Árboles Binarios

Estructuras de Datos

Agenda

Árboles Binarios

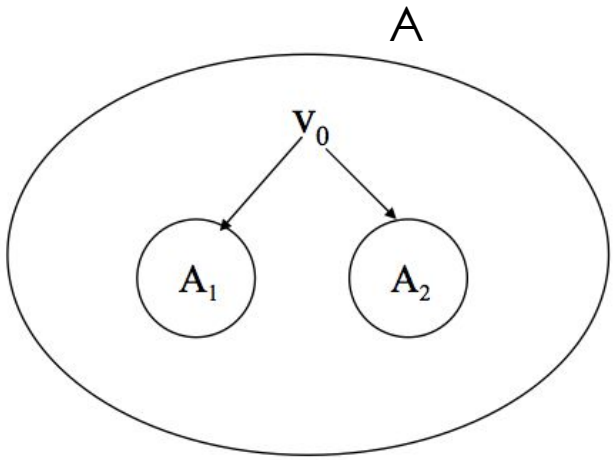
- Concepto.
- Características.
- Terminología.
- Especificación.
- Implementación.
- Recorridos.
- Árboles Binarios de Búsqueda.
  - Concepto.

Árboles Binarios (Binary Tree)

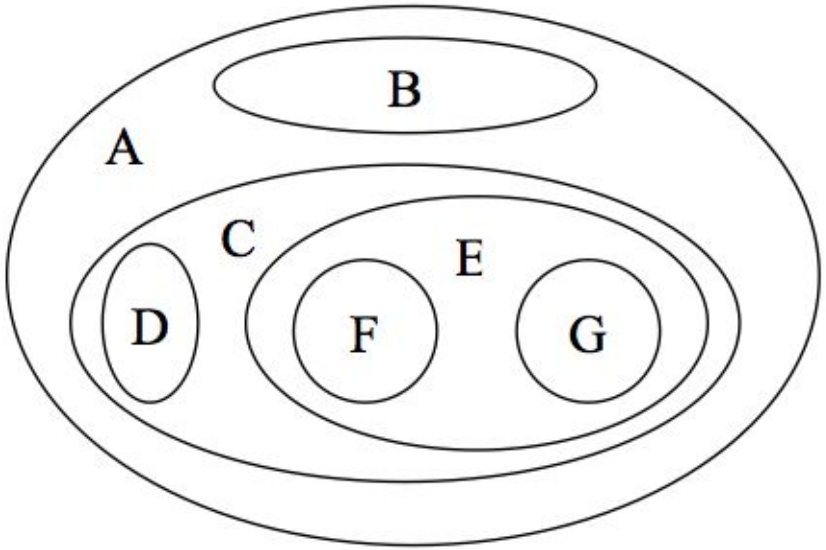
Concepto.

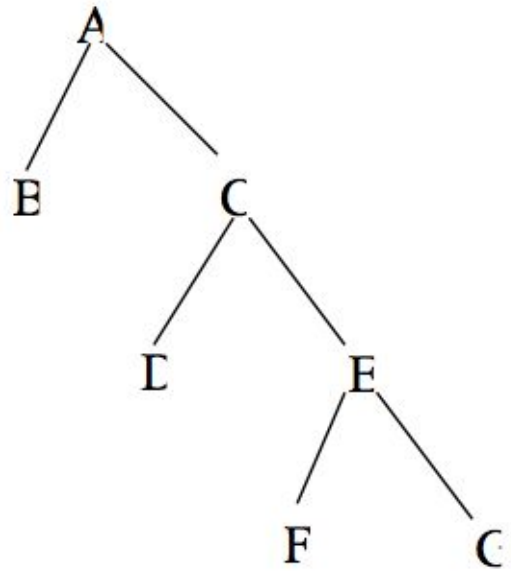
Árbol Binario.

- Es un tipo especial de árbol en el que todo nodo tiene como máximo dos nodos hijos. Son utilizados comúnmente para realizar búsquedas y ordenamientos.
- Se define como un conjunto de elementos llamados nodos o vértices, de forma que:
  - A es vacío, o
  - A contiene un nodo distinguido V_0 llamado raíz de A y los nodos restantes forman dos árboles A_1 (Subárbol izquierdo) y A_2 (Subárbol derecho).



Árbol Binario.





Árboles Binarios.

Características

Árboles Binarios: características

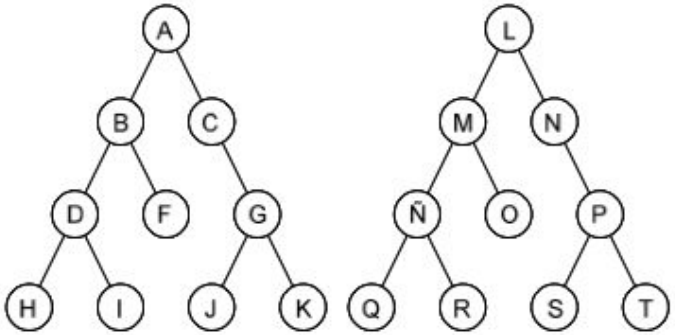
1. Cada nodo puede tener, 0, 1 o 2 nodos como máximo.
2. El número máximo de nodos en el nivel **i** de un árbol binario es $\mathbf{2^i}$.
3. El número máximo de nodos en un árbol binario de altura **k** es $\mathbf{2^{k+1}-1}$.
4. Para cualquier árbol binario no vacío, si $\mathbf{n_0}$ es el número de nodos terminales y $\mathbf{n_2}$ es el número de nodos de grado 2, entonces se cumple que $\mathbf{n_0=n_2+1}$.

Árboles Binarios.

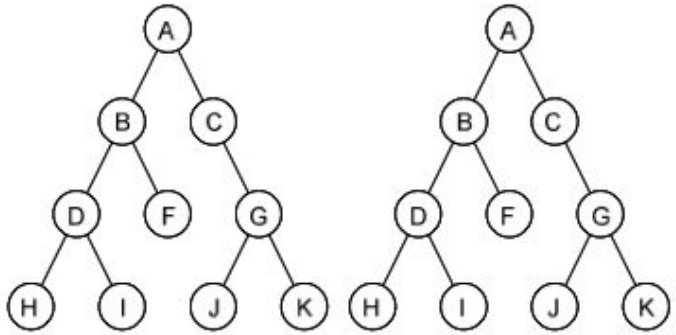
Terminología.

Árboles Binarios: Terminología.

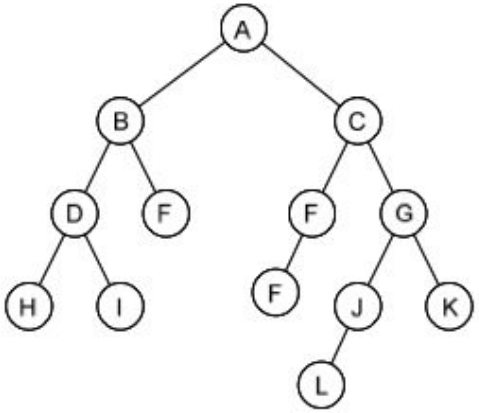
1. Árboles similares: son aquellos que tienen la misma estructura.
2. Árboles equivalentes: son aquellos árboles con la misma estructura y la misma información.
3. Árboles completos o perfectos: son aquellos en los que todos los nodos, excepto las hojas, tienen grado dos.
4. Árbol equilibrado: es aquel en el que las alturas de cada uno de los subárboles de cada uno de sus nodos tiene como máximo una diferencia de una unidad.
5. Árbol degenerado: es aquel en el que todos sus nodos tiene solo un subárbol.



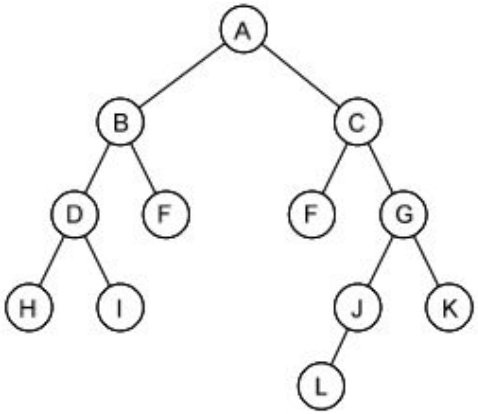
Árboles similares



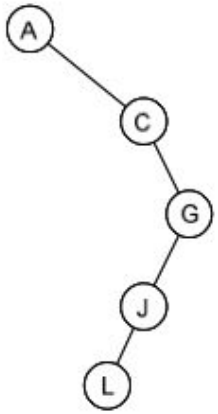
Árboles equivalentes



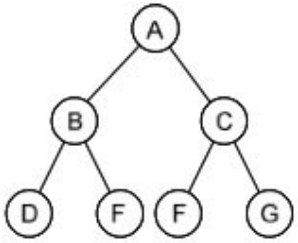
Árbol equilibrado



Árbol no equilibrado



Árbol degenerado



Árbol completo

Árboles Binarios.

Especificación.

Especificación.

- Básicamente es la misma que la especificación de los Árboles Generales.
- Tiene que considerarse las funciones:
  - Hijo_izq(Nodo Idnodo): Nodo
  - Hijo_der(Nodo Idnodo): Nodo

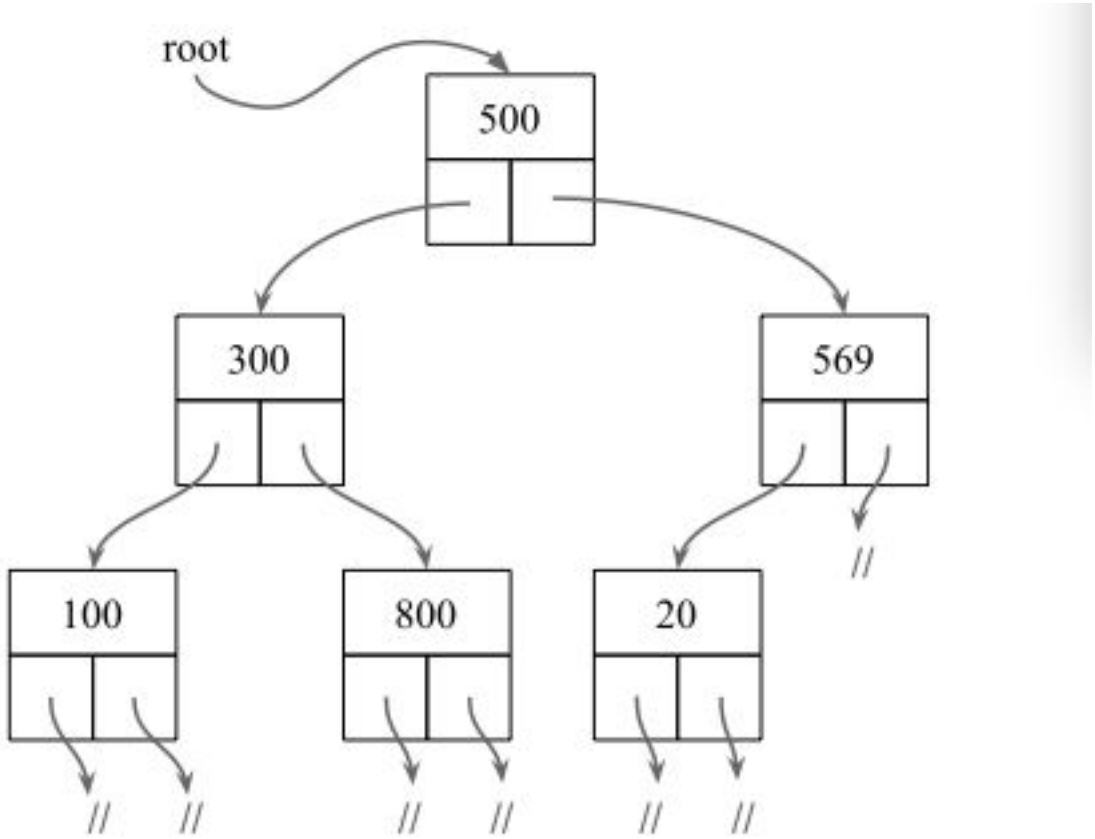
Árboles Binarios.

Implementación.

Implementación con apuntadores

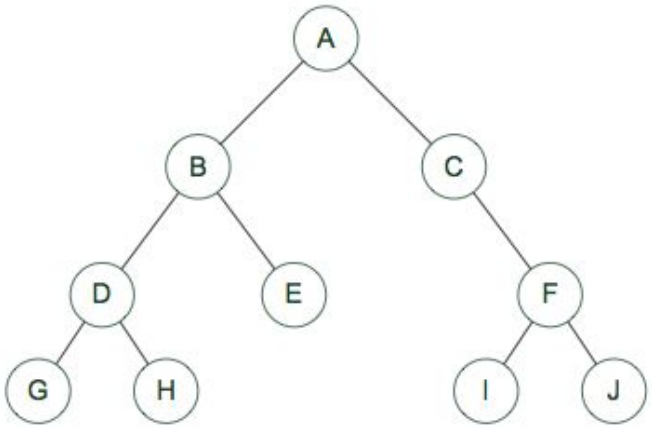
```
Clase Nodo<T>
    público:
    T tInfo
    Nodo<T>* hijo_izq
    Nodo<T>* hijo_der
fin
Clase ÁrbolG <T>
    privado :
    Nodo <T>*  pRoot
    público :
    ... //funciones públicas.
fin
```

Implementación con apuntadores



Implementación con arreglos.

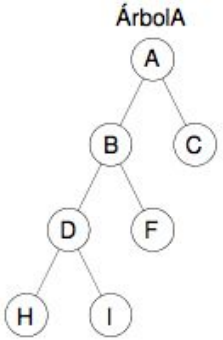
- Se puede implementar como un arreglo del tipo base del árbol.
- El primer elemento corresponde al nivel 0 del árbol.
- Los dos siguientes elementos se corresponden al nivel 1…

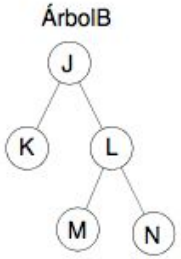


Nivel 0	Nivel 1		Nivel 2				Nivel 3							
A	B	C	D	E		F	G	H					I	J

Implementación con arreglos.

- Se pueden implementar como un arreglo de nodos.
- Hijo_der e hijo_izq son de tipo entero.
- El valor -1 en hijo_der se entiende como que el nodo no tiene un valor válido.





	hizq	raíz	hder
1	15	D	11
2			-1
3	7	A	9
4			-1
5	0	F	0
6	0	M	0
7	1	B	5
8	14	J	12
9	0	C	0
10	0	N	0
11	0	I	0
12	6	L	10
13			-1
14	0	K	0
15	0	H	0

ÁrbolA
3

ÁrbolB
8

Implementación con arreglos.

```
Clase Nodo <T>
   privado:
   T elemento
   entero: hijo_der
   entero: hijo_izq
   Constructor()
   Destructor()
Fin_clase

Clase ArbolB <T>
   privado:
   entero: pRoot, máx
   arreglo [1.. máx] de Nodo: memoria
   público:
   …
Fin_clase
```

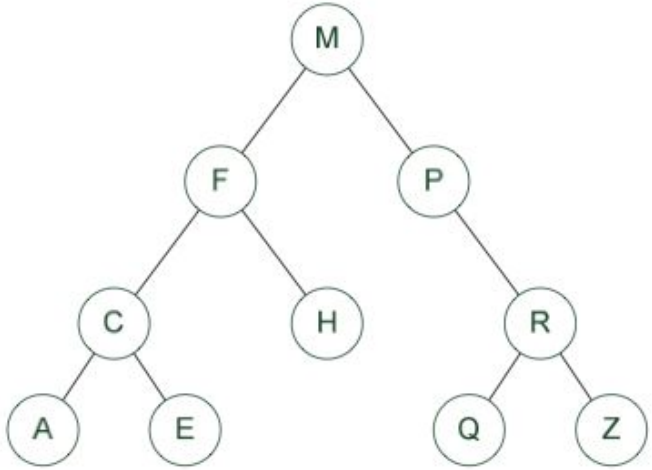
¿Qué faltaría a esta implementación, con respecto al manejo de los elementos no utilizados?

Árboles Binarios.

Recorridos.

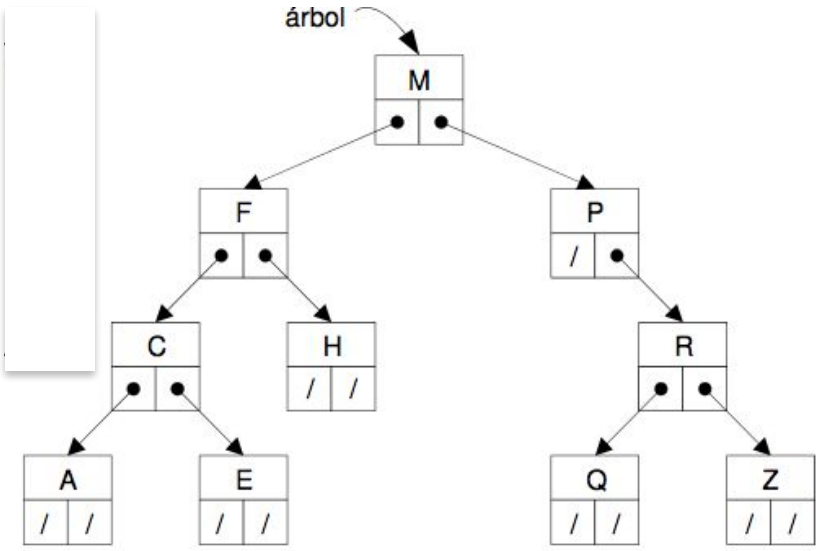
Recorridos

- Recorrido preorden.
  - Por cada elemento no vacío:
    - ✓ Se accede primero al nodo raíz.
    - ✓ Se accede al hijo izquierdo en preorden.
    - ✓ Se accede al hijo derecho en preorden.
    - ✓ Para el siguiente árbol:
      - M,F,C,A,E,H,P,R,Q,Z
- Recorrido inorden.
  - Por cada elemento no vacío:
    - ✓ Se accede al hijo izquierdo en inorden.
    - ✓ Se accede primero al nodo raíz.
    - ✓ Se accede al hijo derecho en inorden.
    - ✓ Para el siguiente árbol:
      - A,C,E,F,H,M,P,Q,R,Z
- Recorrido postorden.
  - Por cada elemento no vacío:
    - ✓ Se accede al hijo izquierdo en postorden.
    - ✓ Se accede al hijo derecho en postorden.
    - ✓ Se accede primero al nodo raíz.
    - ✓ Para el siguiente árbol:
      - A,E,C,H,F,Q,Z,R,P,M



Recorridos

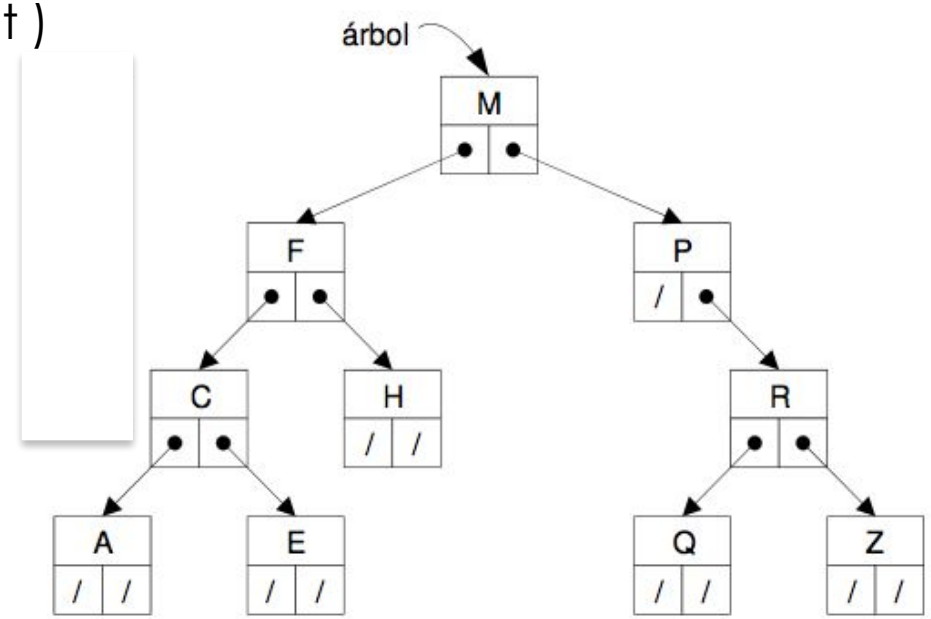
```
Procedimiento Preorden( (E)Nodo <T> * root )
    si root < > NULL entonces
        Visitar( root )
        Preorden( hijo_izq( root ))
        Preorden( hijo_der( root ) )
    fin_si
fin
```



Recorrido preorden: M F C A E H P R Q Z

Recorridos

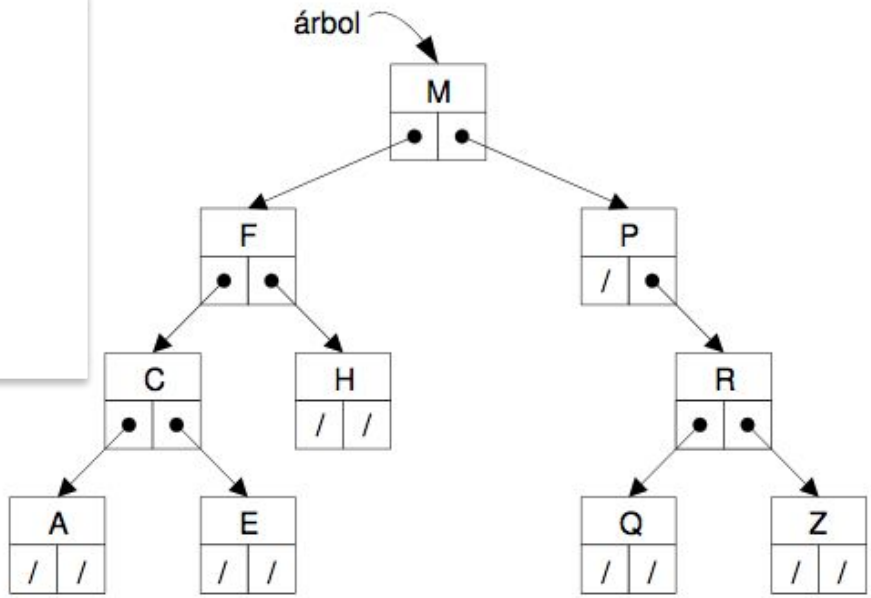
```
Procedimiento Inorden( (E)Nodo <T> * root )
    si root < > NULL entonces
        Inorden( hijo_izq( root ))
        Visitar( root )
        Inorden( hijo_der( root )
    fin_si
fin
```



Recorrido inorden: A C E F H M P Q R Z

Recorridos

```
Procedimiento Postorden( (E)Nodo <T> * root )
    si root < > NULL entonces
        Postorden( hijo_izq( root ))
        Postorden( hijo_der( root ))
        Visitar( root )
    fin_si
fin
```



Recorrido postorden: A E C H F Q Z R P M

Árboles Binarios de Búsqueda.

Concepto.

Árboles BB: concepto.

- Es un árbol binario, que puede estar vacío, y que si no está vacío, cumple las siguientes propiedades:

(1) Todos los nodos están identificados por una clave y no existen dos elementos con la misma clave.

(2) Las claves de los nodos del subárbol izquierdo son menores que la clave del nodo raíz.

(3) Las claves de los nodos del subárbol derecho son mayores que la clave del nodo raíz.

(4) Los subárboles izquierdo y derecho son también árboles binarios de búsqueda.

Árboles BB.

Operaciones.

Operaciones.

- Búsqueda.
- Inserción de un nodo.
- **Eliminación de un nodo.**

Búsqueda.

```
función Buscar ((E) entero x, (E/S) Nodo * root): * Nodo
    si (es_vacío(root)) entonces
        retornar null
    si_no
        si (x < *root.info) entonces
            retornar Buscar (x, Hijo_izq(root))
        si_no
            si (x > *root.info) entonces
                retornar Buscar(x, Hijo_der(root))
            si_no
                retornar root
            fin_si
        fin_si
    fin_si
fin_función
```