TEMA 5

ÁRBOLES(*)

Una de las estructuras las datos más importantes y prominentes que existen es el árbol. No es un árbol en el sentido botánico de la palabra, sino uno de naturaleza más abstracta. Todos hemos visto usar tales árboles para describir conexiones familiares. Los dos tipos más comunes de árboles familiares son el 'árbol de antecesores', que empieza en un individuo y va hacia atrás a través de padres, abuelos, etc., y el 'árbol de descendientes', que va hacia delante a través de hijos, nietos, etc.

David Harel, 'The Spirit of Computing'

(*)Con la colaboración de Gemma Morales de Paz (**)

(**) Fuentes : - N.Wirth, Algoritmos+ E.Datos = Programas, Ed. Castillo, 1983.
- Dale y Lilly, Pascal y Estructuras de Datos, Ed. McGrawHill, 1989.

OBJETIVOS DE ESTE CAPITULO:

- Descubrir los árboles como paradigma de los tipos Recursivos de Datos
- ¿Cúando utilizar un árbol para almacenar información?
- Diferenciar las formas de recorrer un árbol

INDICE TEMA - 5

Árboles. 9 horas.

1. Definición.

➢ Definición de Árbol. Similitud con las Listas.

- Árbol Degenerado.
- ¿Cómo representarlos?

■ Mediante Grafos:

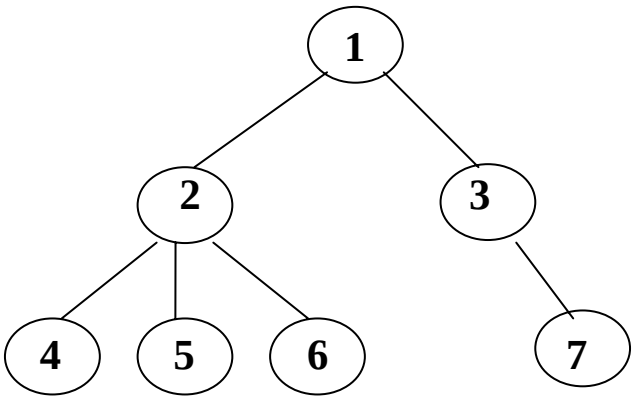

Figura - 1.
Representación de un árbol mediante un grafo

■ Mediante Conjuntos.

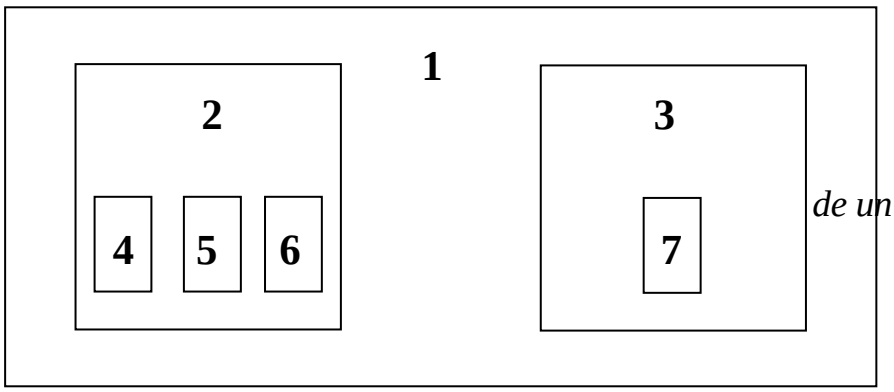

Figura - 2.
Representación de un árbol mediante un grafo

➢ **Una estructura árbol** con tipo base T es:

● Bien la estructura vacía.

● O bien un nodo de tipo T junto con un número finito de estructuras árbol, de tipo base T, disjuntas, llamadas **subárboles**.

➢ Declaración:

```
TArbol = ^TNodo;
TNodo = Record
            Clave: TClave;
            Izq, Der: TArbol
        End;
```

2. Conceptos

Antecesor Antecesor Directo	Sucesor Sucesor Directo	Nodo Raíz. Nodo Hoja Nodo Interno Nivel de un Nodo
Grado de un Nodo Grado de un Árbol	Altura de un Árbol	Longitud de Camino Longitud de Camino Interno
Árbol de expansión Longitud de Camino Externo		

***Tabla - 1**. Conceptos diversos sobre árboles.*

Concepto	*Definiciones*	*Ejemplo*	*Observaciones*
Antecesor Directo	La clave "X" está inmediatamente por encima de la clave "Y" en el árbol y además están unidas.	El 2 es antecesor directo del 4, del 5 y del 6	La clave "X" es antecesora directa de la clave "Y"
Antecesor	Existe una sucesión de claves antecesoras directas para llegar desde "Y" hasta "X".	El 1 es antecesor del 4	La clave "X" es antecesora de "Y"
Sucesor Directo	La clave "X" está inmediatamente por debajo de la clave "Y" en el árbol y además están unidas.	El 5, el 4 y el 6 son sucesores directos del 2	Una clave "X" es sucesora directa de otra clave "Y"
Sucesor	Existe una sucesión de claves sucesoras directas para llegar desde "X" hasta "Y".	El 4 es sucesor del 1	La clave "X" es sucesora de "Y")
Nodo Raíz	aquel que no tiene antecesores.	El 1	

Tabla -2. Conceptos básicos. Los ejemplos hacen referencia a la Figura - 1 (I).

Concepto	*Definiciones*	*Ejemplo*	*Observaciones*

Nodo Hoja	Aquel que no tiene sucesores.	El 4, el 5, el 6 y el 7	
Nodo Interno	Aquel que no es ni raíz ni hoja.	El 2 y el 3	
Nivel de un Nodo	Nº de tramos que hay que recorrer desde el nodo raíz hasta el nodo del que quiero calcular su nivel.	El nodo 7 tiene nivel 3.	Se considera que la **raíz** está en el **nivel 1**.
Grado de un Nodo	Nº de descendientes directos que tiene.	El 1 tiene grado 2. El 2 tiene grado tiene grado 3.	
Grado de un Árbol	Mayor de los grados de los nodos que componen el árbol.	Grado del árbol = 3.	Un árbol de grado 1 → **Lista** Un árbol de grado 2 → **Árbol Binario**
Altura de un Árbol	Mayor de los niveles de los nodos que forman el árbol.	Altura del árbol = 3.	

Tabla -3. Conceptos básicos. Los ejemplos hacen referencia a la Figura - 1 (II).

Concepto	*Definiciones*	*Ejemplo*	*Observaciones*
Longitud de Camino	Nº de arcos que hay que atravesar para ir desde la raíz hasta un nodo.	Longitud de camino del nodo 7 = 3.	La longitud de camino del nodo **raíz** se considera **1**. Coincide con el NIVEL DE UN NODO.
Longitud de camino Interno	Nº de arcos que hay que atravesar para ir desde la raíz hasta un nodo. La longitud de camino del nodo raíz se considera 1.	1+2+2+3+3+3+3 = 17	$\sum_{i=1}^{n} i * $ nº de claves
Árbol de expansión	Resultante de hacer que **todos los nodos** del árbol tengan el mismo grado.		Implica que hay que añadir una serie de **Nodos Especiales** que no pertenecen al árbol para conseguir esto (ver figura-3).
Longitud de camino externo	Suma de las longitudes de camino de los nodos especiales.	4*12 + 2*3 + 2 = 56	

Tabla -4. Conceptos básicos. Los ejemplos hacen referencia a la Figura - 1 (III).

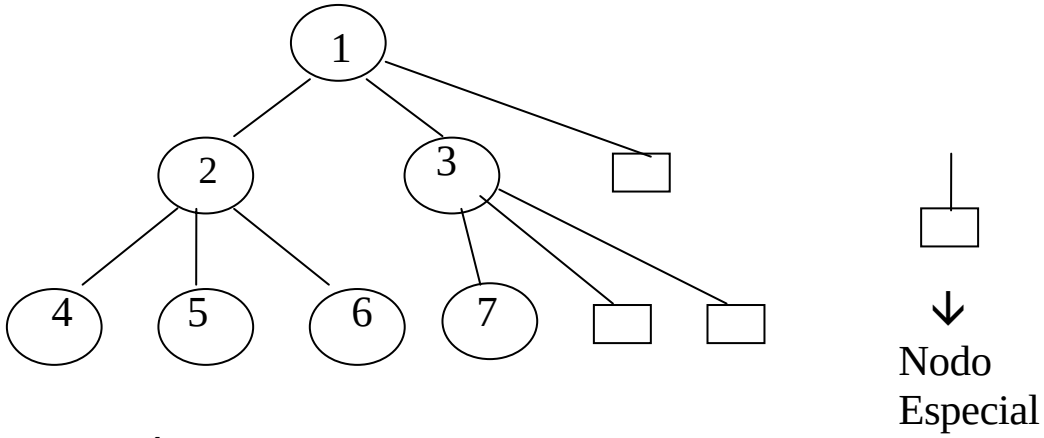

Figura - 3. *Árbol de expansión de uno dado.*

3. Árboles binarios

➢ Definición.

Estructura tipo Árbol de grado dos:

> *Bien una estructura vacía o bien un elemento del tipo base y como máximo 2 estructuras de tipo árbol.*

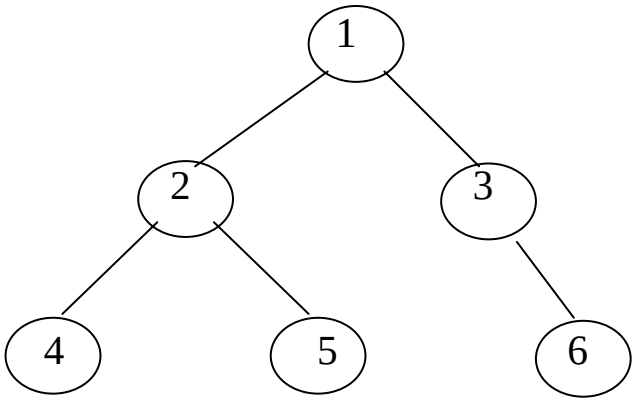

Figura - 4
Árbol binario

3.1. Árbol binario de Búsqueda

> *Elementos Menores → subárbol izquierdo*
> *Elementos Mayores → subárbol derecho*

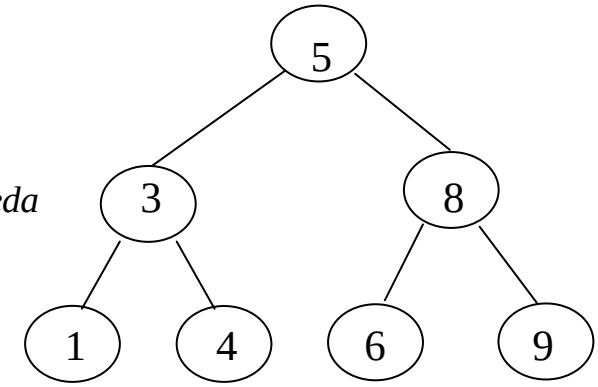

Figura - 5.
Árbol binario de búsqueda

3.2. Tipos de recorrido sobre Árboles.

- Recorridos en Profundidad: se alejan cuanto antes de la raíz.

Tipo de Recorrido	*Secuencia de nodos*
Pre Orden	{ 5; 3; 1; 4; 8; 6; 9 }
Post Orden	{ 1; 4; 3; 6; 9; 8; 5 }
Orden Central	{ 1; 3; 4; 5; 6; 8; 9 }

- Recorridos en Amplitud: trata consecutivamente los nodos que se encuentran al mismo nivel

Tipo de Recorrido	*Secuencia de nodos*
Amplitud	{ 5; 3; 8; 1; 4; 6; 9 }

3.2.1. Codificación

💻 **Recorridos en Profundidad:** Son todos **recursivos** y requieren una **pila**[1] para su implementación.

[1] Normalmente, la pila del sistema.

- PREORDEN: procesar primero el subárbol izquierdo, luego el subárbol derecho y luego la raíz.

```
Procedure PreOrden ( a: TArbol );
begin
    if a <> nil then
    begin
        writeln ( a^.Clave );  (* Procesar Raíz *)
        PreOrden ( a^.Izq );
        PreOrden ( a^.Der )
    end
end;
```

- POSTORDEN: procesar primero el subárbol izquierdo, luego el subárbol derecho y luego la raíz.

```
Procedure PostOrden ( a: TArbol );
begin
    if  a<> nil then
    begin
        PostOrden ( a^.Izq );
        PostOrden ( a^.Der );
        writeln( a^.Clave )  (* Procesar Raíz *)
    end
end;
```

- ORDEN CENTRAL: procesar primero el subárbol izquierdo, luego la raíz y luego el subárbol derecho[2].

```
Procedure OrdenCentral ( a: TArbol );
begin
```

[2] Si el árbol es de búsqueda, puede comprobarse fácilmente que al realizar un recorrido en orden central, obtenemos un listado ordenado de los elementos del árbol.

```
        if  a<> nil then
        begin
            OrdenCentral ( a^.Izq );
            writeln ( a^.Clave );  (* Procesar Raíz *)
            OrdenCentral ( a^.Der )
        end
    end;
```

💻 **Recorridos en Amplitud:** Es imprescindible la utilización de una **cola** y **no** está aconsejada la **recursividad**.

```
    Procedure Amplitud ( a: TArbol );

    (* Se utiliza una cola para efectuar un recorrido por niveles *)

    var Cola: TCola;
        Aux: TArbol;
    begin
        InicializarCola ( Cola );
        Encolar ( Cola, a );    (* raíz del árbol original*)
        while Not Vacia ( Cola ) do
        begin
            Desencolar ( Cola, Aux );
            if Aux<> NIL then
            begin
                writeln ( Aux^.Clave );
                Encolar ( Cola, Aux^.Izq );   (* subárbol izquierdo *)
                Encolar ( Cola, Aux^.Der )   (* subárbol derecho
                *)
            end
    end;
```

3.3. Características y Utilidades de los Recorridos.

- PREORDEN: Se va a utilizar siempre que queramos comprobar alguna propiedad del árbol (p.ej.: localizar elementos).

- ORDENCENTRAL: Se utiliza siempre que nos pidan algo relativo a la posición relativa de las claves o algo que tenga que ver con el orden de las claves (p.ej.: ¿Cuál es la 3ª clave?).

- POSTORDEN: Se utiliza poco. Su principal utilidad consiste en liberar la memoria ocupada por un árbol.

- AMPLITUD: Se utiliza siempre que nos pidan operaciones cuyo tratamiento se haga por niveles.

4. ALGORITMOS FUNDAMENTALES

4.1. Procedimiento de inserción en un árbol Binario de Búsqueda.

→ Todas las inserciones de realizan en Nodos Hoja.

CASOS	*ACCIONES*
No se permiten claves repetidas	Claves menores a la izquierda Claves mayores a la derecha
Sí se permiten claves repetidas	Claves menores a la izquierda Claves mayores a la derecha Claves iguales a la izquierda

➢ CODIFICACIÓN

INSERCIÓN SIN CLAVES REPETIDAS

Pasos:

- Recorrer el árbol buscando la clave a insertar:
  - Si no está → Se inserta
  - Si está → Mensaje de error

```
Procedure Insertar ( var a: TArbol; Elem: TClave );

begin
        if a = nil then   (* búsqueda de la posición adecuada *)
            begin
                new ( a );
                a^.Clave := Elem;
                a^.Izq := nil;
                a^.Der := nil
            end

         if a^.Clave > Elem          (*  búsqueda en el subárbol izq. *)
         then Insertar ( a^.Izq, Elem )

         else if a^.Clave < Elem     (*  búsqueda en el subárbol der. *)
         then Insertar ( a^.Der, Elem )

end;
```

INSERCIÓN CON CLAVES REPETIDAS

Pasos:

- Recorrer el árbol buscando la clave a insertar:
  - Insertar la clave.

```
Procedure Insertar ( var a: TArbol; Elem: TClave );

begin

    if a= nil then          (* búsqueda de la posición adecuada *)
        begin
            new ( a );
            a^.Clave := Elem;
            a^.Izq := nil;
            a^.Der := nil
        end

    else

        if a^.Clave >= Elem then          (* inserción *)
            Insertar ( a^.Izq, Elem )

end;
```

4.2. Procedimiento de borrado en un árbol binario de Búsqueda.

CASOS	*ACCIONES*
La clave no está	Ninguna o mensaje de notificación
La clave está	• Es un nodo hoja (a := NIL) • Es un nodo interno (a^.Der , a^.Izq <> NIL) ♦ Buscar la clave mayor de las menores ó Buscar la clave menor de las mayores. ♦ Se cambia con la clave a borrar y se elimina la clave cambiada.
Nodo con 1 sólo descendiente	a:= a^.Izq a:= a^.Der (ver Figura - 6)

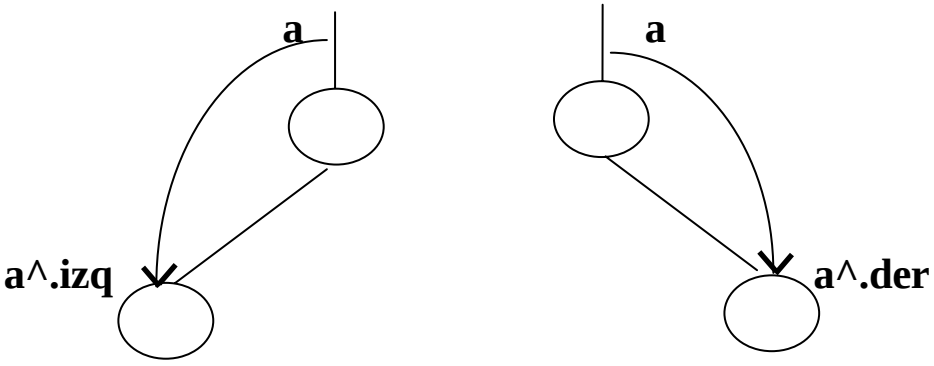

Figura - 6. *Borrado en nodo con un sólo descendiente*

Procedure Borrar (**var** a: TArbol; Elem: TClave);

```
var Aux: TArbol;

        Procedure MayorMenores ( var b: TArbol );
        begin
            if b^.Der = nil then
            begin
                    Aux^.Clave := b^.Clave;
                    Aux := b;
                    b := b^.Izq
            end
            else
                    MayorMenores( b^.Der )
        end;

begin   (* Borrar *)
        if a = nil then
            writeln ( 'la clave no está' )
        else
            if a^.Clave > Clave then     (* búsqueda de la clave *)
                    Borrar ( a^.Izq, Clave )
            else
                    if a^.Clave < Clave then    (* búsqueda de la clave
*)
                            Borrar ( a^.Der, Clave )
                    else begin (* 2 *)
                                    Aux := a;
                                    if a^.Izq = nil then
                                            a := a^.Der
                                    else
                                            if a^.Der = nil then
                                                    a := a^.Izq
                                            else MayorMenores( a^.Izq );
                                    dispose ( Aux )
                                end (* 2 *)
end; (* Borrar *)
```