Tema 14: ÁRBOLES

Estructura Árbol

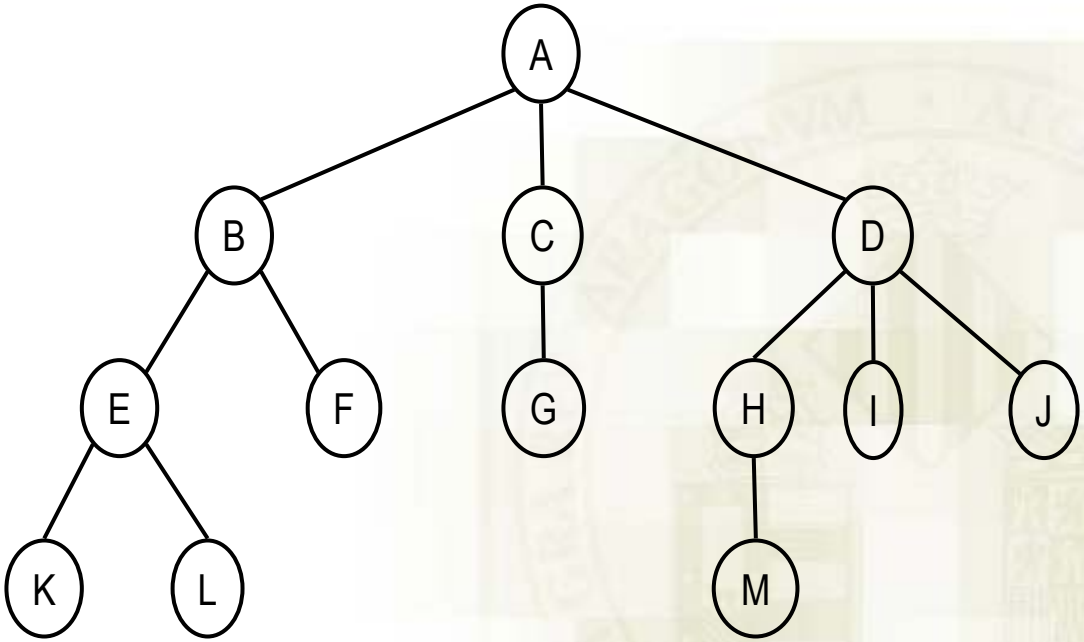
- **Definición**: Una estructura de árbol con tipo base *Valor* es:

  (i) Bien la estructura vacía.

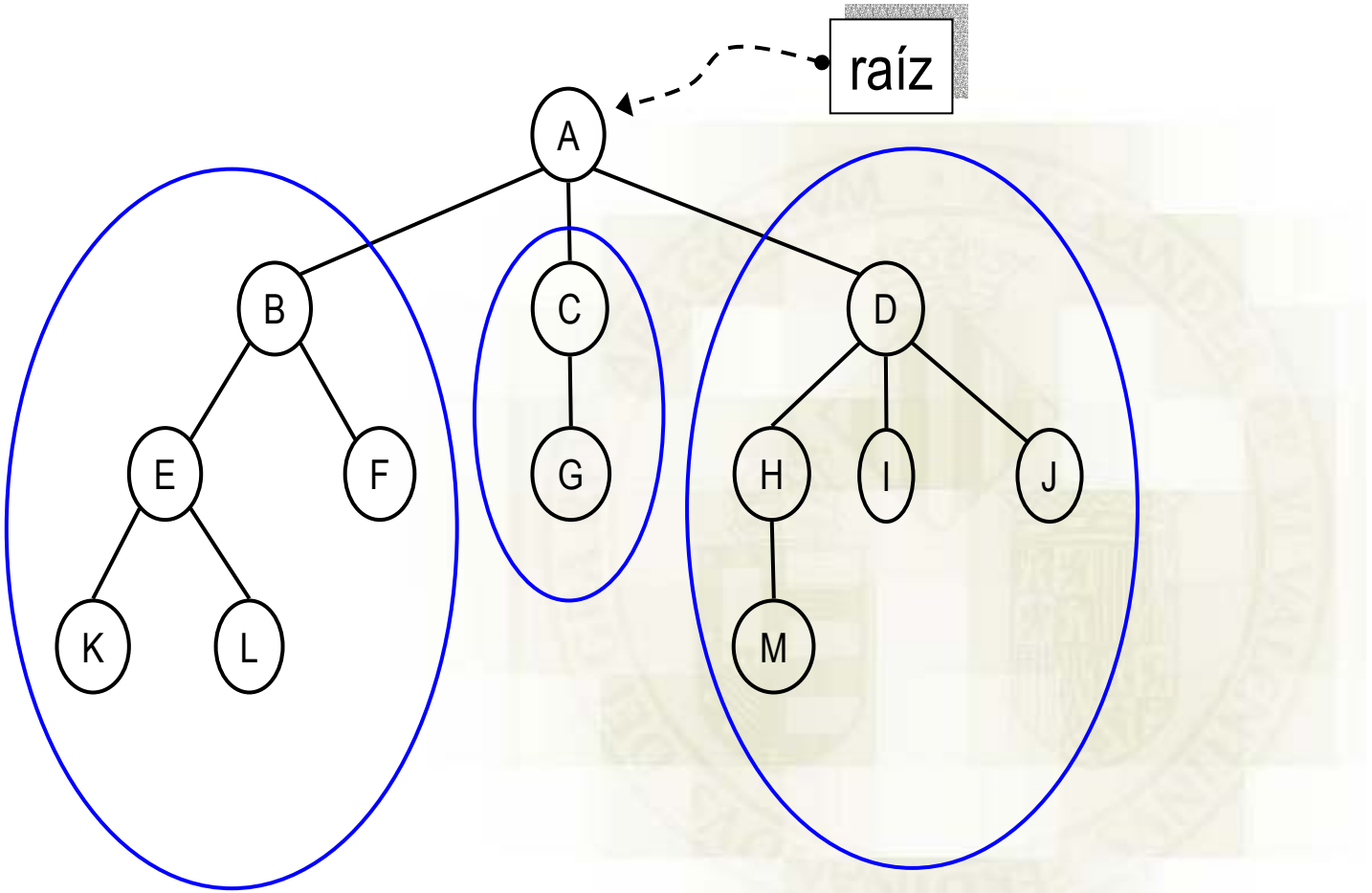
  (ii) Un conjunto finito de uno o más nodos, tal que existe un nodo especial, llamado nodo raíz, y donde los restantes nodos están separados en $n \geq 0$ conjuntos disjuntos, cada uno de los cuales es a su vez un árbol (llamados subárboles del nodo raíz).

- La definición implica que cada nodo del árbol es raíz de algún subárbol contenido en el árbol principal.

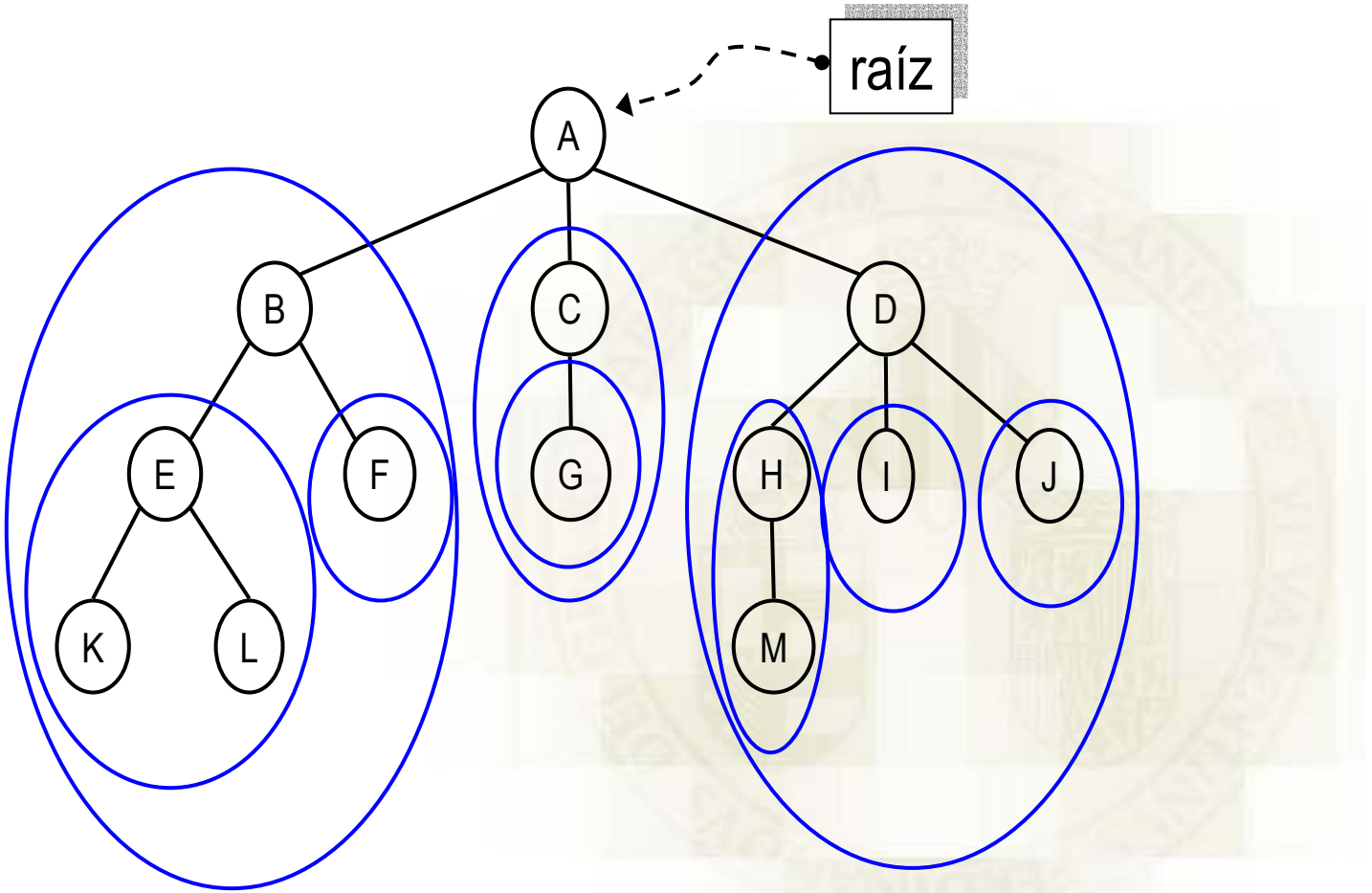
Ejemplo de Árbol



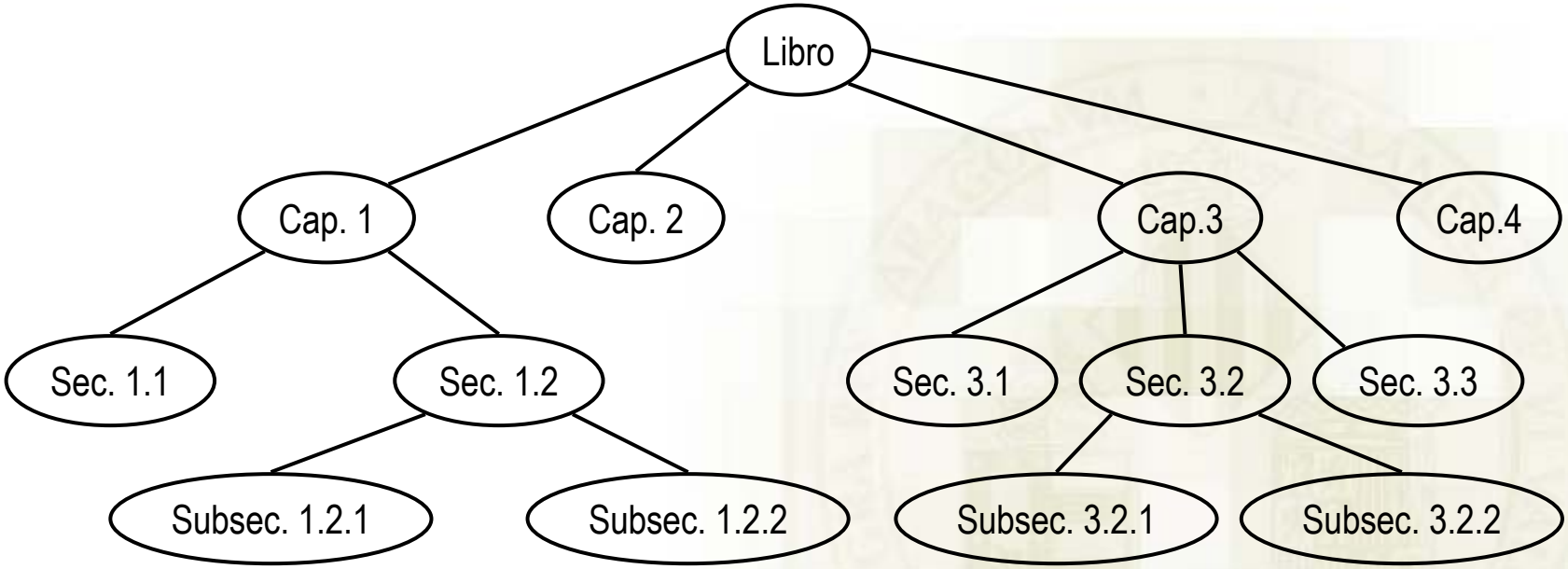
Ejemplo de Árbol



Ejemplo de Árbol



Ejemplo de Árbol (LIBRO)



Terminología

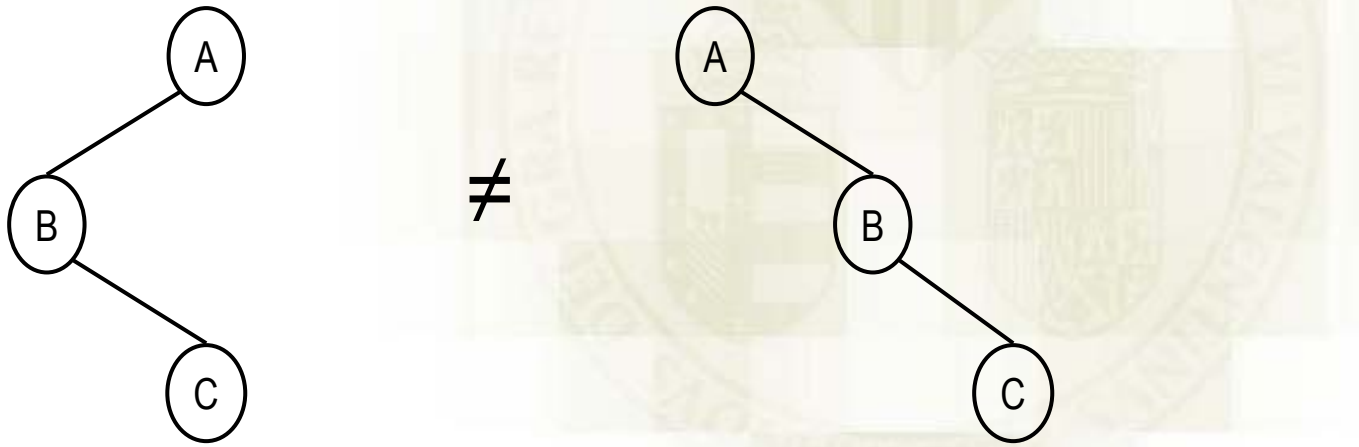
- **Grado de un nodo**: Número de subárboles que tienen como raíz ese nodo (el número de subárboles que "cuelgan" del nodo).
- **Nodo terminal**: Nodo con grado 0, no tiene subárboles.
- **Grado de un árbol**: Grado máximo de los nodos de un árbol.
- **Hijos de un nodo**: Nodos que dependen directamente de ese nodo, es decir, las raíces de sus subárboles.
- **Padre de un nodo**: Antecesor directo de un nodo, nodo del que depende directamente.
- **Nodos hermanos**: Nodos hijos del mismo nodo padre.
- **Camino**: Sucesión de nodos del árbol $n_1, n_2, \ldots, n_k$, tal que n_i es el padre de n_{i+1}.

Terminología (2)

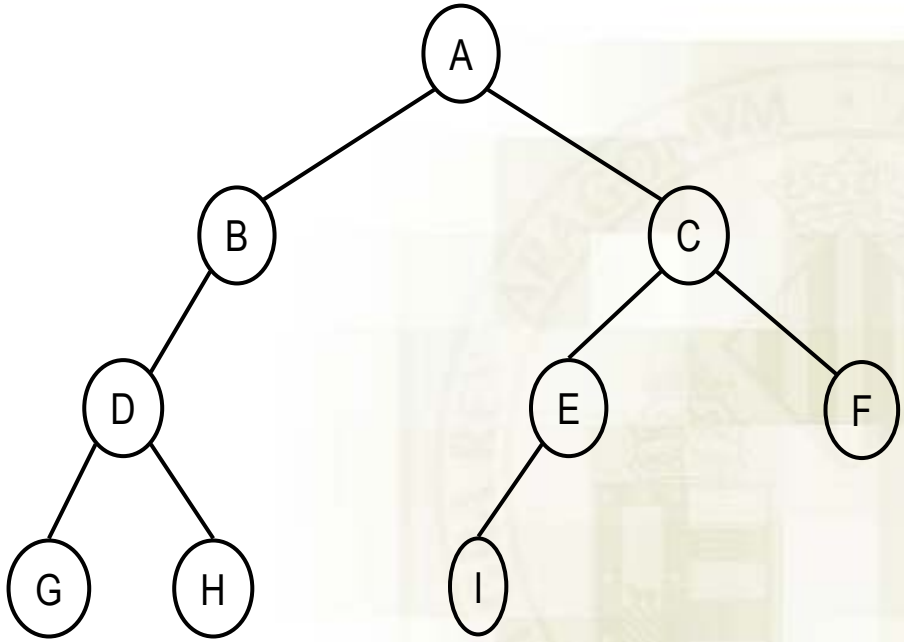
- **Antecesores de un nodo**: Todos los nodos en el camino desde la raíz del árbol hasta ese nodo.
- **Nivel de un nodo**: Longitud del camino desde la raíz hasta el nodo.
- **Profundidad de un nodo**: Nivel de un nodo.
- **Altura (profundidad) de un árbol**: Nivel máximo de un nodo en el árbol.
- **Altura de un nodo**: Altura del árbol que lo tiene como raíz.
- **Longitud de camino de un árbol**: Suma de las longitudes de los caminos a todos sus componentes.
- **Bosque**: Conjunto de n>0 árboles disjuntos.

Árboles Binarios

- Árboles de grado 2.
- **Definición**: Un árbol binario es un conjunto finito de nodos que puede estar vacío o consistir en un nodo raíz y dos árboles binarios disjuntos, llamados *subárbol izquierdo* y *subárbol derecho*.



Ejemplo de Árbol Binario



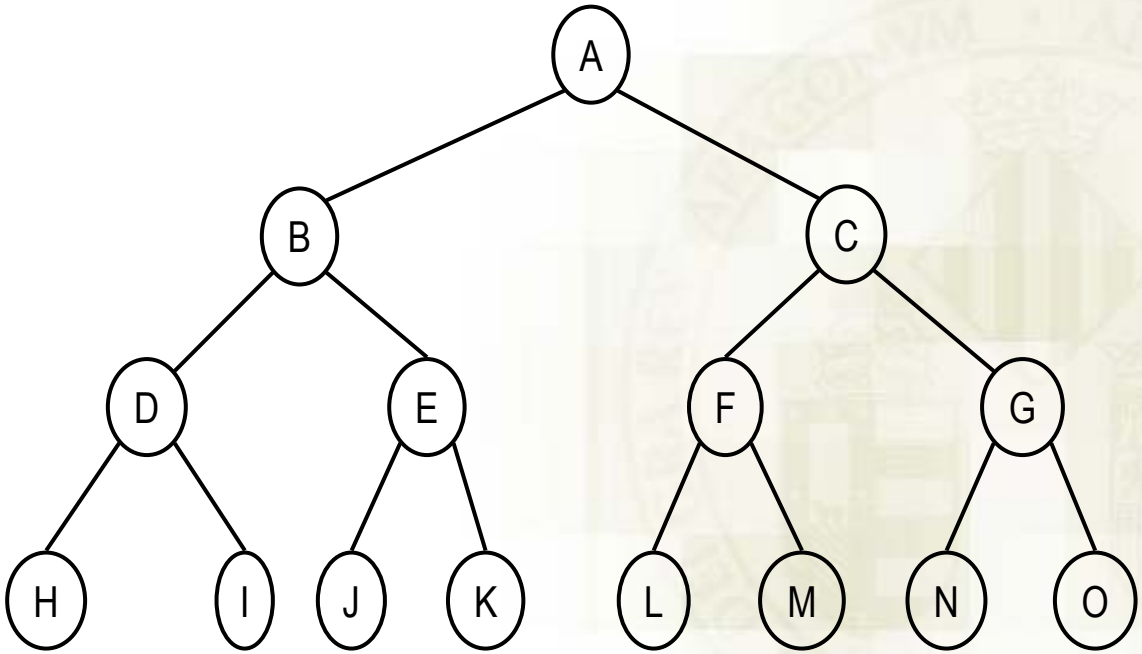
Propiedades de los Árboles Binarios

1. El número máximo de nodos en el nivel (profundidad) i de un árbol binario es 2^{i-1}, $i \geq 1$, y el número máximo de nodos en un árbol binario de altura k es 2^k-1, $k \geq 1$.

2. Para cualquier árbol binario no vacío, si n_0 es el número de nodos terminales y n_2 es el número de nodos de grado 2, entonces se cumple que $n_0 = n_2 + 1$.

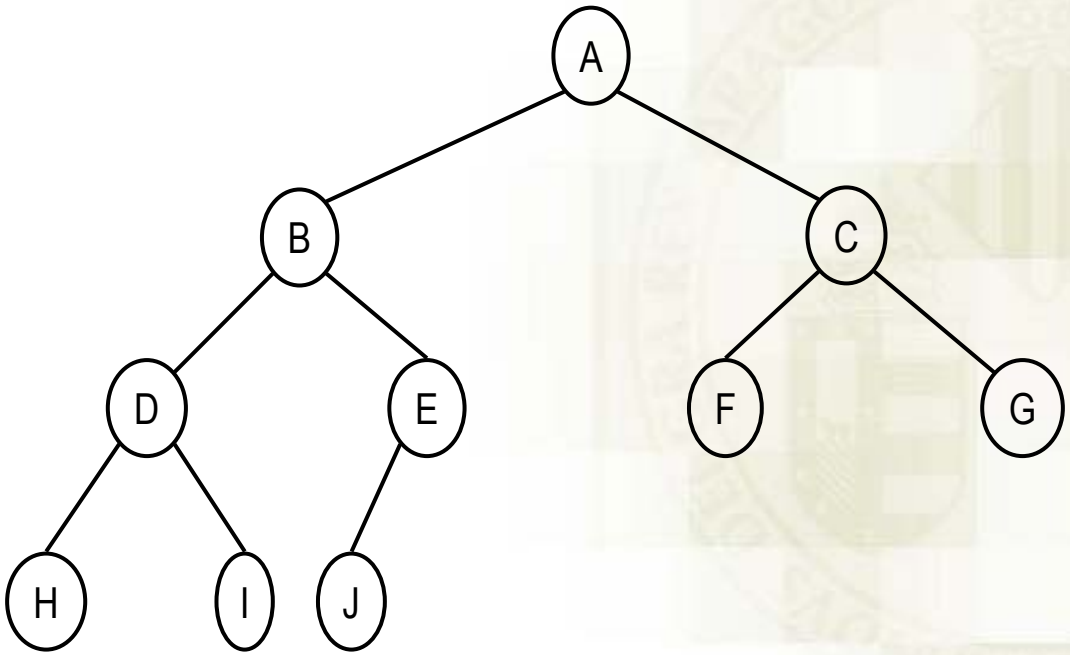
Definiciones

- Árbol binario **lleno**: Se dice que un árbol binario está lleno si es un árbol binario de altura k que tiene 2^k-1 nodos.



Definiciones

- Árbol binario **completo**: Se dice que un árbol binario de altura k está completo si está lleno hasta altura k-1 y el último nivel esta ocupado de izquierda a derecha.



TAD Árbol Binario

Operaciones:

Axiomas:

TAD Árbol Binario

Operaciones:

`IniciarArbol ( ) → AB`

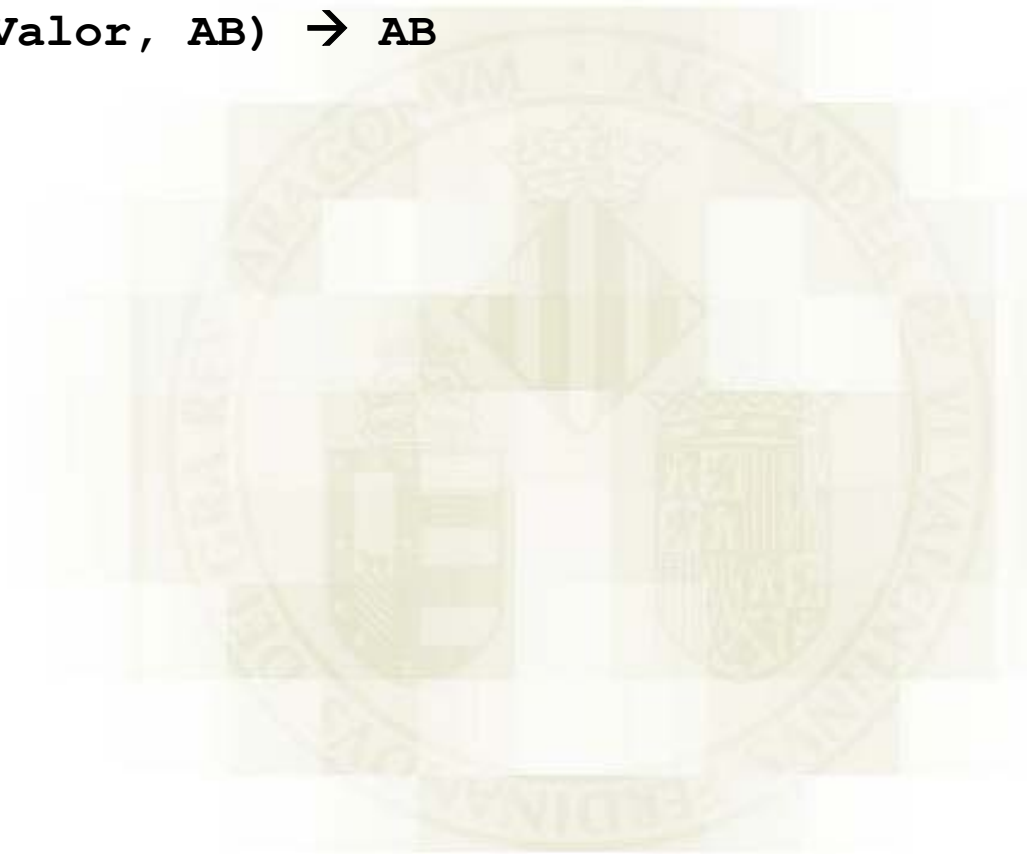
Axiomas:

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
```

Axiomas:

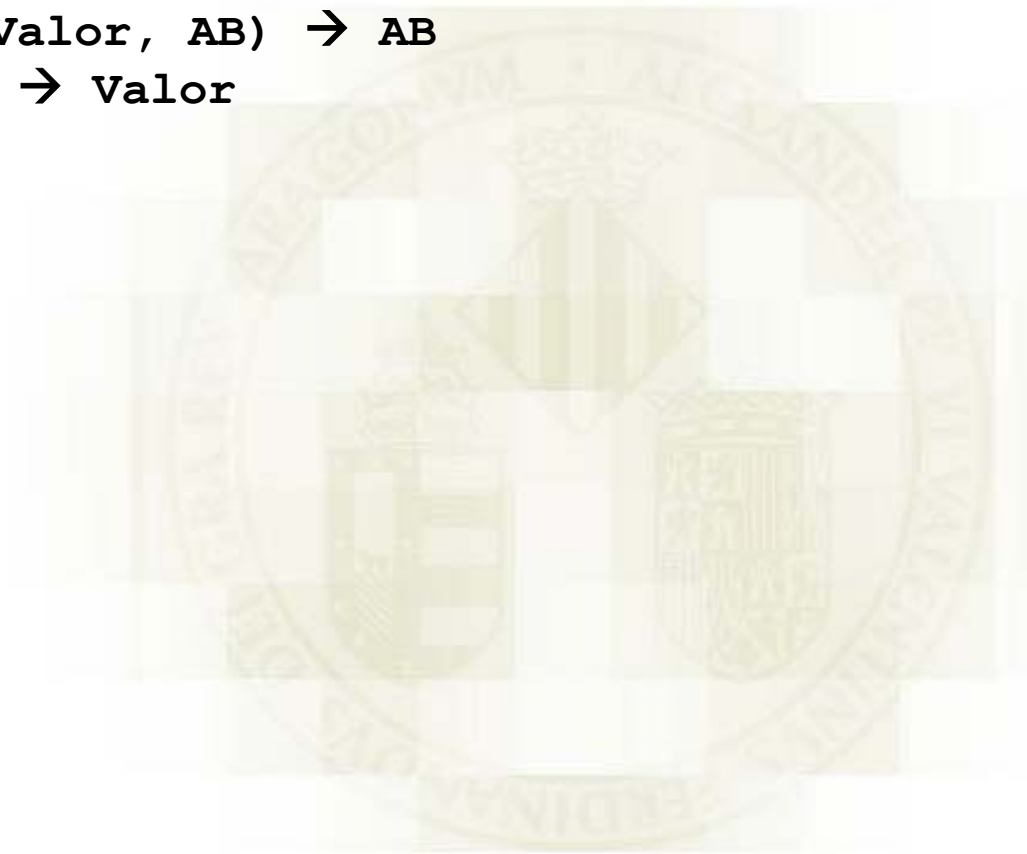


TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
```

Axiomas:



TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
```

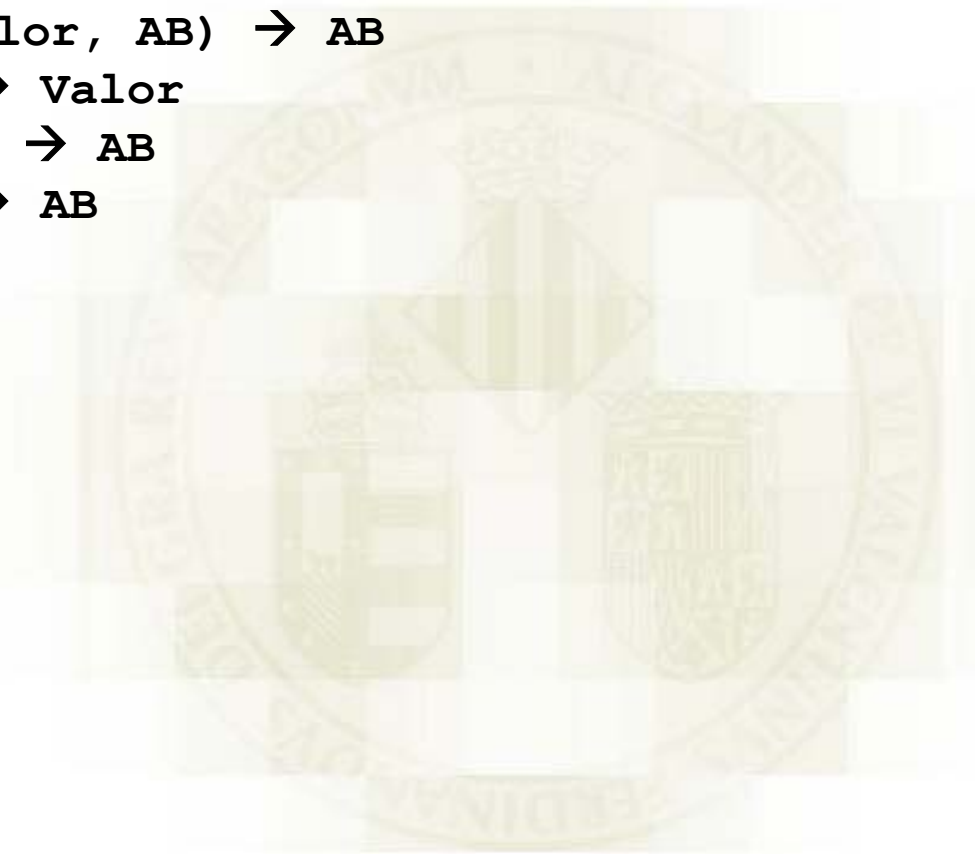
Axiomas:

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
```

Axiomas:



TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` y `d ∈ Valor`, se cumple que:

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` y `d ∈ Valor`, se cumple que:

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
HijoIzquierdo ( IniciarArbol ( ) ) → ERROR
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
HijoIzquierdo ( IniciarArbol ( ) ) → ERROR
HijoDerecho ( HacerArbol ( p, d, r ) ) → r
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
HijoIzquierdo ( IniciarArbol ( ) ) → ERROR
HijoDerecho ( HacerArbol ( p, d, r ) ) → r
HijoDerecho ( IniciarArbol ( ) ) → ERROR
```

TAD Árbol Binario

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
HijoIzquierdo ( IniciarArbol ( ) ) → ERROR
HijoDerecho ( HacerArbol ( p, d, r ) ) → r
HijoDerecho ( IniciarArbol ( ) ) → ERROR
Informacion ( HacerArbol ( p, d, r ) ) → d
```

TAD Árbol Binario (AB)

Operaciones:

```
IniciarArbol ( ) → AB
HacerArbol (AB, Valor, AB) → AB
Informacion (AB) → Valor
HijoIzquierdo (AB) → AB
HijoDerecho (AB) → AB
ArbolVacio (AB) → Lógico
```

Axiomas:

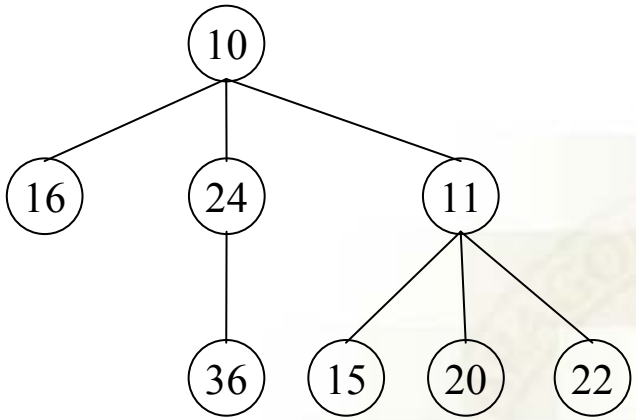
Sean `p, r ∈ AB` **y** `d ∈ Valor`, **se cumple que:**

```
ArbolVacio ( IniciarArbol ( ) ) → cierto
ArbolVacio ( HacerArbol ( p, d, r ) ) → falso
HijoIzquierdo ( HacerArbol ( p, d, r ) ) → p
HijoIzquierdo ( IniciarArbol ( ) ) → ERROR
HijoDerecho ( HacerArbol ( p, d, r ) ) → r
HijoDerecho ( IniciarArbol ( ) ) → ERROR
Informacion ( HacerArbol ( p, d, r ) ) → d
Informacion ( IniciarArbol ( ) ) → ERROR
```

Clase Árbol Binario (AB)

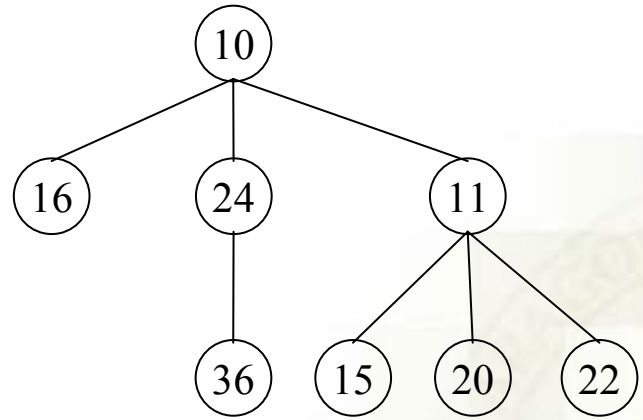
```
class AB
{
    public:
        AB ();
        AB (const AB &);
        ~AB ();

        void HacerArbol (AB&, Valor, AB&);

        bool Informacion (Valor &);
        AB& HijoIzdo ();
        AB& HijoDcho ();
        bool ArbolVacio ();

   private:
       ...
};
```

Árboles generales (grado k)

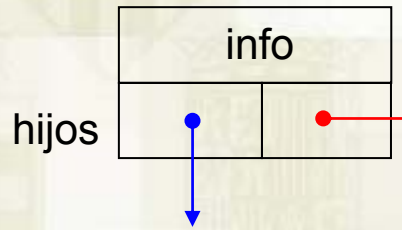


Ejemplo k =3

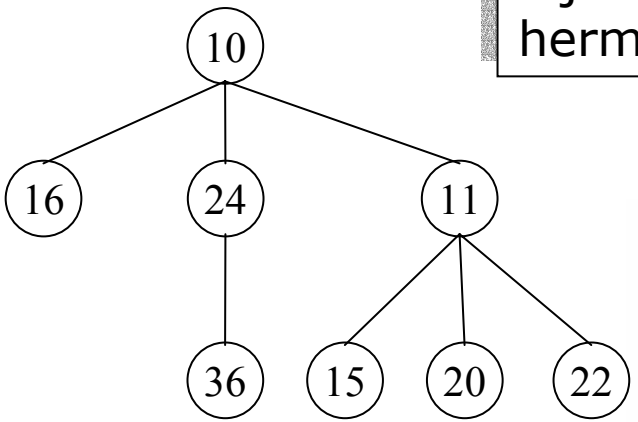
Árboles generales (grado k)



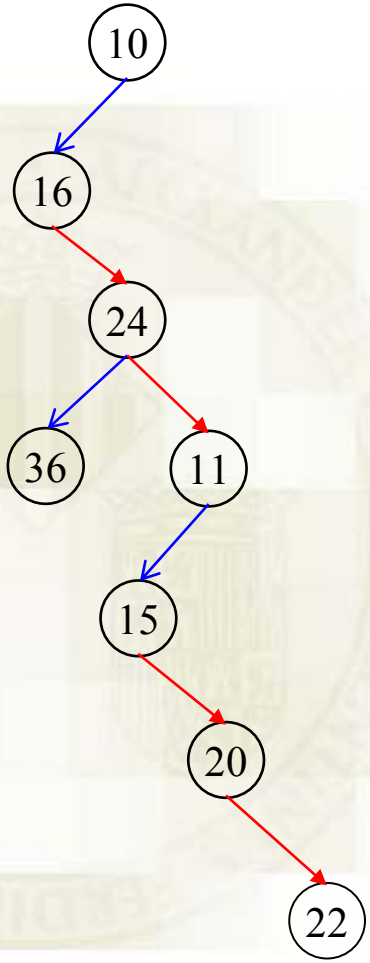
Ejemplo k =3



Árboles generales (grado k)



hijos = HijoIzq
hermano = HijoDer

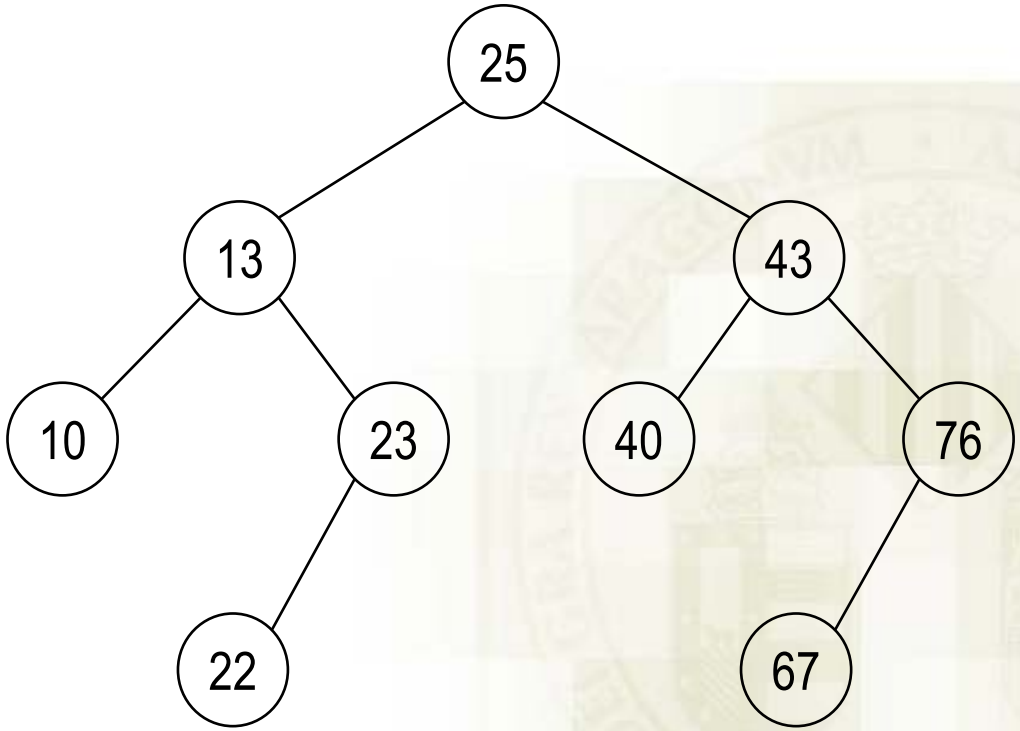


Árboles Binarios de Búsqueda

- **Definición**: Un árbol binario de búsqueda es un árbol binario, que puede estar vacío, y que si es no 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.

Ejemplo: A.B. Búsqueda



Clase Árbol Binario de Búsqueda (ABB)

```
class ABB
{
    public:
        ABB ();
        ABB (const ABB &);
        ~ABB ();
        const ABB& operator= (const ABB&);

        bool Informacion (Valor &);
        ABB& HijoIzdo ();
        ABB& HijoDcho ();
        bool ABBVacio ();

        ABB& Buscar (Valor);
        bool Insertar (Valor);
        bool Eliminar (Valor);
    private:
        ...
};
```

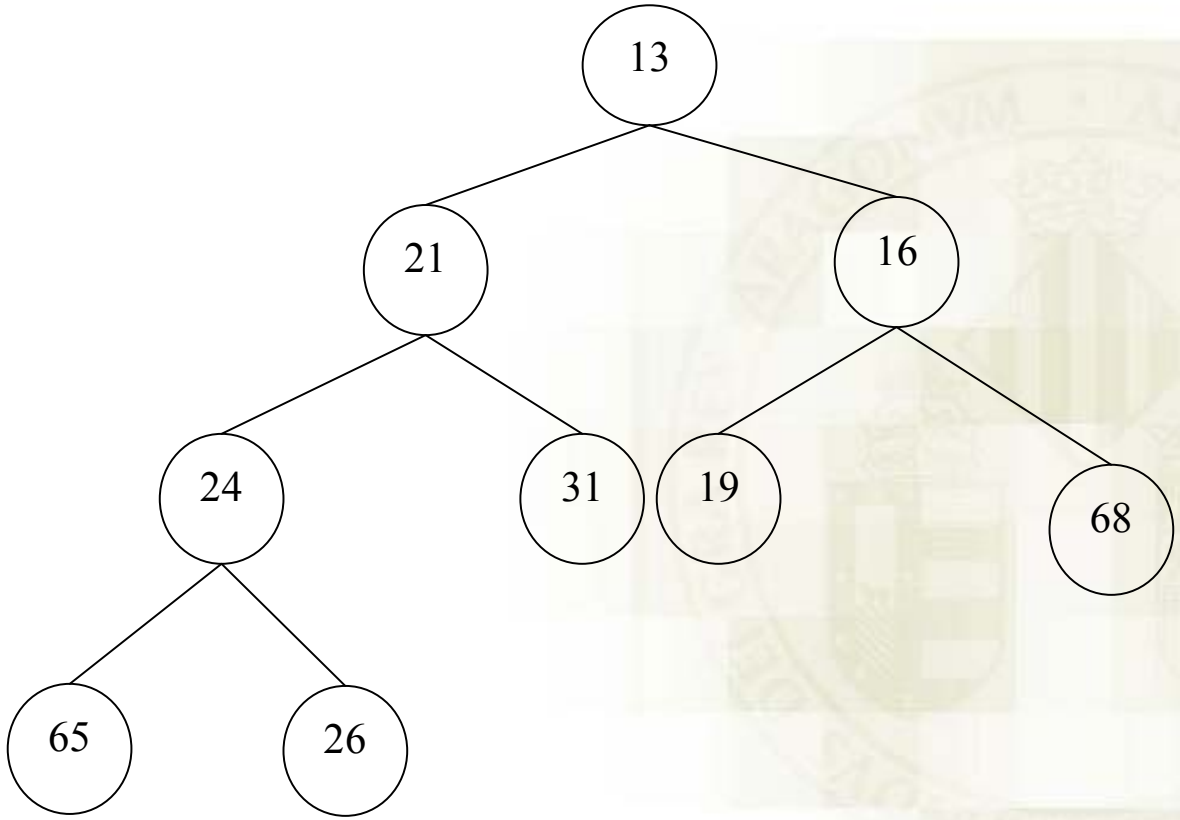
Montículos: Fundamentos

- El problema de la selección:
  - ✓ Seleccionar de entre un conjunto extenso de valores un elemento extremo (máximo o mínimo).
- Aplicación:
  - ✓ Colas de prioridad
- Montículos

Montículos binarios (de mínimos)

- Definición:
  - ✓ Árbol binario completo.
  - ✓ Para todo nodo x se cumple que el valor de x (prioridad) es menor o igual que los valores de sus nodos hijos.

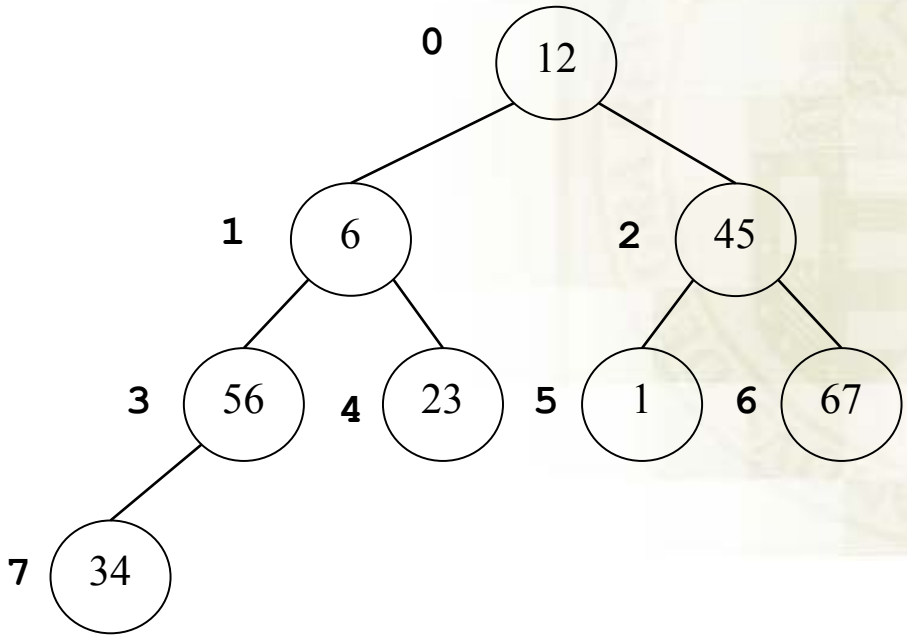
Montículos: Ejemplo



Hacer Montículo un array

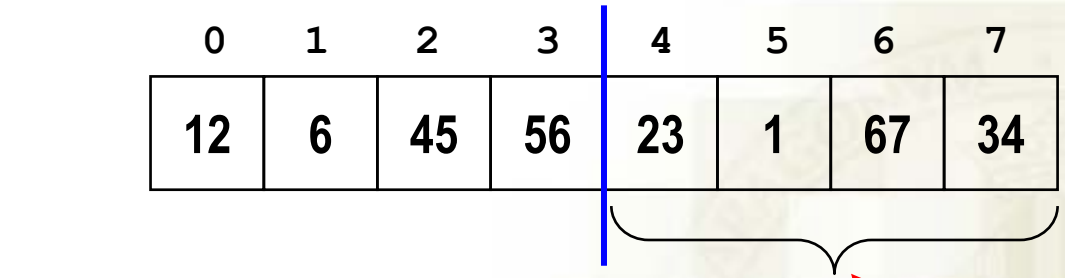
- Todo array se puede ver como un árbol binario completo.

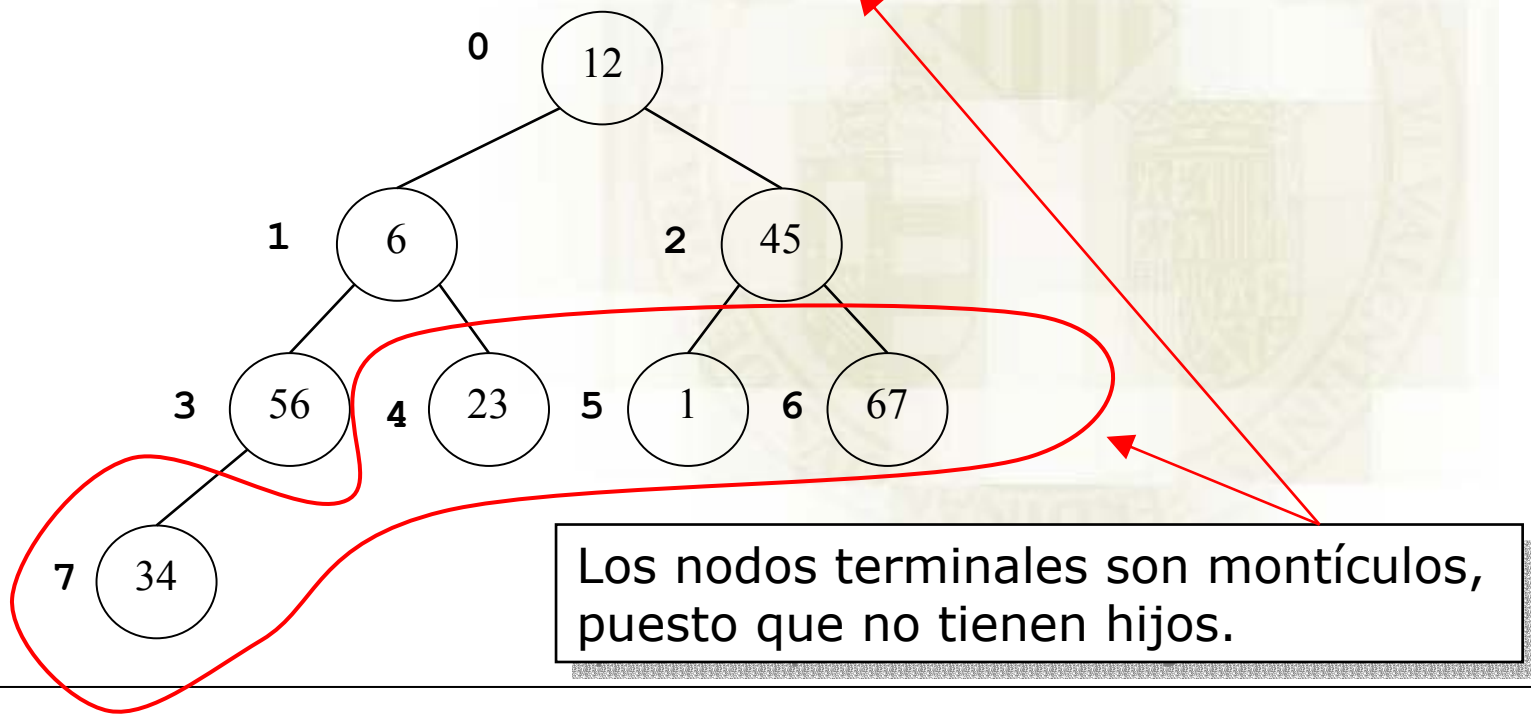
0	1	2	3	4	5	6	7
12	6	45	56	23	1	67	34



Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).





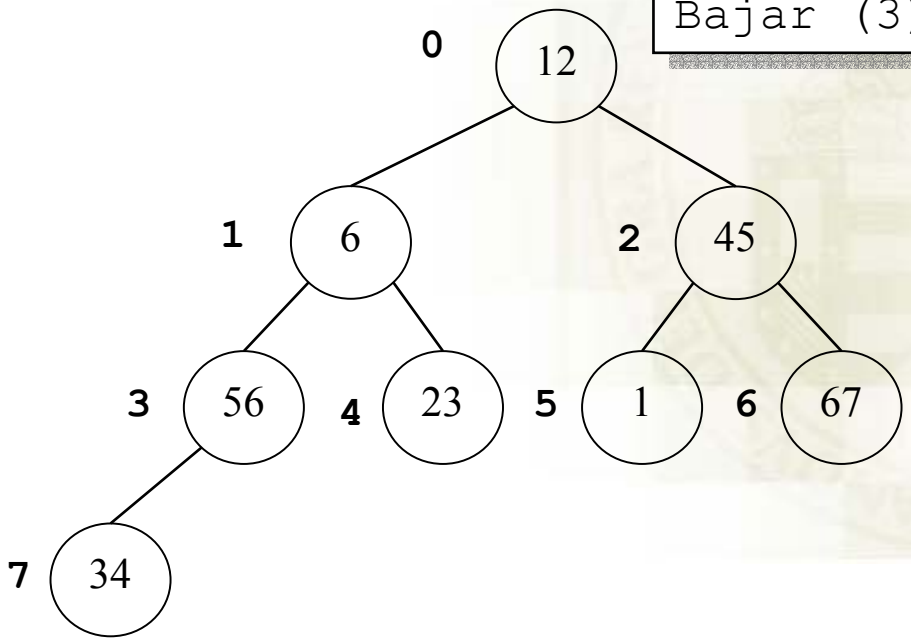
Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).

0	1	2	3	4	5	6	7
12	6	45	56	23	1	67	34

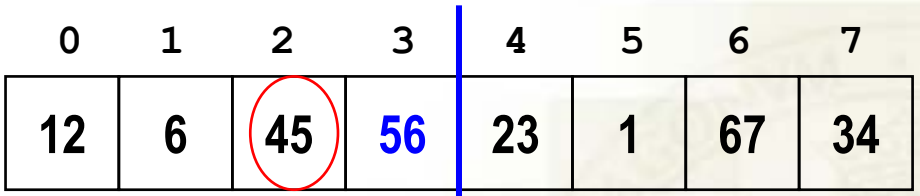
Considerar montículo de máximos.

`Bajar (3)`, llevará 56 a su lugar

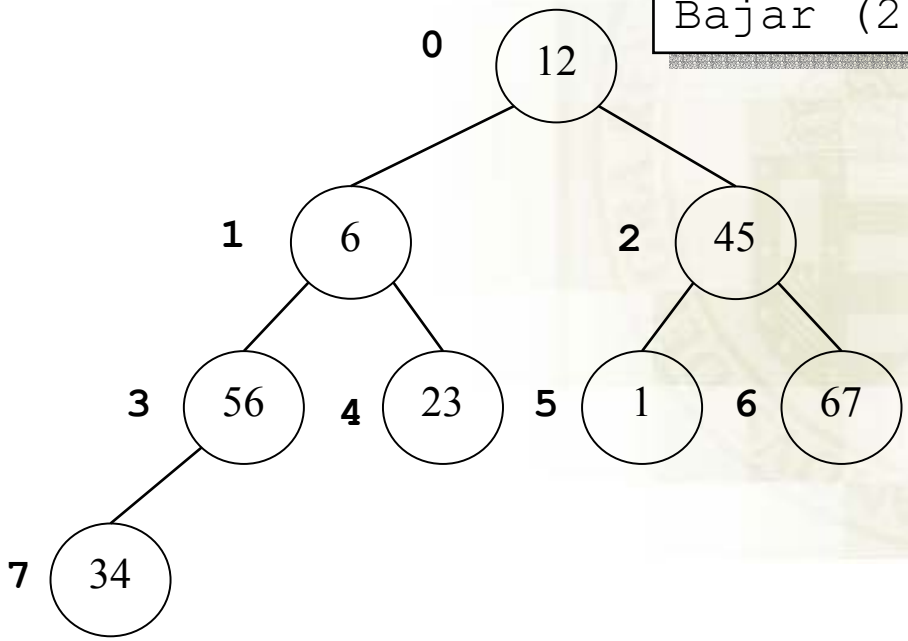


Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).

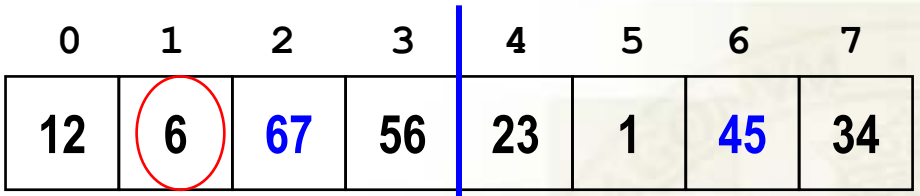


`Bajar (2)`, llevará 45 a su lugar

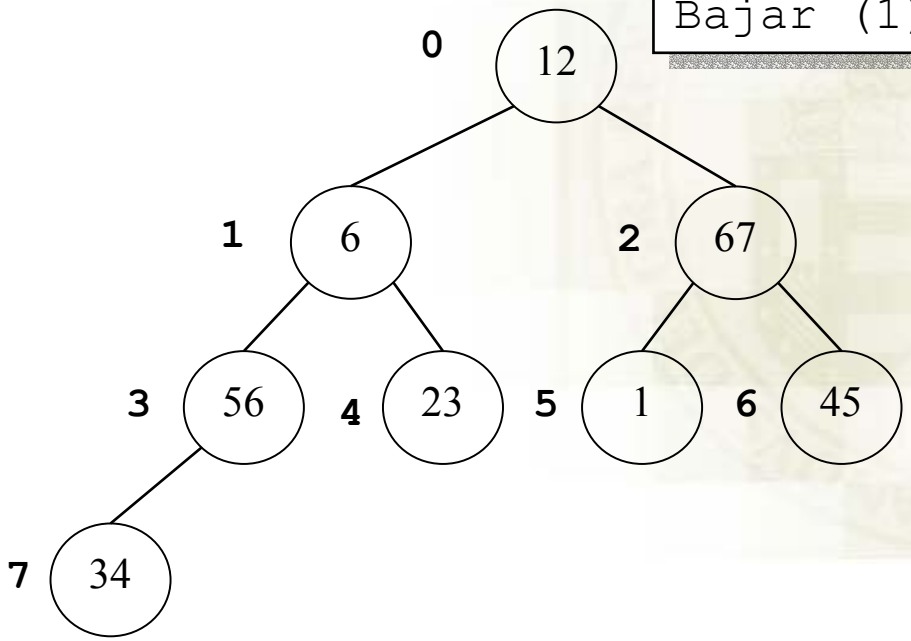


Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).

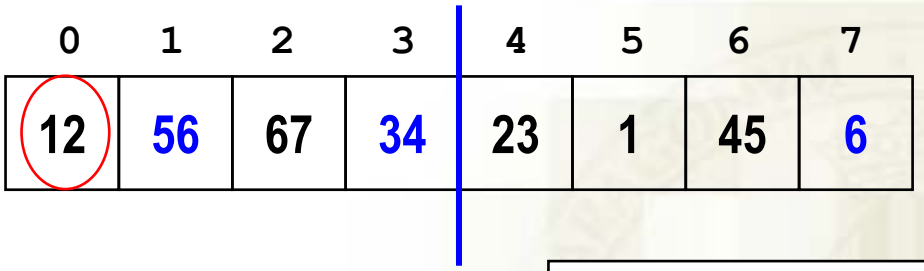


`Bajar (1)`, llevará 6 a su lugar

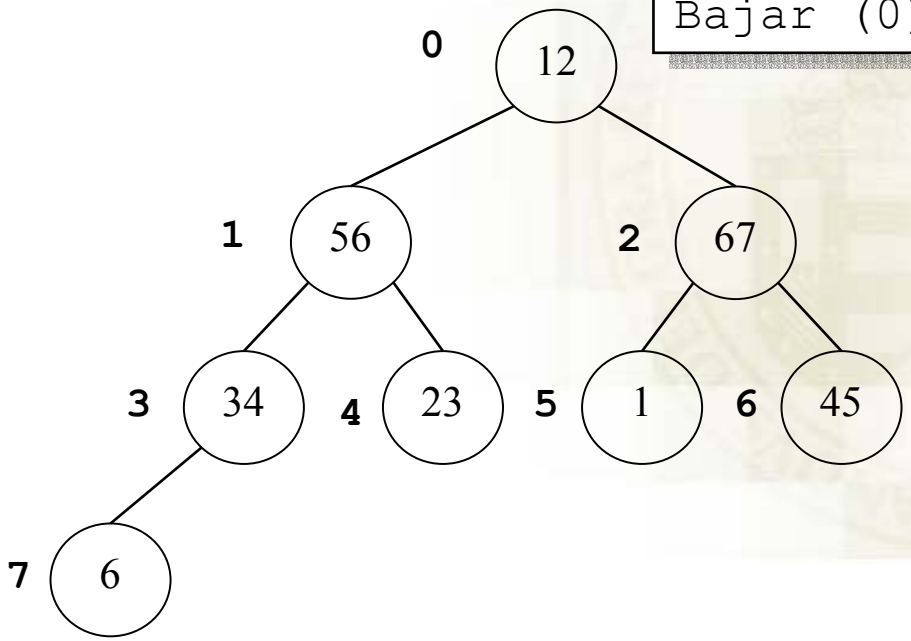


Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).

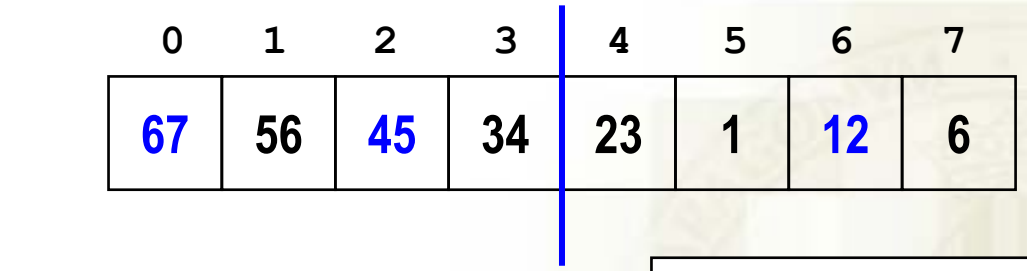


`Bajar (0)`, llevará 12 a su lugar

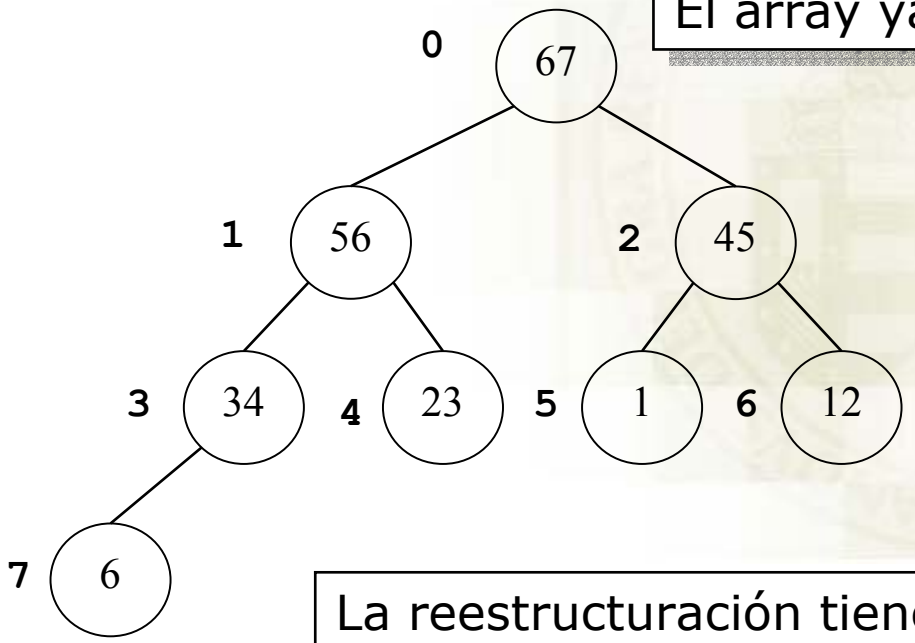


Hacer Montículo un array

- Convertir un array en un montículo consiste en organizar bien los datos (Subir o Bajar hasta su posición correcta).



El array ya es un montículo



La reestructuración tiene un coste lineal, $O(n)$

Ordenación mediante montículos

- Se pueden utilizar los montículos como método alternativo para ordenar vectores → Método *HeapSort*
- El coste medio del algoritmo HeapSort es igual que el del método QuickSort → *O* (*n.lg n*)
- El método consiste en:
  1. Reestructurar el array para convertirlo en un montículo de máximos, según el esquema anterior.
  2. Seleccionar el máximo (primer elemento) e intercambiarlo con el último elemento del vector. El máximo ocupa su posición final.
  3. Considerar que el montículo tienen un elemento menos.
  4. Aplicar Bajar(0) para colocar el elemento que está mal en el montículo.