

Árboles

Definiciones

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

15 de octubre de 2023



- 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:

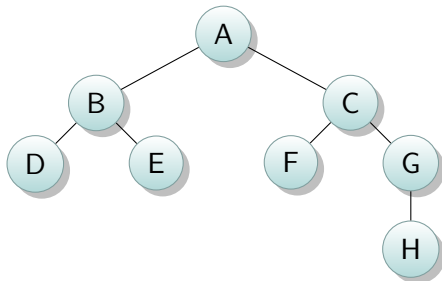
- ❶ Se designa a un nodo r , como la raíz del árbol.
- ❷ Los nodos restantes se dividen en $m \geq 0$ conjuntos ajenos (subconjuntos) T_1, T_2, \dots, T_n , los cuales son a su vez un árbol

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

Ejemplos

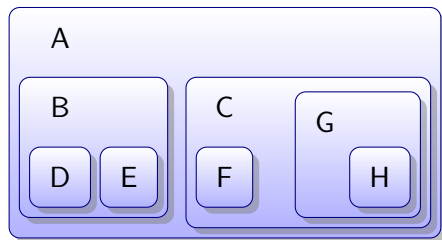
- 1 elemento: $T_a = \{A\}$
- 2 elementos: $T_b = \{B, \{C\}\}$
- varios elementos: $T = \{A, \{B, \{D\}, \{E\}\}, \{C, \{F\}, \{G, \{H\}\}\}$

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 C#

Código: Nodo para árbol

```
1 public class Nodo<T>
2 {
3     public T Dato { get; private set; }
4     private List<Nodo<T>>? _hijos;
5
6     public Nodo(T dato)
7     {
8         Dato = dato;
9     }
10
11     public void AgregaHijo(Nodo<T> hijoNuevo)
12     {
13         if(_hijos == null)
14         {
15             _hijos = new List<Nodo<T>>();
16         }
17         _hijos.Add(hijoNuevo);
18     }
19 }
```

Código: Árbol

```
1 public class Árbol<T>
2 {
3     private Nodo<T>? _raíz;
4
5     public Nodo<T>? Raíz
6     {
7         get => _raíz;
8         private set => _raíz = value;
9     }
10 }
```

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:

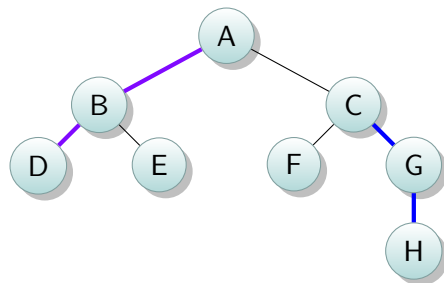
- Dado un árbol T que contiene al conjunto de nodos R , un *camino* en T se define como una secuencia de nodos distinta del vacío:

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

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 $< i + 1 >$ -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.

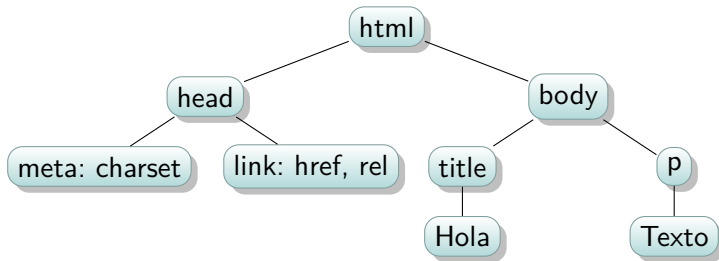
Altura de un árbol A : la altura de su nodo raíz.

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

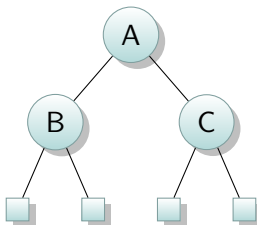
- Definiciones
- Características
- Árboles n-arios
 - Árboles n-arios en arreglos
 - Árboles binarios

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.



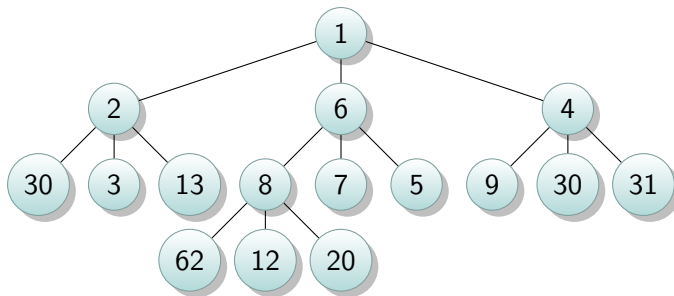
Árboles n-arios

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: $n = 3$

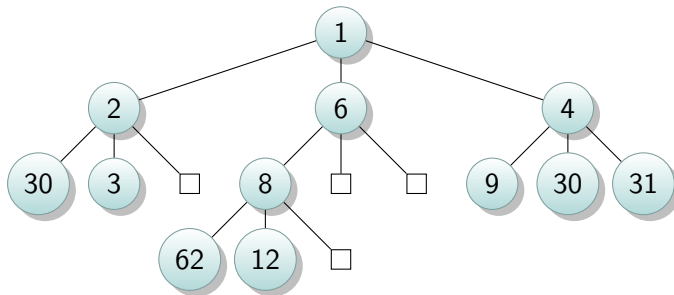


(Árboles n -arios, definición casi equivalente)

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 I

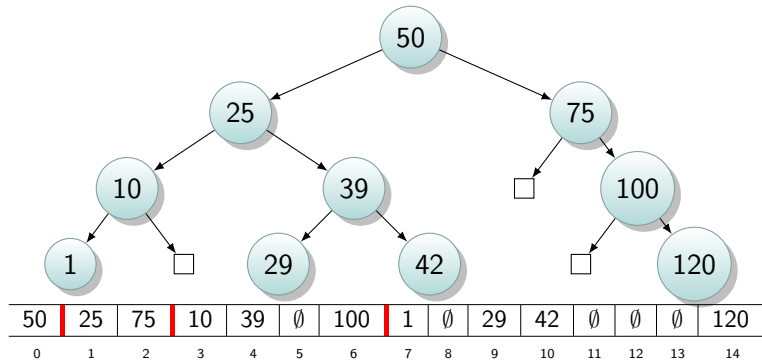
Código: Árbol n-ario

```
1  using System.Diagnostics.CodeAnalysis;
2
3  public class ÁrbolNArio<T>
4  {
5      class Nodo
6      {
7          private ÁrbolNArio<T> _árbol;
8          private T _dato;
9          public T Dato
10         {
11             get => _dato;
12
13             [MemberNotNull(nameof(_dato))]
14             set
15             {
16                 if (value == null)
17                 {
18                     throw new NullReferenceException("Dato_nulo.");
19                 }
20                 _dato = value;
21             }
22         }
23     }
24 }
```

Código Java II

```
23     private Nodo<T>[] _hijos;
24
25     public Nodo(ÁrbolNArio<T> árbol, T dato)
26     {
27         _árbol = árbol;
28         Dato = dato;
29         _hijos = new Nodo<T>[_árbol._grado];
30     }
31 }
32
33 private int _grado;
34 private Nodo<T>? _raíz;
35
36 public ÁrbolNArio(int grado)
37 {
38     if (grado <= 0) throw new ArgumentException("Se necesita al menos un hijo");
39     _grado = grado;
40 }
41 }
```

Ejemplo: árbol binario


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

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:

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

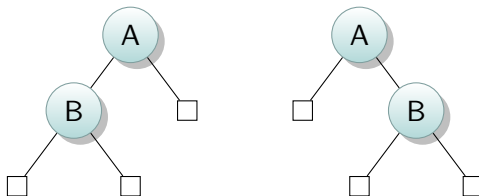
Árboles binarios

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: Árbol binario

```
1 public class ÁrbolBinario<T>
2 {
3     public class Nodo
4     {
5         private T _dato;
6         private Nodo? _hijoI;
7         private Nodo? _hijoD;
8
9         public Nodo(T dato, Nodo? hi, Nodo? hd)
10        {
11            if (dato == null) throw new NullPointerException("Dato_nulo.");
12            _dato = dato;
13            _hijoI = hi;
14            _hijoD = hd;
15        }
16    }
17
18    private Nodo? _raíz;
19 }
```

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.

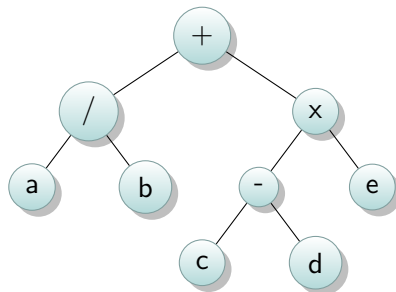
Recorridos

Resumiendo:

- ➊ En profundidad (por subárboles)
 - ➊ Preorden: $r \rightarrow T_L \rightarrow T_R$.
 - ➋ Inorden: $T_L \rightarrow r \rightarrow T_R$.
 - ➌ Postorden: $T_L \rightarrow T_R \rightarrow r$.
- ➋ 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.

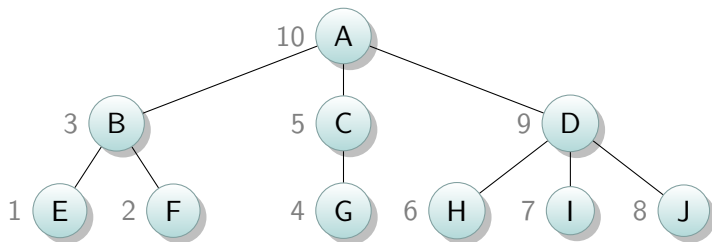
Postorden: $ab/cd - e \times +$ *Postfija

Temas

2 Recorridos

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

Recorrido postorden



Recorrido postorden con código recursivo

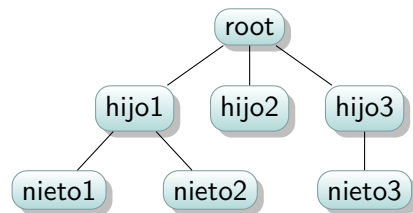
Código: Postorden

```
1 public class Nodo<T>
2 {
3     private T _dato;
4     public List<Nodo<T>> Hijos { get; set; }
5
6     public Nodo(T dato)
7     {
8         _dato = dato;
9         Hijos = new List<Nodo<T>>();
10    }
11
12    public void VisitaPostorden(Action<T> f)
13    {
14        // Visita a los hijos
15        foreach(Nodo<T> h in Hijos)
16        {
17            h.VisitaPostorden(f);
18        }
19        // Visita este nodo
20        f(_dato);
21    }
22 }
```

Ejemplo de uso

Código: Uso postorden

```
1 public class UsoPostorden
2 {
3     public static void Main()
4     {
5         Nodo<string> raíz = new Nodo<string>("root");
6         // hijos
7         List<Nodo<string>> hijos1 = raíz.Hijos;
8         hijos1.Add(new Nodo<string>("hijo1"));
9         hijos1.Add(new Nodo<string>("hijo2"));
10        hijos1.Add(new Nodo<string>("hijo3"));
11        // nietos
12        Nodo<String> h1 = raíz.Hijos[0];
13        h1.Hijos.Add(new Nodo<string>("nieto1"));
14        h1.Hijos.Add(new Nodo<string>("nieto2"));
15
16        Nodo<String> h3 = raíz.Hijos[2];
17        h3.Hijos.Add(new Nodo<string>("nieto3"));
18
19        raíz.VisitaPostorden(o => Console.WriteLine(o));
20        //h1.VisitaPostorden(o => Console.WriteLine(o));
21    }
22 }
```



```
1 nieto1
2 nieto2
3 hijo1
4 hijo2
5 nieto3
6 hijo3
7 root
```

Recorrido postorden con código iterativo

```

1  public class Árbol<T>
2  {
3      private Nodo _raíz;
4      public List<T> RecorridoPostorden()
5      {
6          List<T> l = new List<T>();
7          bool bajan = true;
8          Nodo<T> actual = _raíz, hermano;
9          while(actual != null)
10         {
11             // Llegar al fondo de la rama
12             if (bajan) { while(!actual.EsHoja) { actual = actual.Hijos[0]; } }
13             l.Add(actual.Dato); // Visita
14             if(actual.Padre != null && // Hermano siguiente
15                (hermano = actual.Padre.ObténHermanoSiguiente(actual)) != null)
16             {
17                 actual = hermano;
18                 bajan = true;
19             }
20             else
21             {
22                 // Regresa al padre
23                 actual = actual.Padre();
24                 bajan = false; // Avisar que vamos regresando
25             }
26         }
27         return l;
28     }
29 }

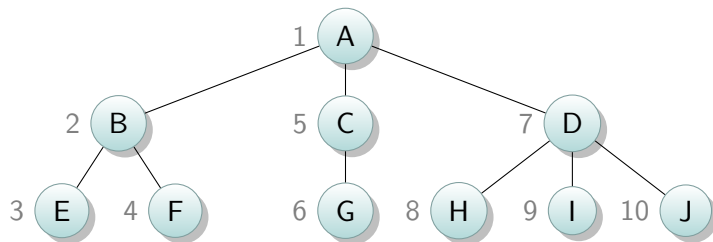
```

Temas

2 Recorridos

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

Recorrido preorden

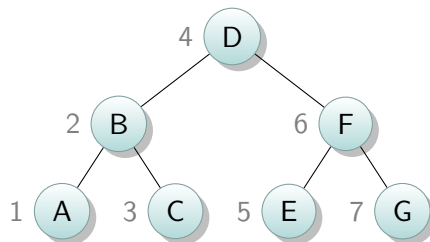


Temas

2 Recorridos

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

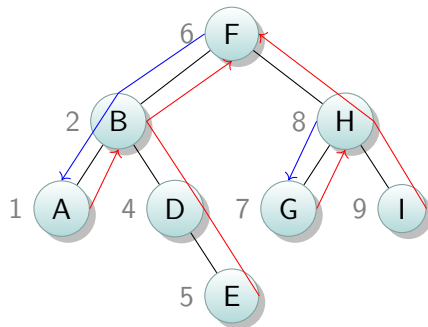
Recorrido inorden



Recorrido inorden con código recursivo

```
1 public class Nodo<T>
2 {
3     private T _dato;
4     private Nodo<T> _hijoI, _hijoD;
5
6     public void VisitaInorden(Action<T> f)
7     {
8         if(_hijoI != null) _hijoI.VisitaInorden(f);
9         f(dato);
10        if(_hijoD != null) _hijoD.VisitaInorden(f);
11    }
12 }
```

Recorrido inorden con código iterativo



Recorrido inorden con código iterativo

```

1 public class Árbol<T>
2 {
3     internal class NodoBinario
4     {
5         public T Dato { get; set; }
6         private NodoBinario HijoI { get; set; }
7         private NodoBinario HijoD { get; set; }
8         private NodoBinario Padre { get; set; }
9     }
10
11     private NodoBinario? _raíz;
12
13     public List<T> RecorridoInorden()
14     {
15         List<T> l = new List<T>();
16         if(_raíz == null) return l;
17         Nodo<T> actual = _raíz;
18         // Llegar al fondo de la rama
19         while(actual.HijoI != null)
20         {
21             actual = actual.HijoI;
22         }

```

```

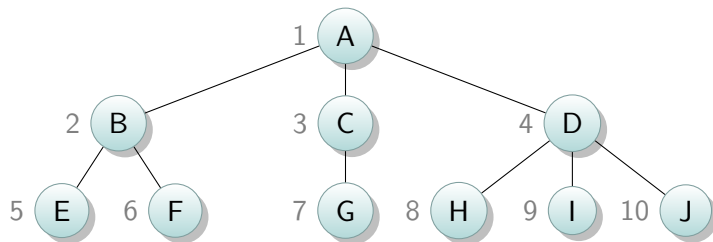
1     while(actual != null)
2     {
3         l.Add(actual.Dato); // Visita
4         if(actual.HijoD != null)
5         {
6             actual = actual.HijoD;
7             while(actual.HijoI != null)
8             {
9                 actual = actual.HijoI;
10            }
11        }
12        else
13        {
14            // Subir
15            NodoBinario anterior;
16            do {
17                anterior = actual;
18                actual = actual.Padre;
19            } while(actual != null && actual.HijoD ==
20                ->anterior);
21        }
22    }
23 }

```

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: Amplitud

```
1  public class Nodo<T>
2  {
3      private T _dato;
4      private List<Nodo<T>> _hijos;
5      public Nodo(T dato)
6      {
7          _dato = dato;
8          _hijos = new List<T>();
9      }
10
11     public void VisitaAmplitud(Action<T> f)
12     {
13         Cola<Nodo<T>> cola = new Cola<Nodo<T>>();
14         cola.Forma(this);
15         Nodo<T> temp;
16         while(!cola.EstáVacía)
17         {
18             temp = cola.Atiende();
19             f(temp.Dato);
20             foreach(Nodo<T> h in _hijos) { cola.Forma(h); } // Forma a los hijos
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

