se llaman nodos interiores.
- ***Rama*** es cualquier camino del árbol.
- ***Bosque*** es un conjunto de árboles desconectados.
- ***Nivel,*** es la longitud del camino desde la raíz hasta ese nodo. En el nivel 0 esta la raíz y nivel (predecesor)+1 para los demás nodos.
- ***Altura o profundidad,*** es el nivel de la hoja del camino más largo desde la raíz + 1. Se lo sabe denotar con la letra h. La altura de un árbol vacío es 0

Conceptos Básicos (P-3)

- Los nodos de la misma generación tienen el mismo nivel.
- ***Grado de un nodo***, es el número de flechas que salen de ese nodo (hijos). El número de las que entran siempre es uno.
- ***Grado de un árbol***, es el mayor grado que puede hallarse en sus nodos.
- ***Longitud del camino entre 2 nodos***: es el número de arcos que hay entre ellos.

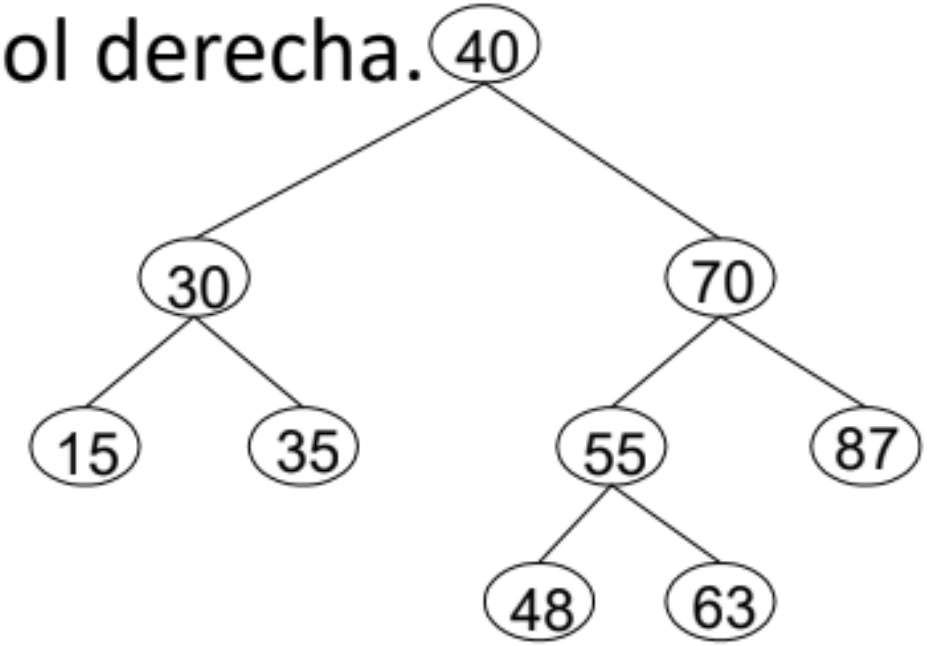
Conceptos Básicos (cont.)



Tipos de árboles

Un árbol ordenado: Es aquel en el que las ramas de los nodos están ordenadas.

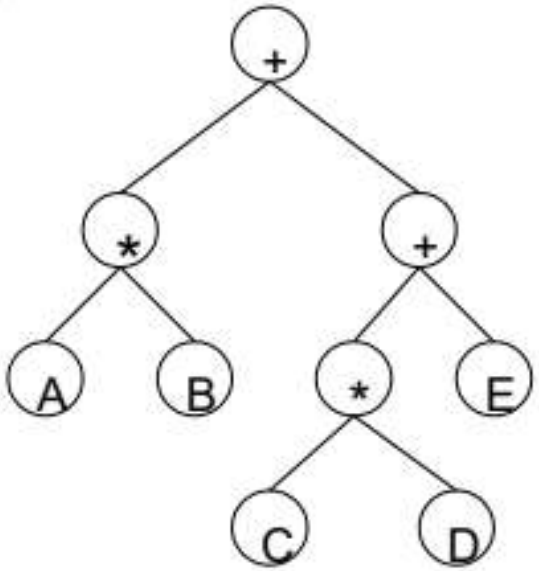
- Los de grado 2 se llaman árboles binarios.
- Cada árbol binario tiene un subárbol izquierda y subárbol derecha.



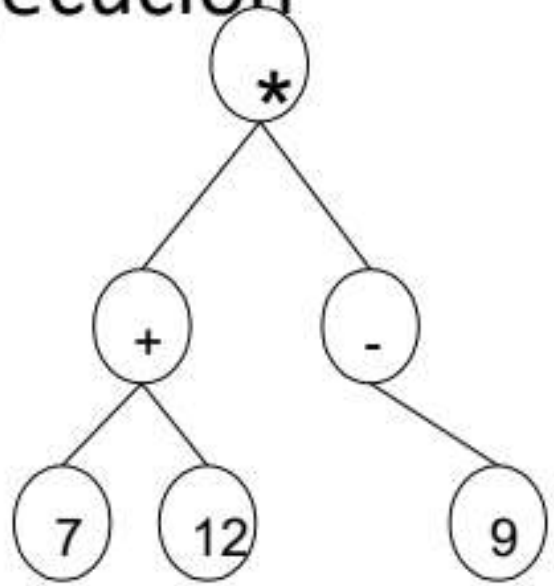
Tipos de árboles (cont.)

Árboles de expresión

- Representan un orden de ejecución



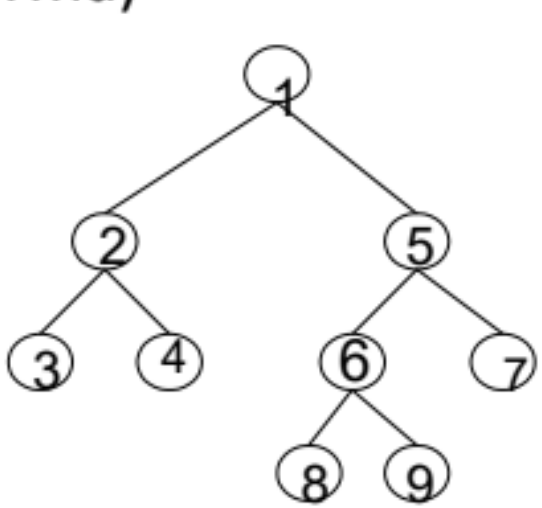
(A* B) + C * D + E

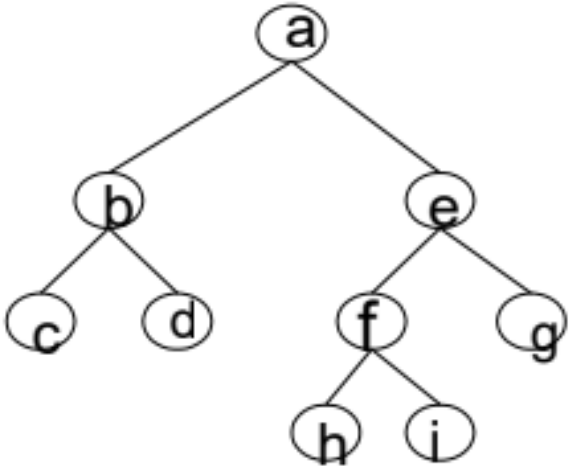


(7 + 12) * (-9) → -171

Tipos de árboles (cont.)

- ***Árboles similares***: Los que tienen la misma estructura (forma)





- ***Árboles Equivalentes (Idénticos):*** Son los árboles similares y sus nodos contienen la misma información.
- ***Árboles n-ario:*** Es un árbol ordenado cuyos nodos tiene N subárboles, y donde cualquier número de subárboles puede ser árboles vacíos

Tipos de árboles (cont.)

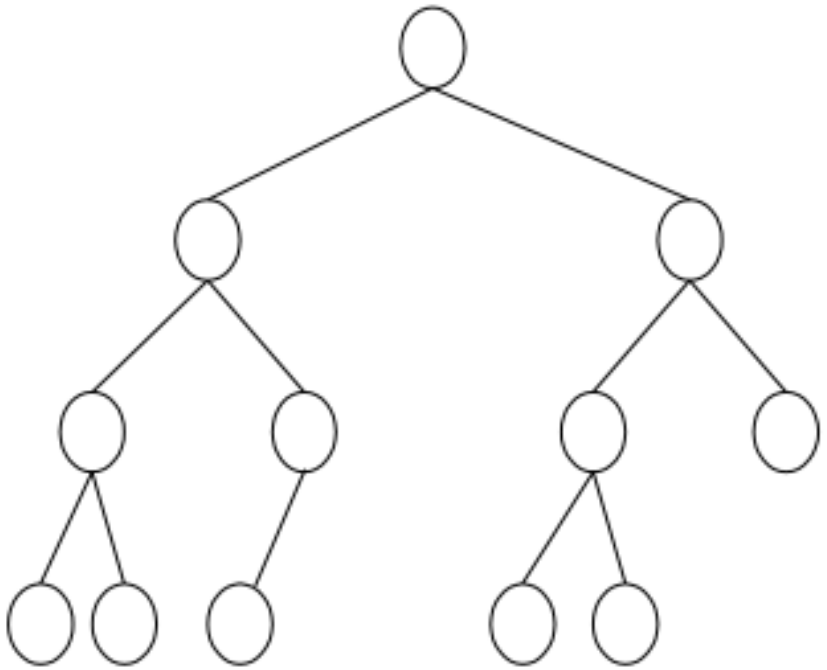
Árbol binario Lleno:

- Es un árbol en el que todos sus nodos, excepto los del ultimo nivel, tienen dos hijos.
- Número de nodos en un árbol binario lleno= 2^h –1 (Donde h es la altura del árbol. En el ejemplo h = 4, → 15) esto nos ayuda a calcular el nivel de árbol necesario para almacenar los datos de una aplicación.

Árboles Binarios de Búsqueda (ABB)

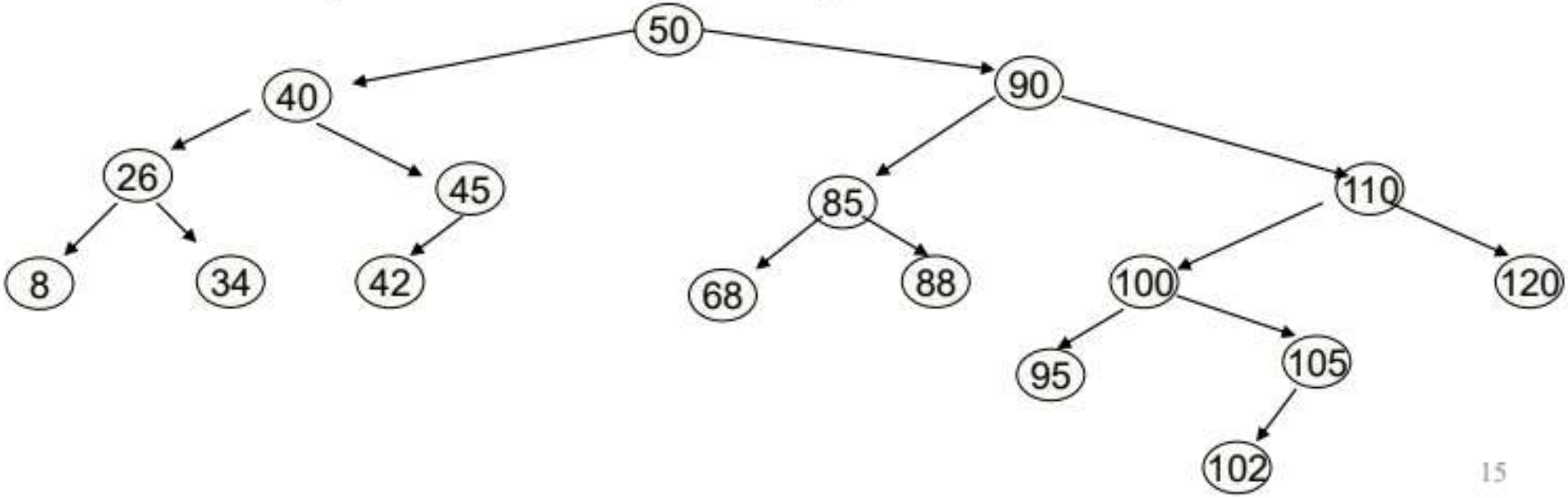
Árboles Binarios de Búsqueda

- Un árbol es un *ABB* si éste es binario y sus nodos son subárboles de búsqueda binarios y contienen información ordenada de tal manera que todos los elementos a la izquierda de la raíz son menores a la raíz y todos lo elementos a la derecha de la raíz son mayores a la raíz.



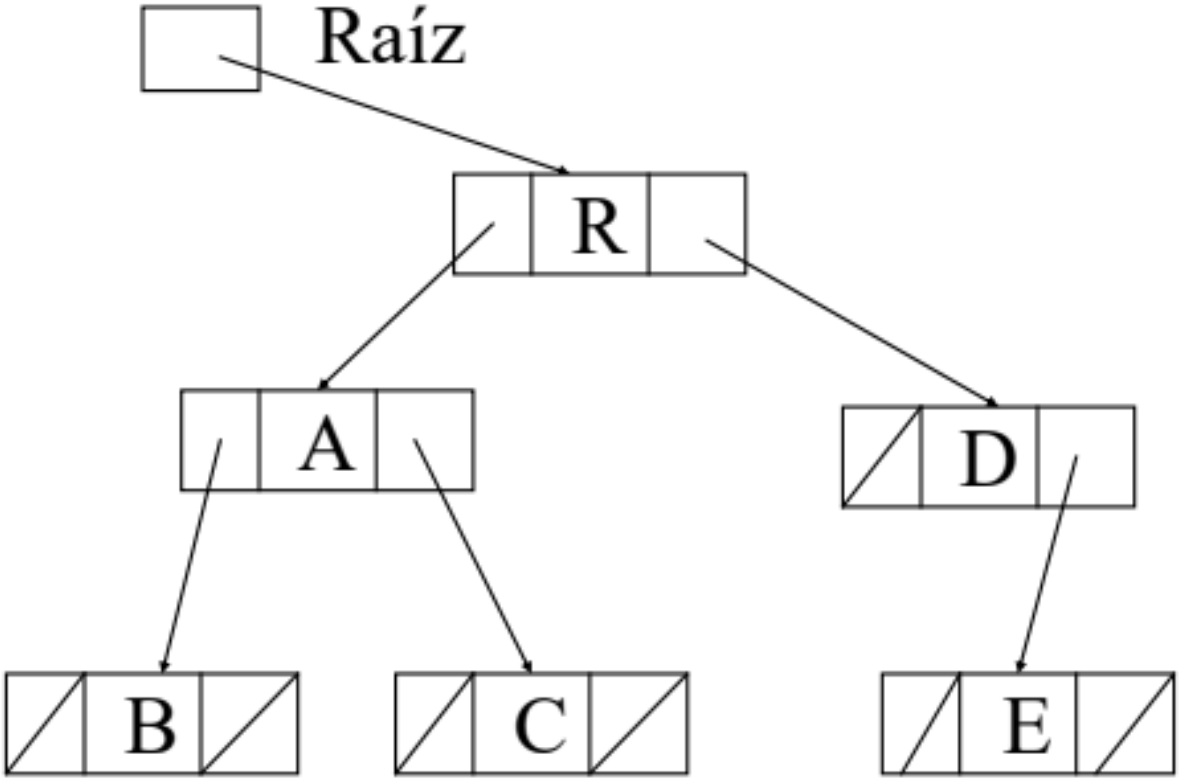
Características de un ABB

- Todos los nodos a la izquierda son menores al padre.
- Todos los nodos a la derecha son mayores al padre.
- Y solo pueden tener 2 hijos a lo mucho.



Representación de un árbol binario en la memoria.

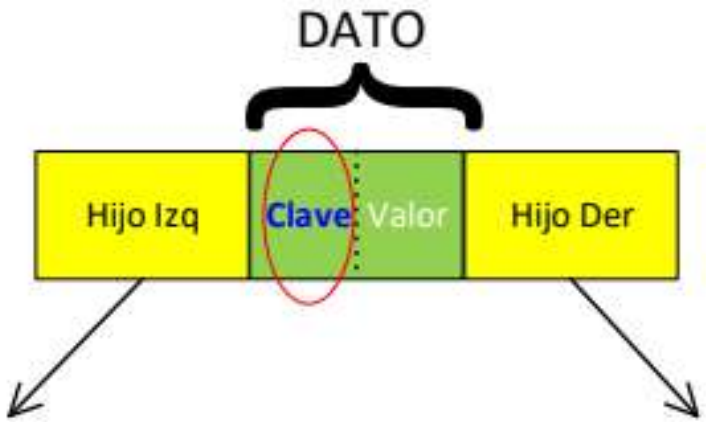
- Cada nodo tiene la siguiente forma:



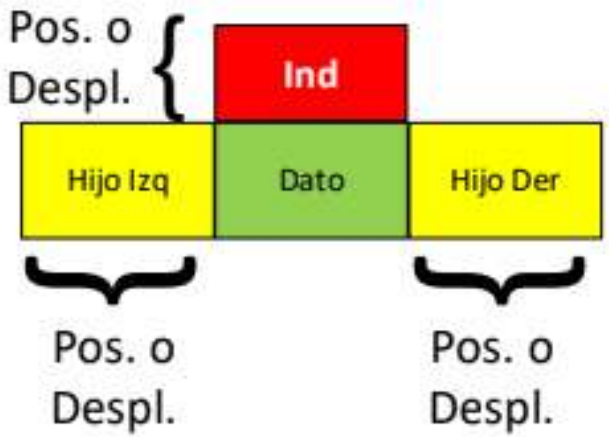
Representación de un nodo de árbol binario mediante punteros o ref.

- Almacenando solo algún dato:

- O almacenando Clave-Valor



Representación de nodo de árbol binario arreglos o archivos.



Arreglo o Archivo

Pos. o Despl.	Dato	Hijo Izq	Hijo Der
0			
1			
2			
3			
...			
n			

ADT NodoBinario v1

```
public class NodoBinario<T> {
    private T dato;
    private NodoBinario<T> hijoIzquierdo;
    private NodoBinario<T> hijoDerecho;
    public NodoBinario() {};
    public NodoBinario(T dato) {…};
    public T getDato() { … }
    public NodoBinario<T> getHijoIzquierdo() { … }
    public NodoBinario<T> getHijoDerecho() { … }
    public void setDato(T dato) {…}
    public void setHijoIzquierdo(NodoBinario<T> hijo) { … }
    public void setHijoDerecho(NodoBinario<T> hijo) { … }
    public boolean esVacioHijoIzquierdo() {…};
    public boolean esVacioHijoDerecho() { …};
    public boolean esHoja() {…};
    public static boolean esNodoVacio(NodoBinario nodo) {…}
}
```

ADT NodoBinario v2

```
public class NodoBinario<K,V> {
    private K clave;
    private V valor;
    private NodoBinario<K,V> hijoIzquierdo;
    private NodoBinario<K,V> hijoDerecho;
    public NodoBinario() {};
    public NodoBinario(K clave, V valor) {…};
    public K getClave() { … }
    public V getValor() { … }
    public NodoBinario<K,V> getHijoIzquierdo() { … }
    public NodoBinario<K,V> getHijoDerecho() { … }
    public void setClave(K clave) {…}
    public void setValor(V dato) {…}
    public void setHijoIzquierdo(NodoBinario<K,V> hijo) { … }
    public void setHijoDerecho(NodoBinario<K,V> hijo) { … }
    public boolean esVacioHijoIzquierdo() {…};
    public boolean esVacioHijoDerecho() { …};
    public boolean esHoja() {…};
    public static boolean esNodoVacio(NodoBinario nodo) {…}
}
```

Operaciones sobre un árbol

- Inserción nodo
- Eliminar nodo
- Buscar nodo con información
- Calcular profundidad del árbol
- Contar nodos
- Contar hojas.
- Recorrer árbol
  - Preorden
  - Inorden
  - Postorden
  - Por Niveles
- Reconstruir árbol a partir de sus recorridos

Interfaz IArbolBusqueda – v1

```
public interface IArbolBusqueda<T extends Comparable<T>> {
    void insertar(T dato) throws DatoYaExisteExcepcion;
    T eliminar(T dato) throws DatoNoExisteExcepcion;
    T buscar(T dato) throws DatoNoExisteExcepcion;
    boolean contiene(T dato);
    int size();
    int altura();
    void vaciar();
    boolean esArbolVacio();
    int nivel();
    List<T> recorridoEnInOrden();
    List<T> recorridoEnPreOrden();
    List<T> recorridoEnPostOrden();
    List<T> recorridoPorNiveles();
}
```

Interfaz IArbolBusqueda – v2

```
public interface IArbolBusqueda<K extends Comparable<K>, V> {
    void insertar(K clave, V valor);
    V eliminar(K clave) throws ClaveNoExisteExcepcion;
    V buscar(K clave);
    boolean contiene(K clave);
    int size();
    int altura();
    void vaciar();
    boolean esArbolVacio();
    int nivel();
    List<K> recorridoEnInOrden();
    List<K> recorridoEnPreOrden();
    List<K> recorridoEnPostOrden();
    List<K> recorridoPorNiveles();
}
```

Inserción en un ABB

- La *inserción* es una operación que se puede realizar eficientemente en un árbol binario de búsqueda. La estructura crece conforme se inserten elementos al árbol.
- Los pasos que deben realizarse para insertar un elemento a un ABB son los siguientes:
  - Debe compararse el valor o dato 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**.

Inserción en un ABB (cont.)

- Repetir sucesivamente el paso 1 hasta que se cumpla alguna de las siguientes condiciones
  - El subárbol derecho es igual a vació, o el subárbol izquierdo es igual a vació; en cuyo caso se procederá a insertar el elemento en el lugar que le corresponde.
  - El valor o dato que quiere insertarse es igual a la raíz del árbol; en cuyo caso no se realiza la inserción.

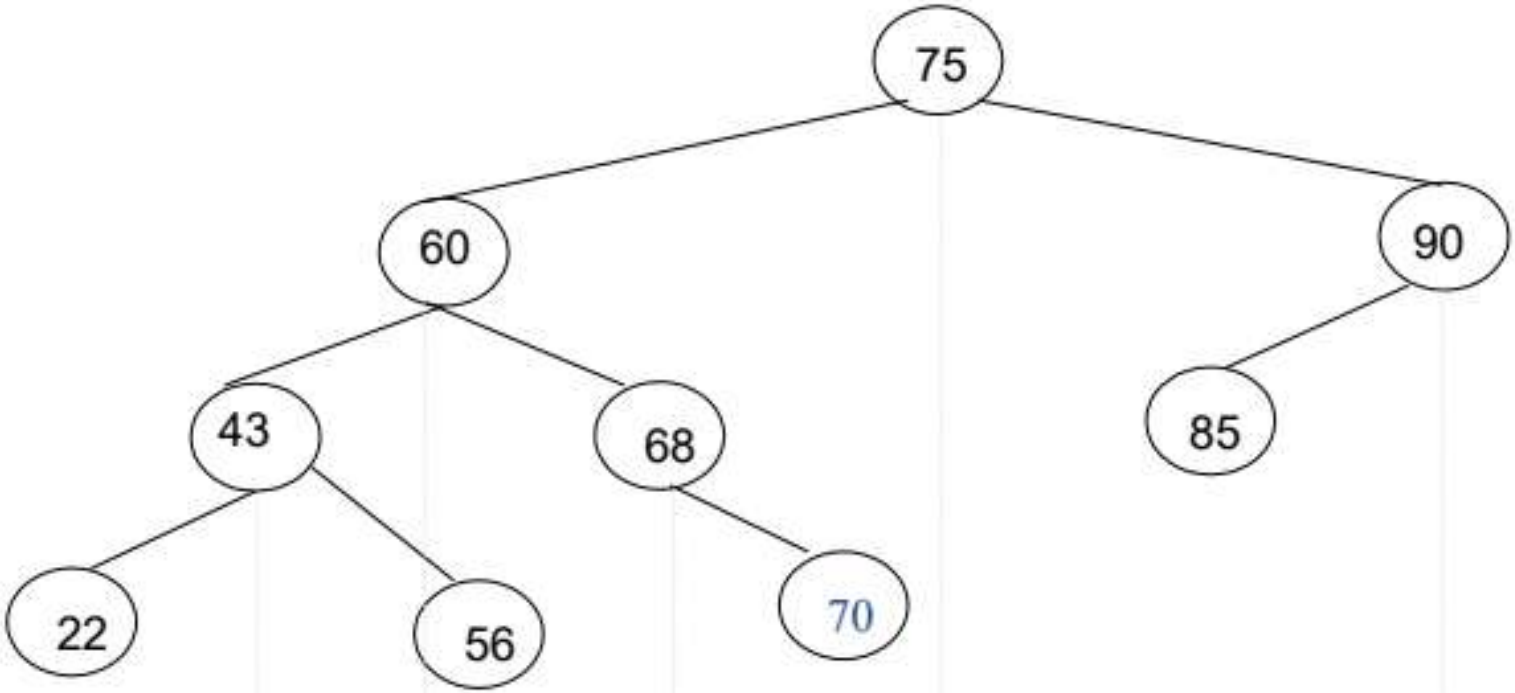
Inserción en un ABB (cont.)

- Supóngase que quieren insertarse las siguientes los siguientes datos en un árbol binario de búsqueda que se encuentra vació.

75 –60 – 43 – 68 – 90 – 70 – 85 – 22 – 56

Inserción en un ABB (cont.) Solución

75 –60 – 43 – 68 – 90 – 70 – 85 – 22 – 56

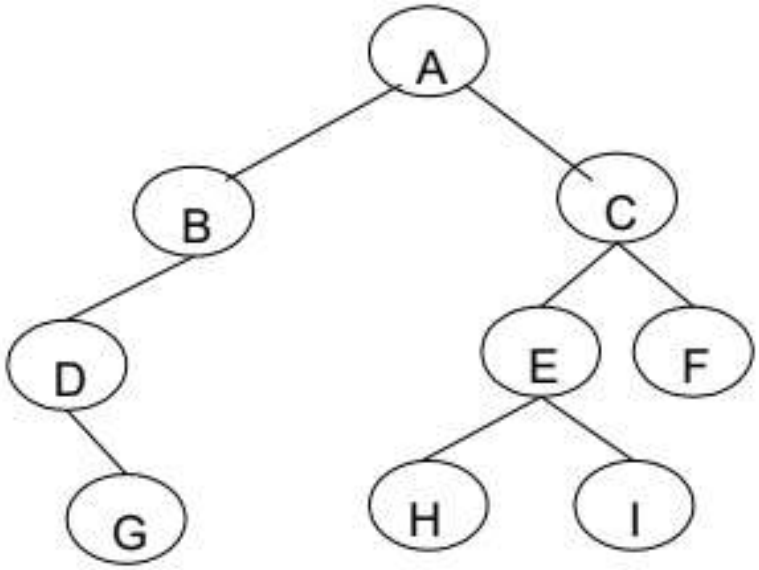


Recorridos en un árbol binarios

- Recorrido por niveles (Recorrido en amplitud)
  - Utiliza un cola en su implementación
- Recorridos en profundidad
  - Utiliza una pila en su implementación (o se implementa recursivamente). Este recorrido puede ser:
    - Recorrido en PreOrden
    - Recorrido en InOrden
    - Recorrido en PostOrden

Recorrido Por Niveles

- Recorrido por niveles
  - Visita todos los nodos de un nivel de izquierda a derecha.
  - Visita los nodos del siguiente nivel de izquierda a derecha

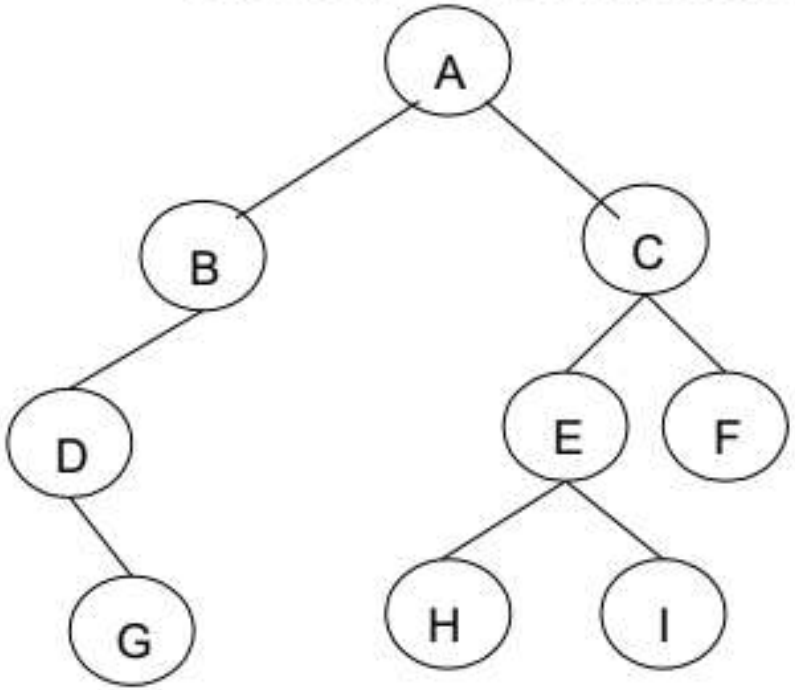


DE Izq a Der

Por Niveles = A, B, C,D, E, F, G, H, I

Recorridos de un árbol de Búsqueda Binaria (ABB)

- Recorrido en preorden (prefijo)
  - Visita el nodo en turno.
  - Recorre el subárbol izquierdo en preorden.
  - Recorre el subárbol derecho en preorden.

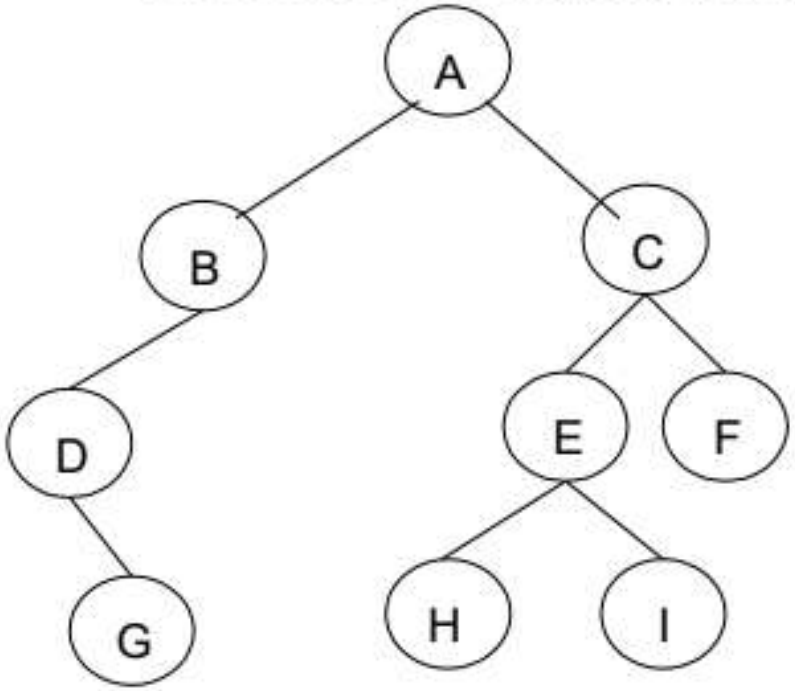


RID

Preorden = A B D G C E H I F

Recorridos de un árbol de Búsqueda Binaria (ABB) (cont.)

- Recorrido en inorden (infijo)
  - Recorre el subárbol izquierdo en inorden.
  - Visita el nodo en turno
  - Recorre el subárbol derecho en inorden.

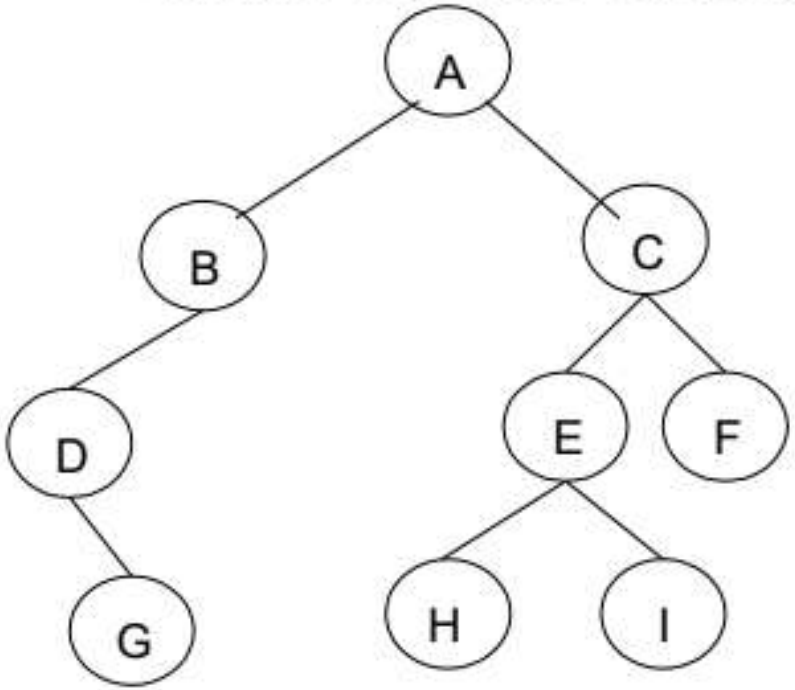


IRD

Inorden: D G B A H E I C F

Recorridos de un árbol de Búsqueda Binaria (ABB) (cont.)

- Recorrido en postorden (postfijo)
  - Recorre el subárbol izquierdo en postorden.
  - Recorre el subárbol derecho en postorden.
  - Visita el nodo en turno.



IDR

Postorden : G D B H I E F C A

Recursividad en arboles

- Se puede aplicar recursión por inducción completa sobre los arboles utilizando como variable de inducción la altura del árbol
- Los algoritmos recursivos precisan tener un parámetro para controlar la recursividad.
- En los arboles este parámetro es un nodo, para iniciar con la raíz del árbol a procesar en la primera llamada.
- Para implementar recursividad en árboles se requiere:
  - Una rutina pública que será a la que llame el usuario de nuestra clase.
  - Una rutina privada o protegida:
    - Es la rutina amiga o compañera a la que invocará la rutina pública
    - Es la incorpora al menos el parámetro del tipo del nodo que usa el árbol
    - Realiza casi todo el trabajo, siendo invocada por la rutina pública con la raíz del árbol como nodo de partida y accede a los otros nodos del árbol de forma recursiva.

RECONSTRUCCION DE ARBOLES BINARIOS POR SUS RECORRIDOS

- En general se puede reconstruir un árbol binario con dos recorridos, siempre y cuando uno de ellos sea el recorrido InOrden.
- Entonces podemos usar el recorrido en preorden con el recorrido inOrden o el recorrido en PostOrden con el recorrido InOrden.

RECONSTRUCCION DE ARBOLES BINARIOS POR SUS RECORRIDOS

- Si tenemos el recorrido en preorden y el recorrido InOrden:
  - El primer elemento en el recorrido en preorden es la raíz del árbol. Llamémoslo x.
  - Luego buscamos x en el recorrido en InOrden, y los elementos a la izquierda de x en el recorrido InOrden estarán el subárbol izquierdo y los que están a la derecha de x estarán en el subárbol derecho del árbol.
  - Luego de este paso se ha dividido los recorridos en nuevos recorridos inorden y preorden a la derecha y a la izquierda de x.
  - Luego repetimos la operación en cada división que se haga de los recorridos

RECONSTRUCCION DE ARBOLES BINARIOS POR SUS RECORRIDOS

- Si tenemos el recorrido en postorden y el recorrido InOrden:
  - El ultimo elemento en el recorrido en postorden es la raíz del árbol. Llamémoslo x.
  - Luego buscamos x en el recorrido en InOrden, y los elementos a la izquierda de x en el recorrido InOrden estarán el subárbol izquierdo y los que están a la derecha de x estarán en el subárbol derecho del árbol.
  - Luego de este paso se ha dividido los recorridos en nuevos recorridos inorden y postorden a la derecha y a la izquierda de x.
  - Luego repetimos la operación en cada división que se haga de los recorridos

Eliminar un nodo

Para eliminar un nodo existen los siguientes casos:

1. Si el elemento a borrar es Terminal (hoja),
2. Si el elemento a borrar tiene un solo hijo,
3. Si el elemento a borrar tiene los dos hijo,

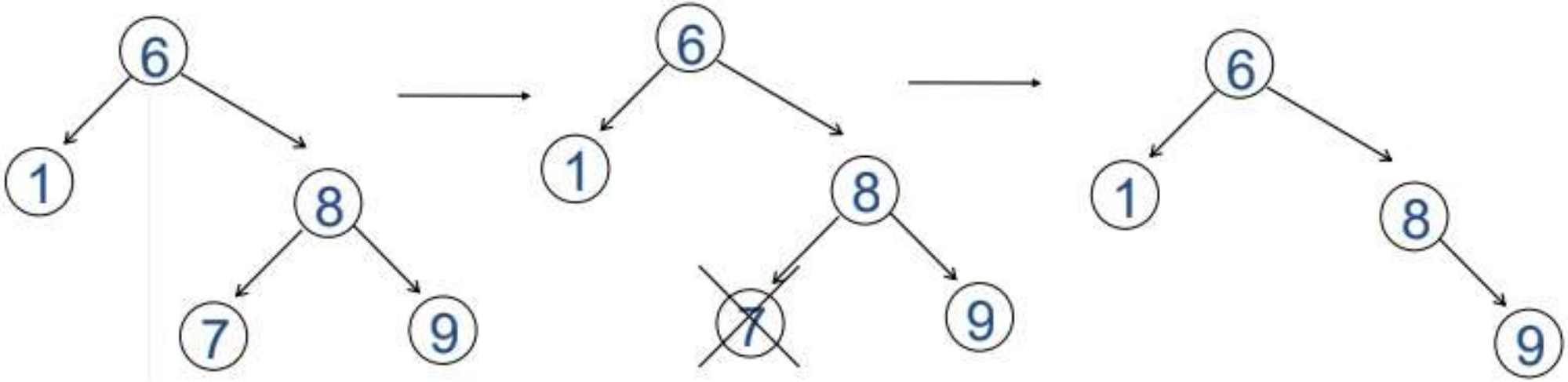
Eliminar un nodo (cont.)

- Caso 1

Si el elemento a borrar es terminal (hoja), simplemente se elimina.

aux = aux.izq = null

Ejemplo eliminar nodo 7

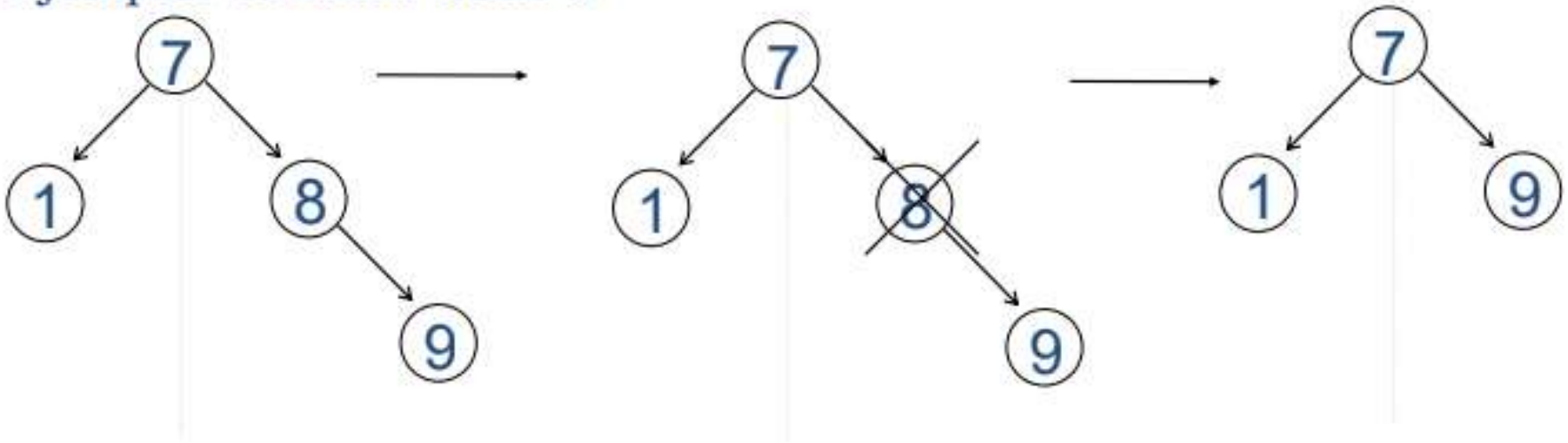


Eliminar un nodo (cont.)

- Caso 2

  Si el elemento a borrar tiene un solo hijo, entonces tiene que sustituirlo por el hijo

Ejemplo: eliminar nodo 8

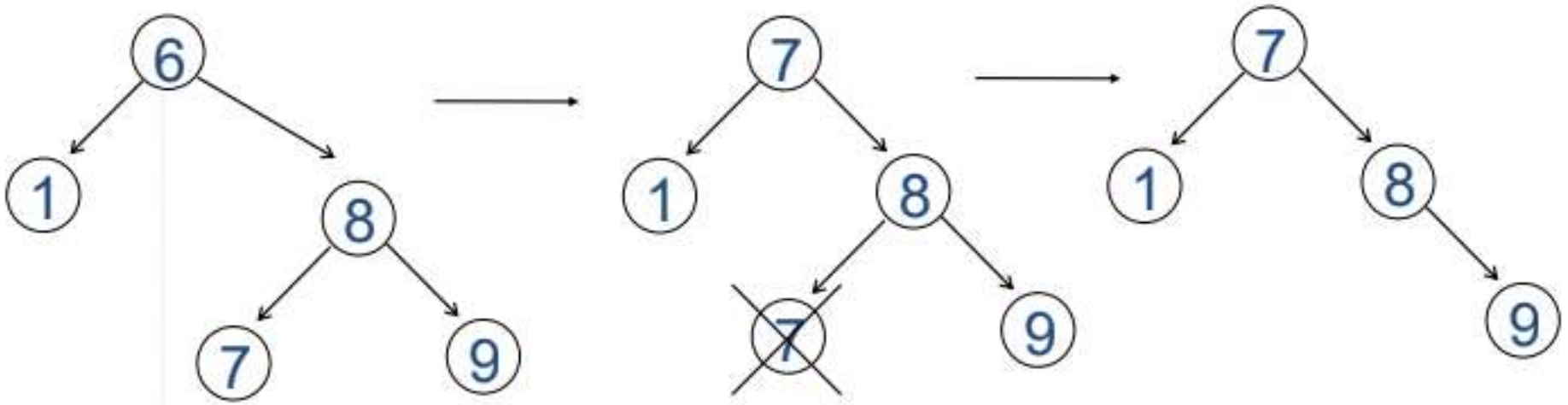


Eliminar un nodo (cont.)

- Caso 3

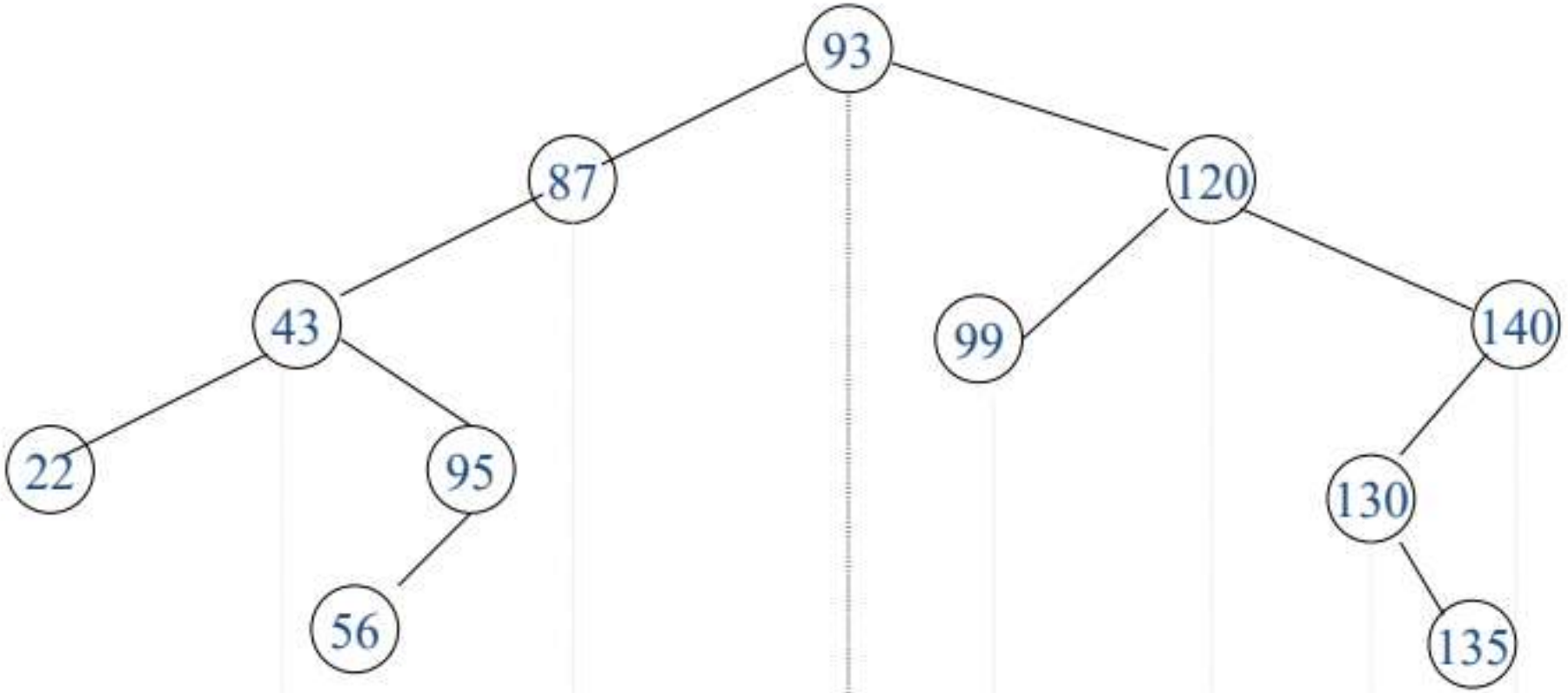
Si el elemento a borrar tiene los dos hijos, entonces se tienen que sustituir por el nodo que se encuentra mas a la izquierda en el subárbol derecho, o por el nodo que se encuentra mas a la derecha en el subárbol izquierdo.

Ejemplo: eliminar el 6



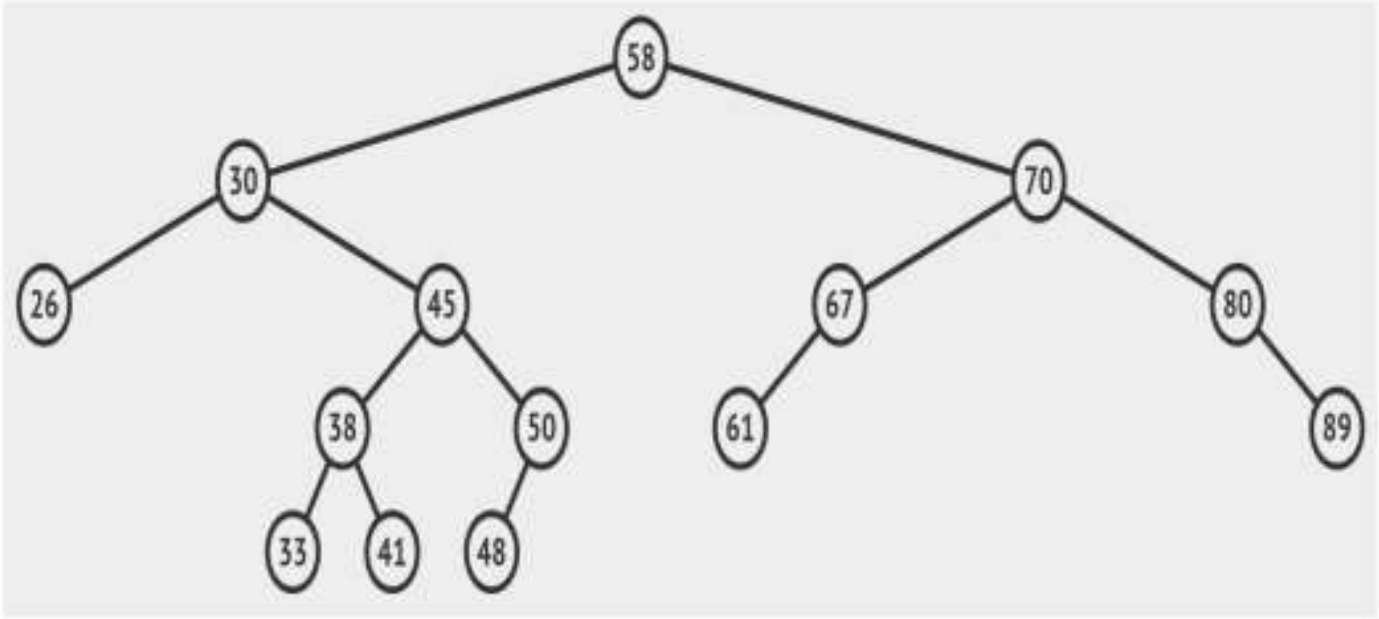
Eliminar un nodo (cont.)

- Elimina el 22, 99, 87, 120, 140, 135, 56



Ejemplo - Inserciones

Datos a Insertar
58
70
80
30
45
50
38
67
89
26
33
41
48
61



Ejemplo con Arreglos - Inserciones

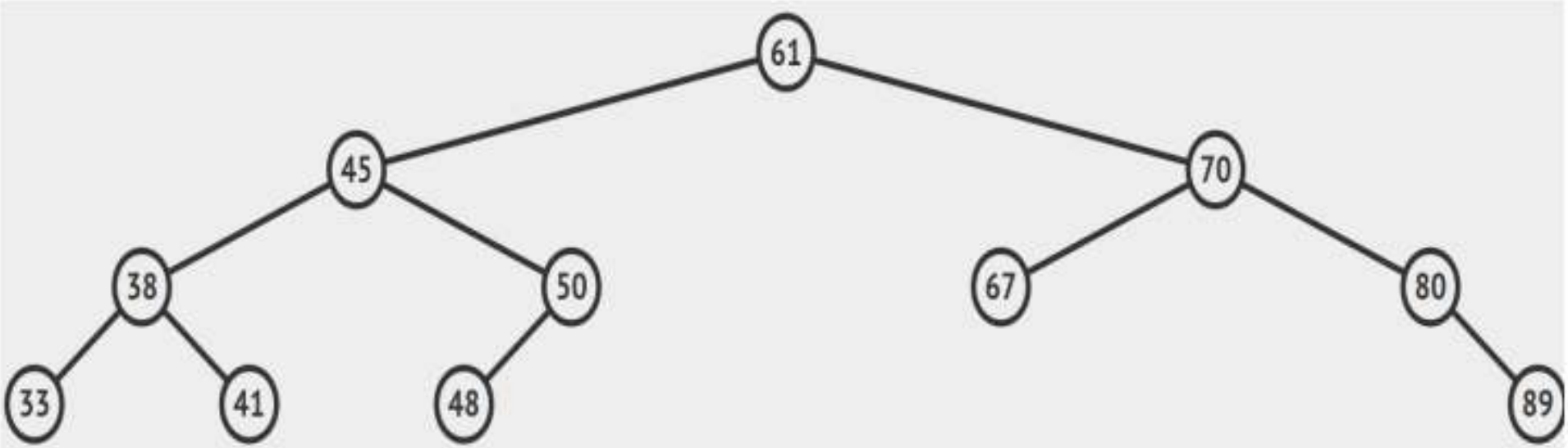
Datos a Insertar
58
70
80
30
45
50
38
67
89
26
33
41
48
61

Raíz = 0

	Dato	Hij. Izq	Hij. Der.
0	58	3	1
1	70	7	2
2	80	-1	8
3	30	9	4
4	45	6	5
5	50	12	-1
6	38	10	11
7	67	13	-1
8	89	-1	-1
9	26	-1	-1
10	33	-1	-1
11	41	-1	-1
12	48	-1	-1
13	61	-1	-1
14		-1	-1
15		-1	-1
16		-1	-1

Ejemplo - Eliminaciones

Datos Eliminar
58
26
30



Ejemplo con arreglos-Eliminaciones

Raíz = 13

Datos Eliminar
58
26
30

	Dato	Hij. Izq	Hij. Der.
0		-1	-1
1	70	7	2
2	80	-1	8
3		-1	-1
4	45	6	5
5	50	12	-1
6	38	10	11
7	67	-1	-1
8	89	-1	-1
9		-1	-1
10	33	-1	-1
11	41	-1	-1
12	48	-1	-1
13	61	4	1
14		-1	-1
15		-1	-1
16		-1	-1