

# Árboles

## Definiciones

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

20 de julio de 2021



- Obsérvese que, aunque las definiciones siguientes son tipos de datos recursivos, éstos se pueden programar utilizando dos métodos:
  - 1 Con referencias de unos miembros a otros.
  - 2 Colocando a los miembros dentro de arreglos de espacio contiguo.

## Árboles

- 1 Árboles
- 2 Recorridos

# Temas

## 1 Árboles

- Definiciones
- Características
- Árboles n-arios

# Definición de árbol 1

Un árbol es una colección de elementos llamados *nodos*, uno de los cuales se distingue como *raíz*, junto con una relación de «paternidad» que impone una estructura jerárquica sobre los nodos Vargas Villazón, Lozano Moreno y Levine Gutiérrez 1998.

Formalmente:

- 1 Un solo nodo es, por sí mismo, un árbol. Ese nodo es también la raíz de dicho árbol.
- 2 Supóngase que  $n$  es un nodo y que  $A_1, A_2, \dots, A_k$  son árboles con raíces  $n_1, n_2, \dots, n_k$ , respectivamente. Se puede construir un árbol nuevo haciendo que  $n$  se constituya en el padre de los nodos  $n_1, n_2, \dots, n_k$ . En dicho árbol,  $n$  es la raíz y  $A_1, A_2, \dots, A_k$  son los *subárboles* de la raíz. Los nodos  $n_1, n_2, \dots, n_k$  reciben el nombre de *hijos* del nodo  $n$ .

## Definición de árbol 2

Un árbol  $T$  es un conjunto de nodos finito y distinto del vacío.

$$T = r \cup T_1 \cup T_2 \cup \dots \cup T_n = \{r, T_1, T_2, \dots, T_n\} \quad (1)$$

con las siguientes propiedades:

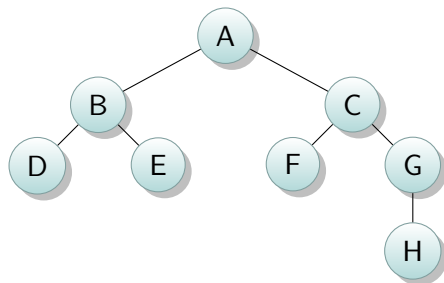
- ① Se designa a un nodo  $r$ , como la raíz del árbol.
- ② Los nodos restantes se dividen en  $m \geq 0$  conjuntos ajenos (subconjuntos)  $T_1, T_2, \dots, T_n$ , los cuales son a su vez un árbol

$$T_C = \{D, \{E, \{F\}\}, \{G, \{H, \{I\}\}, \{J, \{K\}, \{L\}\}, \{M\}\} \quad (2)$$

# Ejemplos

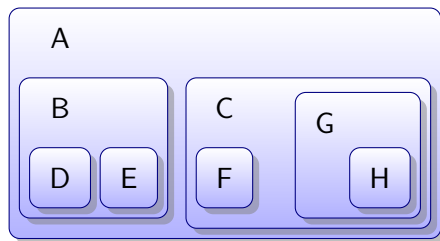
- 1 elemento:  $T_a = \{A\}$
- 2 elementos:  $T_b = \{B, \{C\}\}$
- varios elementos:  $T = \{A, \{B, \{D\}, \{E\}\}, \{C, \{F\}, \{G, \{H\}\}\}$

Como gráfica:



*Ojo: es necesario determinar la raíz, pues en principio se puede tomar cualquier nodo y de él cuelgan todos los demás.*

Como conjuntos:





# Código Java

## Código 1: Árbol

```
1 import java.util.List;
2 import java.util.LinkedList;
3
4 public class Árbol<E> {
5     public static class Nodo<E> {
6         private E dato;
7         private List< Nodo<E> > hijos;
8
9         public Nodo(E dato) {
10             this.dato = dato;
11             this.hijos = new LinkedList<>();
12         }
13     }
14
15     private Nodo<E> raíz;
16 }
```

# Temas

## 1 Árboles

- Definiciones
- Características
- Árboles n-arios

# Terminología

Considérese el árbol  $T = \{r, T_1, T_2, \dots, T_n\}$ ,  $n \geq 0$ .

**Grado de un nodo:** el número de subárboles asociados a él.

Ej:  $\text{grado}(T) = n$

**Hoja:** un nodo de grado cero (no tiene subárboles).

**Hijo:** la raíz  $r_i$  del subárbol  $T_i$  del árbol  $r$  es *hija* de  $r$ . El término *nieto* se define análogamente.

**Padre:** el nodo raíz  $r$  del árbol  $T$  es el *padre* de todas las raíces  $r_i$  de los subárboles  $T_i$ ,  $1 \leq i \leq n$ . El término *abuelo* se define análogamente. La relación Padre-Hijo es representada por una *arista* dirigida.

**Hermanos:** dos raíces  $r_i$  y  $r_j$  de subárboles distintos  $T_i$  y  $T_j$  de un árbol  $T$  son *hermanos*.

**Nodo intermedio:** un nodo que no es raíz ni hoja.

# Camino

## Definición:

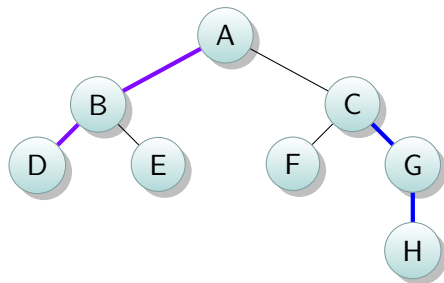
- Dado un árbol  $T$  que contiene al conjunto de nodos  $R$ , un *camino* en  $T$  se define como una secuencia de nodos distinta del vacío:

$$C = \{r_1, r_2, \dots, r_k\}, \quad (3)$$

donde  $r_i \in R$ , para  $1 \leq i \leq k$  tal que el  $i$ -ésimo nodo en la secuencia  $r_i$ , es el padre del  $\langle i + 1 \rangle$ -ésimo nodo en la secuencia  $r_{i+1}$ .

- La longitud del camino  $C$  es  $k - 1$  (i.e. el número de aristas recorridas).

# Caminos



# Terminología II

**Nivel o Profundidad** de un *nodo*  $r_i \in R$  en un árbol  $A$ : la longitud del único camino en  $A$  desde su raíz  $r$  al nodo  $r_i$ .

$$\text{raíz} \rightarrow \text{nivel } 0 \quad (4)$$

**Altura de un nodo**  $r_i \in R$  en un árbol  $A$  es la longitud del camino más largo del nodo  $r_i$  a una hoja.

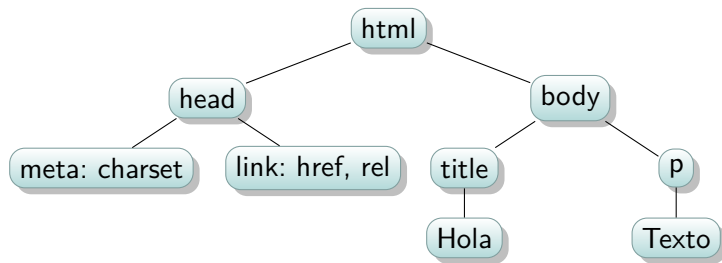
**Altura de un árbol**  $A$ : la altura de su nodo raíz.

**Ancestro:** Sean  $r_i$  y  $r_j$  dos nodos del árbol  $A$ ,  $r_i$  es *ancestro* de  $r_j$  si existe un camino en  $A$  de  $r_i$  a  $r_j$ .  $r_i$  es *ancestro propio* de  $r_j$  si  $r_i \neq r_j$  ( $\text{longitud}(c) \neq 0$ ).

**Descendiente y descendiente propio**  $\Leftrightarrow \exists P$  de  $r_i$  a  $r_j$

# Ejemplo de aplicación

```
1 <html>
2   <head>
3     <meta charset="utf-8" />
4     <link href="./styles/coollook.css" rel="stylesheet"/>
5   </head>
6   <body>
7     <title>Hola</title>
8     <p>Texto</p>
9   </body>
10 </html>
```





# Temas

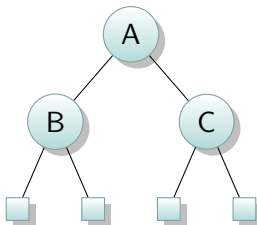
- 1 Árboles
  - Definiciones
  - Características
  - Árboles n-arios
    - Árboles n-arios en arreglos
    - Árboles binarios

# Árboles n-arios

## Definición

Un árbol *n-ario* es aquel en que todos sus nodos tienen **exactamente**  $n$  hijos.

Destacan los árboles *binarios*, en los que cada nodo tiene exactamente dos hijos.



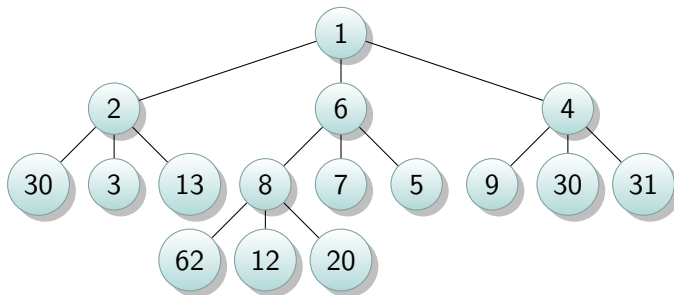
# Árboles n-arios

## Definición:

Un árbol *n-ario* es un conjunto  $T$  de nodos finito con las siguientes propiedades:

- 1 El conjunto es vacío,  $T = \emptyset$ , o
- 2 El conjunto consiste en una raíz  $r$  y exactamente  $n$  subárboles *n-arios* Preiss 1999.

Ej:  $n = 3$

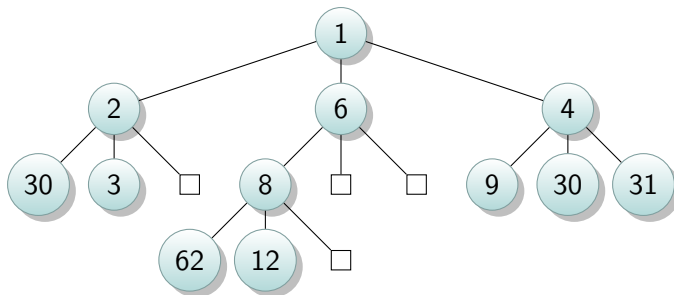


# (Árboles n-arios, definición casi equivalente)

Obsérvese que algunos de los subárboles pueden ser vacíos, por lo que en ocasiones se propone como equivalente la definición:

Un árbol *n-ario* es:

- 1 El árbol vacío,  $T = \emptyset$ , o
- 2 Una raíz  $r$  y a lo más  $n$  subárboles  $n$ -arios.

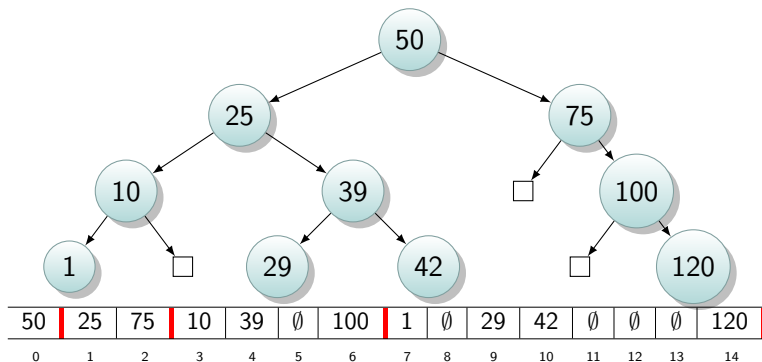


# Código Java

## Código 2: Árbol n-ario

```
1 import java.lang.reflect.Array;
2
3 public class ÁrbolNario<E> {
4     class Nodo<E> {
5         private E dato;
6         private Nodo<E>[] hijos;
7
8         @SuppressWarnings("unchecked")
9         public Nodo(E dato) {
10             if (dato == null) throw new NullPointerException("Dato_nulo.");
11             this.dato = dato;
12             hijos = (Nodo<E>[]) Array.newInstance(this.getClass(), grado);
13         }
14     }
15
16     private int grado;
17     private Nodo<E> raíz;
18
19     public ÁrbolNario(int grado) {
20         if (grado <= 0) throw new IllegalArgumentException("Se_necesita_al_menos_un_hijo");
21         this.grado = grado;
22     }
23 }
```

# Ejemplo: árbol binario



$\text{hijoIzquierdo}(i) \rightarrow 2i + 1$      $\text{padre}(i) \rightarrow \lfloor (i - 1)/2 \rfloor$

$\text{hijoDerecho}(i) \rightarrow 2(i + 1)$

# Árboles n-arios en arreglos

Dado que el número de hijos en un árbol n-ario es fijo:

- El tamaño  $l$  del arreglo donde se almacenará al árbol es  $l = 2^h$  donde  $h$  es el altura del árbol.

# Acceso a hijos y padres

- Es posible determinar la posición del  $i$ -ésimo descendiente, dada la posición de su padre.

$$\text{hijo}_j(i) = ni + (j + 1) \quad (5)$$

donde  $i$  es la posición del nodo;  $j$ , el índice de su hijo, contando desde cero de izquierda a derecha y  $n$ , el grado del árbol.

- Y la posición del padre dada la posición del hijo:

$$\text{padre}(i) = \lfloor \frac{i-1}{n} \rfloor \quad (6)$$



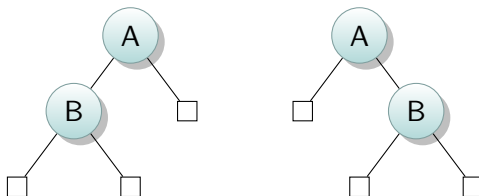
# Árboles binarios

## Definición:

Un árbol *binario* es un conjunto  $T$  de nodos finito con las siguientes propiedades:

- 1 El conjunto es vacío,  $T = \emptyset$ , o
- 2 El conjunto consiste en una raíz  $r$  y exactamente 2 subárboles binarios  $T_L$  y  $T_R$ .

$$T = \{r, T_L, T_R\} \quad (7)$$



# Código Java

## Código 3: Árbol binario

```
1 public class Árbol<E> {
2     public static class Nodo<E> {
3         private E dato;
4         private Nodo<E> hijoI;
5         private Nodo<E> hijoD;
6
7         public Nodo(E dato, Nodo<E> hi, Nodo<E> hd) {
8             if (dato == null) throw new NullPointerException("Dato_nulo.");
9             this.dato = dato;
10            this.hijoI = hi;
11            this.hijoD = hd;
12        }
13    }
14
15    private Nodo<E> raíz;
16 }
```

# Recorridos

1 Árboles

2 Recorridos

# Temas

## 2 Recorridos

- Tipos de recorridos
  - Recorrido postorden
  - Recorrido preorden
  - Recorrido inorden
  - Recorrido en amplitud

# Recorridos

- Cualquier árbol puede ser recorrido en los órdenes siguientes:
  - Amplitud:** Primero la raíz, luego los nodos a profundidad 1, 2, ... , etc.
  - Preorden:** Cada nodo es visitado antes que sus hijos.
  - Postorden:** Los hijos son visitados primero y posteriormente el nodo raíz del subárbol.
- Los árboles binarios también pueden ser recorridos en:
  - Inorden:** El subárbol izquierdo es visitado primero, luego el nodo raíz y finalmente el subárbol derecho.

# Recorridos

Resumiendo:

❶ En profundidad (por subárboles)

❶ Preorden:  $r \rightarrow T_L \rightarrow T_R$ .

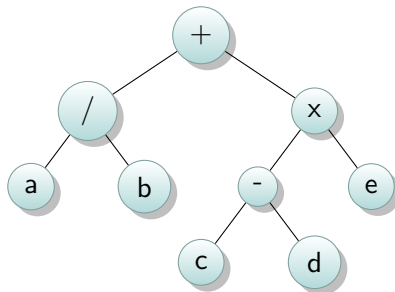
❷ Inorden:  $T_L \rightarrow r \rightarrow T_R$ .

❸ Postorden:  $T_L \rightarrow T_R \rightarrow r$ .

❷ En amplitud (por niveles)

# Ejemplos

$$a/b + (c - d)e$$



Preorden:  $+/ab \times -cde$  \*Prefija

Inorden:  $a/b + c - d \times e$  \*Ojo, no recuperó los paréntesis automáticamente, si se agregan se obtiene infija.

Postorden:  $ab/cd - e \times +$  \*Postfija

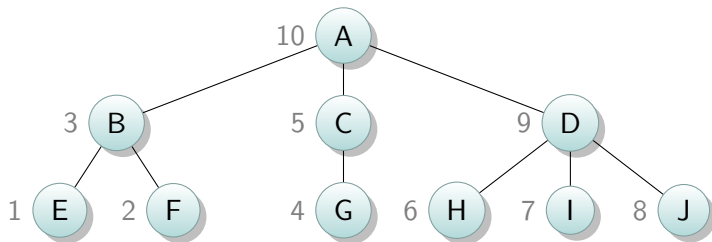
# Temas

## 2 Recorridos

- Tipos de recorridos
- Recorrido postorden
- Recorrido preorden
- Recorrido inorden
- Recorrido en amplitud



# Recorrido postorden



# Recorrido postorden con código recursivo

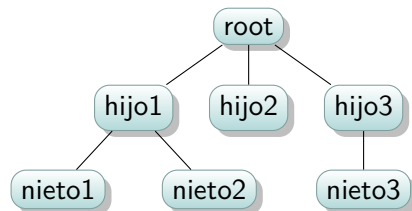
## Código 4: Postorden

```
1  import java.util.function.Consumer;
2  import java.util.List;
3  import java.util.LinkedList;
4
5  public class Nodo<E> {
6      private E dato;
7      private List< Nodo<E> > hijos;
8
9      public Nodo(E dato) {
10         this.dato = dato;
11         this.hijos = new LinkedList<>();
12     }
13
14     public void visitaPostorden(Consumer<E> f) {
15         // Visita a los hijos
16         for(Nodo<E> h : hijos) {
17             h.visitaPostorden(f);
18         }
19         // Visita este nodo
20         f.accept(dato);
21     }
22 }
```

# Ejemplo de uso

## Código 5: Uso postorden

```
1 public class UsaNodoArbol {
2     public static void main(String[] args) {
3         Nodo<String> raíz = new Nodo("root");
4         // hijos
5         List<Nodo<String>> hijos1 = raíz.getHijos();
6         hijos1.add(new Nodo("hijo1"));
7         hijos1.add(new Nodo("hijo2"));
8         hijos1.add(new Nodo("hijo3"));
9         // nietos
10        Nodo<String> h1 = raíz.getHijos().get(0);
11        h1.getHijos().add(new Nodo("nieto1"));
12        h1.getHijos().add(new Nodo("nieto2"));
13
14        Nodo<String> h3 = raíz.getHijos().get(2);
15        h3.getHijos().add(new Nodo("nieto3"));
16
17        raíz.visitaPostorden(new Consumer<String>() {
18            public void accept(String o) {
19                System.out.println(o);
20            }
21        });
22        h1.visitaPostorden(o -> System.out.println(o));
23    }
24 }
```



```
nieto1
nieto2
hijo1
hijo2
nieto3
hijo3
root
```

# Recorrido postorden con código iterativo

## Código 6: Postorden iterativo

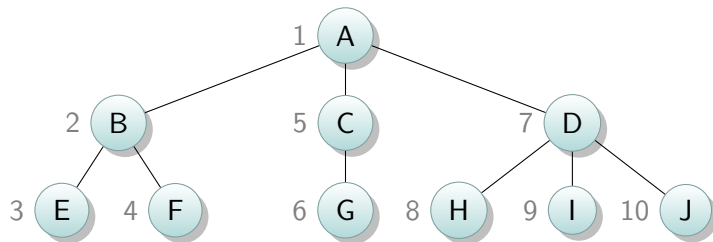
```
1 public class Nodo<E> {
2     private E dato;
3     private List< Nodo<E> > hijos;
4     private Nodo<E> padre;
5
6     public List<E> recorridoPostorden() {
7         List<E> l = new ListaDoblementeLigada<>();
8         boolean bajan = true;
9         Nodo<E> actual, hermano;
10        actual = raíz;
11        while(actual != null) {
12            // Llegar al fondo de la rama
13            if(bajan) { while(!actual.esHoja()) { actual = actual.getHijo(0); } }
14            l.add(actual.getElemento()); // Visita
15            if(actual.getPadre() != null && // Hermano siguiente
16                (hermano = actual.getPadre().getHermanoSiguiente(actual)) != null) {
17                actual = hermano;
18                bajan = true;
19            } else { // Regresa al padre
20                actual = actual.getPadre();
21                bajan = false; // Avisar que vamos regresando
22            }
23        }
24        return l;
25    }
26 }
```

# Temas

## 2 Recorridos

- Tipos de recorridos
- Recorrido postorden
- **Recorrido preorden**
- Recorrido inorden
- Recorrido en amplitud

# Recorrido preorden

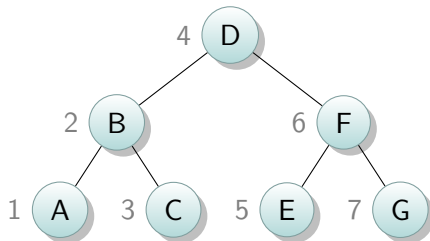


# Temas

## 2 Recorridos

- Tipos de recorridos
- Recorrido postorden
- Recorrido preorden
- Recorrido inorden
- Recorrido en amplitud

# Recorrido inorden

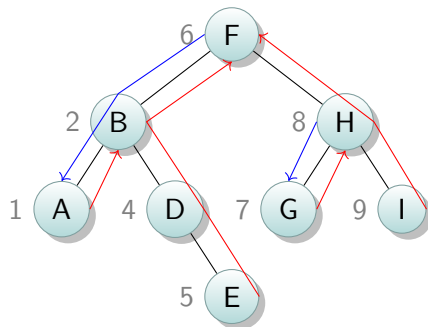




# Recorrido inorden con código recursivo

```
1 import java.util.function.Consumer;
2 public class Nodo<E> {
3     private E dato;
4     private Nodo<E> hijoI, hijoD;
5
6     public void visitaInorden(Consumer<E> f) {
7         if(hijoI != null) hijoI.visitaInorden(f);
8         f.accept(dato);
9         if(hijoD != null) hijoD.visitaInorden(f);
10    }
11 }
```

# Recorrido inorden con código iterativo



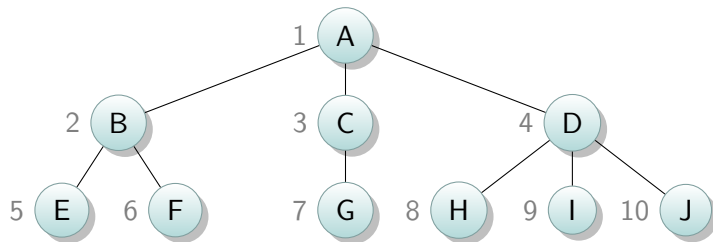
# Recorrido inorden con código iterativo

```
1 public class Nodo<E> {
2     private E dato;
3     private Nodo<E> hijoI, hijoD;
4     private Nodo<E> padre;
5
6     public List<E> recorridoInorden() {
7         List<E> l = new ListaDoblementeLigada<>();
8         if(raíz == null) return l;
9         Nodo<E> actual = raíz;
10        // Llegar al fondo de la rama
11        while(actual.getHijoI() != null) {
12            actual = actual.getHijoI();
13        }
14        while(actual != null) {
15            l.add(actual.getElemento()); // Visita
16            if(actual.getHijoD() != null) {
17                actual = actual.getHijoD();
18                while(actual.getHijoI() != null) {
19                    actual = actual.getHijoI();
20                }
21            } else {
22                // Subir
23                NodoBinario<C> anterior;
24                do {
25                    anterior = actual;
26                    actual = actual.getPadre();
27                } while(actual != null && actual.getHijoD() == anterior);
28            }
29        }
30    }
31 }
```

# Temas

- 2 Recorridos
  - Tipos de recorridos
  - Recorrido postorden
  - Recorrido preorden
  - Recorrido inorden
  - Recorrido en amplitud

# Recorrido en amplitud






- Requiere usar una estructura auxiliar explícitamente: una **cola**.

# Recorrido en amplitud

## Código 7: Amplitud

```
1  import java.util.function.Consumer;
2
3  public class Nodo<E> {
4      private E dato;
5      private List< Nodo<E> > hijos;
6
7      public Nodo(E dato) {
8          this.dato = dato;
9          this.hijos = new LinkedList<>();
10     }
11
12     public void visitaAmplitud(Consumer<E> f) {
13         Cola<E> cola = new Cola<>();
14         cola.queue(this);
15         Nodo<E> temp;
16         while(!cola.isEmpty()) {
17             temp = cola.dequeue();
18             f.accept(temp.getDato());
19             // Forma a los hijos
20             for(Nodo<E> h : hijos) { cola.queue(h); }
21         }
22     }
23 }
```

# Bibliografía I

-  Cormen, Thomas H. y col. (2009). *Introduction to Algorithms*. 3rd. The MIT Press.
-  Preiss, Bruno (1999). *Data Structures and Algorithms with Object-Oriented Design Patterns in Java*. John Wiley & Sons.
-  Vargas Villazón, América, Jorge Lozano Moreno y Guillermo Levine Gutiérrez, eds. (1998). *Estructuras de datos y Algoritmos*. John Wiley & Sons, 438 pp.

# Licencia

Creative Commons  
Atribución-No Comercial-Compartir Igual

