Belén Sáenz Rubio
J. Ángel Velázquez Iturbide

Revisión Bibliográfica de Algoritmos de Programación Dinámica

Número 2012-03

Serie de Informes Técnicos DLSI1-URJC
ISSN 1988-8074
Departamento de Lenguajes y Sistemas Informáticos I
Universidad Rey Juan Carlos

Índice

Revisión Bibliográfica de Algoritmos de Programación Dinámica

Belén Sáenz, J.Ángel Velázquez Iturbide
Departamento de Lenguajes y Sistemas Informáticos I, Universidad Rey Juan Carlos,
C/Tulipán s/n,28936,Móstoles, Madrid
{belen.saenz, angel.velazquez}@urjc.es

Abstract. Existen diferentes técnicas que permiten abordar un mismo problema empleando una secuencia de pasos o algoritmos diferentes, buscando siempre una solución óptima y mejor a las ya existentes. En este caso nos centramos en la técnica de resolución de problemas basados en programación dinámica, que persigue principalmente la optimización en la resolución de los mismos. En el presente documento se lleva a cabo una revisión bibliográfica que recoge y clasifica los diferentes tipos de problemas más representativos que emplean la técnica de programación dinámica para su resolución.

Keywords: problemas, optimización, algoritmos, programación dinámica.

1 Introducción

A la hora de consultar información sobre la resolución de problemas empleando una técnica dada, en nuestro caso programación dinámica, se hace necesario tener que consultar una gran variedad de diferentes fuentes. Para evitar esto, se ha realizado una recopilación de los principales ejercicios que ilustran los problemas más populares empleando la técnica de programación dinámica. Para ello se han consultado más de una veintena de libros de texto reconocidos sobre algoritmos [1-14]. El procedimiento de búsqueda de información en cada libro se ha basado en la consulta del uso de esta técnica (programación dinámica, *dynamic programming*) para la resolución de problemas en el índice del mismo, de no encontrarse información relevante, se ha realizado una búsqueda de los principales problemas que emplean esta técnica o algoritmo clásico:

- Caminos mínimos (algoritmos de Floyd y Warshall)
- Grafo multietapa (*multistage graph*)
- Mochila 0/1
- …

1.1 Programación dinámica

A continuación se define de forma escueta la técnica de resolución de problemas que se emplean en todos los ejercicios que se recopilan en este informe.

La técnica de programación dinámica consiste básicamente en resolver métodos pequeños para después combinarlos resolviendo problemas mayores, siendo una subsecuencia óptima. La subsecuencia de esta técnica consiste en resolver los subproblemas una única vez, almacenando las soluciones en una tabla para su futura utilización, principio conocido como memorización.

Esta técnica tiene mayor aplicación en problemas de optimización, y la solución de estos problemas se basa en el principio de óptimo de Bellman que dice: "En una secuencia de decisiones óptima toda subsecuencia ha de ser también óptima".

1.2 Estructura del informe

En el documento se han recogido los principales tipos de ejercicios existentes en la bibliografía que emplean la técnica de programación dinámica en la resolución de problemas. En primer lugar, se ha realizado una selección de los ejercicios más populares en las diferentes fuentes, dando como resultado una lista de problemas que se puede consultar en apartado 3 y se desarrollan a lo largo del informe.

En el informe se ha dedicado un apartado específico para cada uno de los tipos de ejercicios seleccionados en la lista. En cada uno de estos puntos se aúna toda la información de ese tipo de ejercicio existente en toda la literatura consultada. Inicialmente se muestra una tabla resumen que ofrece información de los diferentes libros de los que se han obtenido los ejercicios clasificados en ese punto. El formato de dicha tabla tiene la siguiente configuración:

Libro	Capítulo/Apartado	Visualización	Implementación	Nomenclatura

A modo de resumen, se ha dedicado un último punto a una tabla en la que se recogen todos los problemas vistos en este informe.

2 Objetivos

La recopilación que se lleva a cabo en el este documento tiene como objeto principal la clasificación de diferentes problemas que emplean la técnica de programación dinámica en su resolución. De este modo se tendrá en un único documento toda la información que sirve como referencia a la hora de buscar o consultar los problemas resueltos con esta técnica.

Otros propósitos a alcanzar en este trabajo son:

- Recopilar las ilustraciones que permiten complementar las explicaciones que se llevan a cabo en los diferentes problemas planteados.
- Reunir el código o pseudocódigo que explique los diferentes pasos a seguir para la resolución de un problema dado.

3 Problemas

En este apartado se han clasificado los ejercicios más relevantes en los que se aplica la técnica de programación dinámica recogidos de diferentes fuentes. Se ha realizado una clasificación según el tipo de problema que resuelven quedando la siguiente lista ordenada alfabéticamente que se irá desarrollando a lo largo del informe:

1. Cadenas de caracteres.
2. Caminos mínimos.
3. Coeficientes binomiales.
4. Árboles.
5. Grafos.
6. Monedas.
7. Multiplicación de matrices.
8. Planificación.
9. Sucesión de Fibonacci.
10. Otros problemas.

3.1 Cadenas de Caracteres

Existen diferentes problemas planteados que emplean cadenas de caracteres y son resolubles mediante la técnica de programación dinámica, pero sin duda el ejercicio más extendido es el problema de la subsecuencia común más larga (SCML).

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 2 (pág. 205)	Tabla con resultado de ejemplo (Fig. 7.1 pág. 207)	Pseudocódigo (pág. 207)	*The longest common subsequence problem*
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to Algorithms***	Capítulo 15 Apartado 4 (pág. 390)	Tabla con resultado de ejemplo (Fig. 15.8 pág. 395)	Pseudocódigo (págs. 394/5)	*Longest common subsequence*
P3	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 2 (pág. 263)	Método (Fig. 7.13 pág. 265) Tabla con resultado (Fig. 7.2 pág. 66)	NO	*El problema de la subsecuencia común más larga*
P4	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.3 (pág. 503)	Dos casos del algoritmo (Fig. 12.10 pág. 504) Tabla de construcción (Fig. 12.11 pág. 506)	Pseudocódigo (pág. 505)	*The longest common subsequence*
P5	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 5 (pág. 471)	NO	Pseudocódigo (pág. 474)	*Separating sequences of words into lines*
P6	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 13 (pág. 432)	Esquema de tabla (Fig. 13.9 pág. 433)	Pseudocódigo (pág. 434)	*Transformar A en B*
P7	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 2 (pág. 266)	Tablas con resultados (Tablas 7.3 / 4 pág. 268)	NO	*El problema de alineación de dos secuencias*

P8	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 3 (pág. 289)	Matriz editando cadenas. (Fig. 8.4 pág. 284) Matriz padre de editando cadenas.(Fig. 8.5 pág. 285)	Código C - editando cadena, recursivo (págs. 281/2) Código C - editando cadena, pd (pág. 283) Código C - reconstruyendo cadena pd (págs. 285/6) Código C - variedades-tabla de inicialización pd (pág. 286) Código C - variedades-coste de penalización pd (pág. 287) Código C - variedades-Identificación de la celda objetivo pd (pág. 287) Código C - variedades-acciones de rastreo pd (pág. 287) Código C - variedades-Correspondencia de substring pd (pág. 288) Código C - variedades-Subsecuencia común más larga pd (pág. 288) Código C - Comparación de string, pd (http://www.cs.sunysb.edu/~skiena/392/programs/editdistance.c) Código Java (http://www.cs.sunysb.edu/~skiena/392/javaprograms/)	*Approximate String Matching*
P9	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 3 (pág. 289)	Tabla con valores de ejemplo (pág. 291)	NO	*Longest increasing sequence*
P10	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 6 (pág. 298)	Gramática libre de contexto y árbol. (Fig. 8.9 pág. 298)	NO	*Parsing Context-Free Grammars*
P11	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 3 (pág. 270)	Estructura secundaria (Fig. 7.14 pág. 271) Ilustraciones de casos (Fig. 7.15-17 pág. 273/4) Tabla con resultado ejemplo (Tabla 7.5 pág. 275)	Pseudocódigo (págs. 275-6)	*Problema de apareamiento del máximo par de bases de ARN*

3.1.1 Subsecuencia Común Más Larga [P1-P4]

Problema:

En el problema de la subsecuencia común más larga (SCML) se consideran dos secuencias: $X = \langle x_1,\ldots, x_m \rangle$ e $Y = \langle y_1,\ldots, y_n \rangle$. Se dice que $Z = \langle z_1,\ldots, z_k \rangle$ es una subsecuencia de X si existe una secuencia de índices de X, $i_1,\ldots, i_k$ tal que $x_{i\ j} = z_j$ para $j = 1, 2,\ldots, k$. Z será común a X e Y si es subsecuencia de ambas y será una SCML si no existe otra de longitud mayor.

Visualizaciones:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3	3	3
4	0	0	1	1	2	2	2	3	4	4	4	4	4
5	0	1	1	2	2	2	3	3	4	4	4	4	5
6	0	1	2	2	2	2	3	4	4	4	5	5	5
7	0	1	2	2	3	3	3	4	4	5	5	5	5
8	0	1	2	3	3	3	4	4	4	5	5	5	6
9	0	1	2	3	3	3	4	5	5	5	6	6	6
10	0	1	2	3	4	4	4	5	5	6	6	6	6

Ilustración 1 - P1 Tabla con resultado de ejemplo aplicado a A="xyxxzxyzxy" B="zxzyyzxxyxxz".

Algorithm 7.1 LCS
Input: Two strings A and B of lengths n and m, respectively, over an alphabet Σ.
Output: The length of the longest common subsequence of A and B.

```
1.  for i ← 0 to n
2.      L[i,0] ← 0
3.  end for
4.  for j ← 0 to m
5.      L[0,j] ← 0
6.  end for
7.  for i ← 1 to n
8.      for j ← 1 to m
9.          if a_i = b_j then L[i,j] ← L[i-1,j-1] + 1
10.         else L[i,j] ← max{L[i,j-1], L[i-1,j]}
11.         end if
12.     end for
13. end for
14. return L[n,m]
```

Ilustración 2 - P1 Pseudocódigo

---

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	←1	↖ 1
2	B	0	↖ 1	←1	←1	↑ 1	↖ 2	←2
3	C	0	↑ 1	↑ 1	↖ 2	←2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	←3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Ilustración 3 – P2 Tabla con resultado de ejemplo aplicado a A="ABCBDAB" B="BDCABA".

```
LCS-LENGTH(X, Y)
1   m = X.length
2   n = Y.length
3   let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4   for i = 1 to m
5       c[i, 0] = 0
6   for j = 0 to n
7       c[0, j] = 0
8   for i = 1 to m
9       for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```

Ilustración 4 - P2 Pseudocódigo

---

b_j \ a_i			a	b	c	d
		0	1	2	3	4
	0	0	0	0	0	0
c	1	0	0	0	1	1
b	2	0	0	1	1	1
d	3	0	0	1	1	2

$L_{i,j}$

Ilustración 5 – P3 Tabla con resultado de ejemplo aplicado a A="abcd" B="cba".

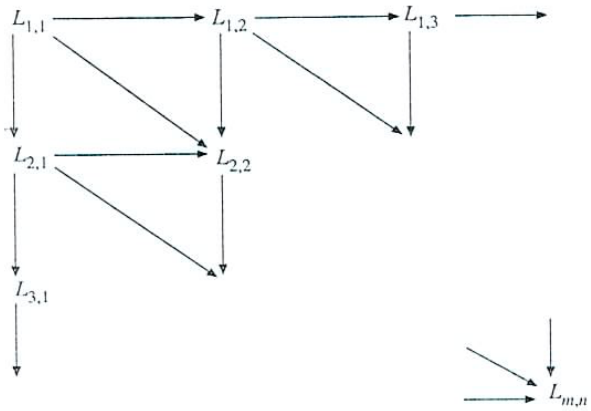


Ilustración 6 - P3 Método para resolver el problema

$$L[i, j] = L[i-1, j-1] + 1 \quad \text{if } x_i = y_j.$$

$$L[i, j] = \max\{L[i-1, j], L[i, j-1]\} \quad \text{if } x_i \neq y_j.$$

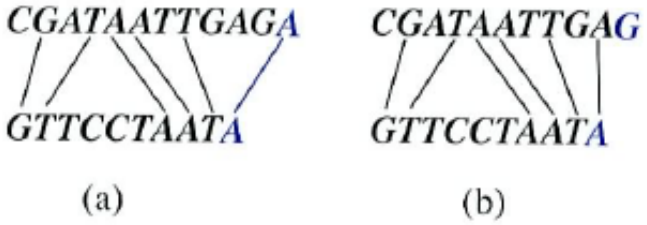


Ilustración 7 - P4 Dos casos del algoritmo de la subsecuencia común más larga; a) $x_i = y_j$ b) $x_i \neq y_j$

Algorithm LCS(X,Y):

Input: Strings X and Y with n and m elements, respectively

Output: For $i = 0,\ldots,n-1$, $j = 0,\ldots,m-1$, the length $L[i,j]$ of a longest string that is a subsequence of both the string $X[0..i] = x_0x_1x_2\ldots x_i$ and the string $Y[0..j] = y_0y_1y_2\ldots y_j$

```
for i ← −1 to n − 1 do
  L[i, −1] ← 0
for j ← 0 to n − 1 do
  L[−1, j] ← 0
for i ← 0 to n − 1 do
  for j ← 0 to m − 1 do
    if x_i = y_j then
      L[i, j] ← L[i − 1, j − 1] + 1
    else
      L[i, j] ← max{L[i − 1, j], L[i, j − 1]}
return a matrix representing L
```

Ilustración 8 - P4 Pseudocódigo

L	-1	0	1	2	3	4	5	6	7	8	9	10	11
-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	**1**	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	**2**	2	2	2	2	2	2	2	2
2	0	0	1	1	2	2	2	3	3	3	3	3	3
3	0	1	1	1	2	2	2	3	3	3	3	3	3
4	0	1	1	1	2	2	2	3	3	3	3	3	3
5	0	1	1	1	2	2	2	3	4	4	4	4	4
6	0	1	1	2	2	**3**	3	3	4	4	5	5	5
7	0	1	1	2	2	3	**4**	4	4	4	5	5	6
8	0	1	1	2	3	3	4	5	**5**	5	5	5	6
9	0	1	1	2	3	4	4	5	5	5	6	6	**6**

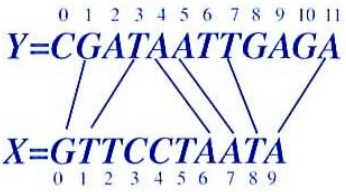


Ilustración 9 - P4 Ilustración del algoritmo para construir la subsecuencia común más larga del array L

3.1.2 Separating Sequences of Words into Lines [P5]

***Problema*:**
The problem is separating a sequence of words into a series of lines that comprise a paragraph. The objective is to avoid a lot of extra spaces on any line. This is an important problem in computerized typesetting. Because extra spaces on the last line of the paragraph are not objectionable, the paragraph is a natural unit to optimize. Of course, the order of the words must be maintained as they are placed in lines. The input to the line-breaking problem is a sequence of n word lengths, $w_1,\ldots, w_n$, representing the lengths of words that make up a paragraph, and a line width W.

The basic constraint on word placement is that, if words i through j are placed on a single line, then $w_i+\ldots+w_j \leq W$. in this case the number of extra spaces is

$X = W- (w_i+\ldots+w_j)$.

The penalty for extra spaces is assumed to be some function of X. For our discussion, the line penalty is specified as X^3.

Example:
We take to be the whole paragraph:

i	1	2	3	4	5	6	7	8	9	10	11
	Those	who	cannot	remember	the	past	are	condemned	to	repeat	it.
w_i	6	4	7	9	4	5	4	10	3	7	4

Suppose $W = 17$. The greedy strategy groups words into lines as follows:

words	(1, 2, 3)	(4, 5)	(6, 7)	(8, 9)	(10, 11)
X	0	4	8	4	0
penalty	0	64	512	64	0

Visualizaciones:

```
lineBreak(w, W, i, n, L)  // OUTLINE
    if (w_i + ··· + w_n ≤ W)
        Put all words on line L and set penalty to 0.
    else
        Set penalty to the minimum of kPenalty over all k > 0 such that
        w_i + ··· + w_{i+k-1} ≤ W, where X = W − (w_i + ··· + w_{i+k-1}), and
        kPenalty = lineCost(X) + lineBreak(w, W, i+k,  n,  L+1);
        Let k_min be the k that produced the minimum penalty.
        Put words i through i + k_min − 1 on line L.
    return penalty;
```

Ilustración 10 - P5 Pseudocódigo

3.1.3 Transformar A en B [P6]

Problema:
Sean $A=a_1a_2...a_n$ y $B=b_1b_2...b_m$ dos cadenas sobre un alfabeto finito de caracteres desea transformar *A* en *B* utilizando una serie de cambios de caracteres de las tres siguientes clases:

Insertar (*c*, *k*) Insertar el carácter *c* en la posición *k* de la cadena
Borrar (*k*) Borra el carácter en la posición *k* de la cadena
Sustituir (*c*, *k*) Sustituye el carácter en la posición *k* de la cadena por el carácter *c*

Por ejemplo, la cadena *abbc* se transforma en la cadena *babb* y sustituir los tres siguientes cambios: borrar la *a* (quedando *bbc*), insertar una *a* entre las *b* (*babc*) y sustituir la *c* por una *b*.

También se puede conseguir mediante sólo dos cambios: insertar *b* al principio (*babbc*) y borrar la *c*.

Desarrollar un algoritmo para saber cuál es el número mínimo de cambios necesarios para transformar *A* en *B* y cuáles son tales cambios.
Por ejemplo la trasformación del enunciado es:

$$abbc \xrightarrow{\text{borrar } (a)} bbc \xrightarrow{\text{insertar } (a)} babc \xrightarrow{\text{sustituir } (b,\ c)} babb$$

Es equivalente entre otras a:

$$abbc \xrightarrow{\text{borrar } (a)} bbc \xrightarrow{\text{sustituir } (b,c)} bbb \xrightarrow{\text{insertar } (a)} babb$$

Visualizaciones:

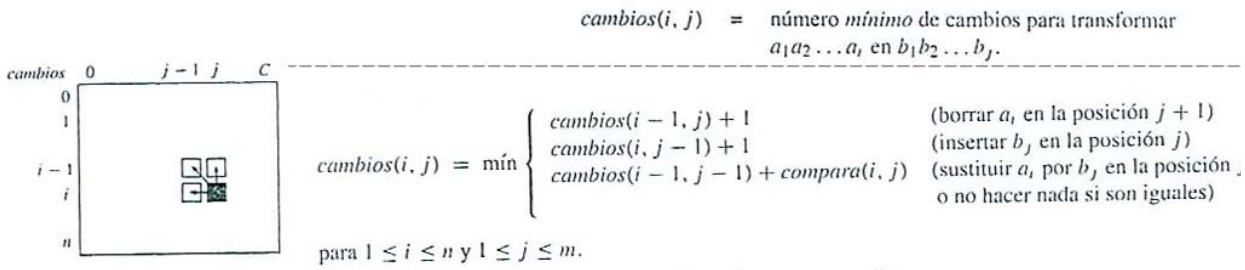


Ilustración 11 - P6 Esquema de tabla y función recursiva para el problema de la transformación de cadenas

```
tipos
    operación  =  { borrar, insertar, sustituir }
    cambio     =  reg
                      op : operación
                      pos : nat
                      car : car
                  freg
ftipos

fun transformar(A[1..n], B[1..m] de car) dev (núm-cambios : nat, qué-cambios[1..n + m] de cambio)
var cambios[0..n, 0..m] de nat
    { inicialización }
    para i = 0 hasta n hacer cambios[i, 0] := i fpara
    para j = 1 hasta m hacer cambios[0, j] := j fpara
    { rellenamos la matriz }
    para i = 1 hasta n hacer
        para j = 1 hasta m hacer
            si A[i] = B[j] entonces compara := 0 si no compara := 1 fsi
            cambios[i, j] := mín(mín(cambios[i − 1, j] + 1, cambios[i, j − 1] + 1),
                                 cambios[i − 1, j − 1] + compara)
        fpara
    fpara
    núm-cambios := cambios[n, m]
    { cálculo de las transformaciones }
    k := núm-cambios ; i := n ; j := m
    mientras k > 0 hacer
        casos
            cambios[i, j] = cambios[i − 1, j] + 1 →
                qué-cambios[k].op := borrar
                qué-cambios[k].pos := j + 1
                k := k − 1 ; i := i − 1
          □ cambios[i, j] = cambios[i, j − 1] + 1 →
                qué-cambios[k].op := insertar
                qué-cambios[k].pos := j ; qué-cambios[k].car := B[j]
                k := k − 1 ; j := j − 1
          □ cambios[i, j] = cambios[i − 1, j − 1] + 1 →
                qué-cambios[k].op := sustituir
                qué-cambios[k].pos := j ; qué-cambios[k].car := B[j]
                k := k − 1 ; j := j − 1 ; i := i − 1
          □ cambios[i, j] = cambios[i − 1, j − 1] →
                j := j − 1 ; i := i − 1
        fcasos
    fmientras
ffun
```

Ilustración 12 P6-Pseudocódigo

3.1.4 Alineación de 2 Secuencias [P7]

Problema:
Sean $A=a_1a_2...a_m$ y $B=b_1b_2...b_n$ dos secuencias sobre un conjunto alfabeto $\sum$. Una alineación de secuencias de A y B es una matriz $2\times k$ de M ($k \geq m, n$) caracteres sobre $\sum \cup \{-\}$ tal que ninguna columna de M consta completamente de guiones, y los resultados que se obtienen al eliminar todos los guiones en el primer renglón y en el segundo renglón de M son iguales a A y B, respectivamente. Por ejemplo, si $A=a\ b\ c$ y $B=c\ b\ d$, una alineación posible de estas sería

a b c - d
- - c b d

Otra alineación posible de las dos secuencias es:

a b c d
c b - d

Visualizaciones:

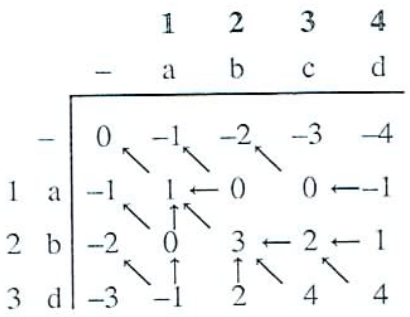
$$A_{0,0} = 0$$
$$A_{i,0} = i \cdot f(a_i, -)$$
$$A_{0,j} = j \cdot f(-, b_j)$$

$$A_{i,j} = \text{máx}\begin{cases} A_{i-1,j} + f(a_i, -) \\ A_{i-1,j-1} + f(a_i, b_j) \\ A_{i,j-1} + f(-, b_j) \end{cases}$$

Ilustración 13 P7- Fórmula de recurrencia

a_i / b_j			*a*	*b*	*d*	*a*	*d*
		0	**1**	**2**	**3**	**4**	**5**
	0	0	−1	−2	−3	−4	−5
b	1	−1	1	1	0	−1	−2
a	2	−2	1	2	2	2	1
c	3	−3	0	2	3	3	3
d	4	−4	−1	1	4	4	5

Ilustración 14 P7-Tabla ejemplo con valores de las Aij para A=a b d a d y B=b a c d



			1	2	3	4
		–	a	b	c	d
	–	0	−1	−2	−3	−4
1	a	−1	1	0	0	−1
2	b	−2	0	3	2	1
3	d	−3	−1	2	4	4

Ilustración 15 P7-Tabla ejemplo con valores de las A_{ij} para A=a b c d y B=a b d

3.1.5 Approximate String Matching [P8]

Problema:
An important task in text processing is string matching, finding all the occurrences of a word in the text. Unfortunately, many words in documents are mispelled (sic). How can we search for the string closest to a given pattern in order to account for spelling errors?

To be more precise, let *P* be a pattern string and *T* a text string over the same alphabet. The *edit distance* between *P* and *T* is the smallest number of changes sufficient to transform a substring of *T* into *P*, where the changes may be:

1. *Substitution* - two corresponding characters may differ: $KAT \rightarrow CAT$.
2. *Insertion* - we may add a character to *T* that is in *P*: $CT \rightarrow CAT$.
3. *Deletion* - we may delete from *T* a character that is not in *P*: $CAAT \rightarrow CAT$.

For example, *P* = *a b c d e f g h i j k l* can be matched to *T* = *b c d e f f g h i x k l* using exactly three changes, one of each of the above types.

Visualizaciones:

	T		y	o	u	-	s	h	o	u	l	d	-	n	o	t
P	pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
:		**0**	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t:	1	**1**	1	2	3	4	5	6	7	8	9	10	11	12	13	13
h:	2	2	**2**	2	3	4	5	5	6	**7**	8	9	10	11	12	13
o:	3	3	3	**2**	3	4	5	6	5	6	7	8	9	10	11	12
u:	4	4	4	3	**2**	3	4	5	6	5	6	**7**	8	9	10	11
-:	5	5	5	4	3	**2**	3	4	5	6	6	**7**	7	8	9	10
s:	6	6	6	5	4	3	**2**	3	4	5	6	7	8	8	9	10
h:	7	7	7	6	5	4	3	**2**	**3**	4	5	6	**7**	8	9	10
a:	8	8	8	**7**	6	5	4	3	3	**4**	5	6	**7**	8	9	10
l:	9	9	9	8	7	6	5	4	4	4	**4**	5	6	7	8	9
t:	10	10	10	9	8	7	6	5	5	5	5	**5**	6	7	8	8
-:	11	11	11	10	9	8	**7**	6	6	6	6	6	**5**	6	7	8
n:	12	12	12	11	10	9	8	7	7	7	**7**	7	6	**5**	6	7
o:	13	13	13	12	11	10	9	8	7	8	8	8	7	6	**5**	6
t:	14	14	14	13	12	11	10	9	8	8	9	9	8	7	6	**5**

Ilustración 16 P8- Example of a dynamic programming matrix for editing distance computational with the optimal alignment path highlighted in bold

---

	T		y	o	u	-	s	h	o	u	l	d	-	n	o	t
P	pos	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	0	**-1**	1	1	1	1	1	1	**1**	1	1	1	1	1	1	1
t:	1	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
h:	2	2	**0**	0	0	0	0	0	**1**	1	1	1	1	1	1	1
o:	3	2	0	**0**	0	0	0	0	0	1	1	1	1	1	0	1
u:	4	2	0	2	**0**	1	1	1	**1**	0	1	1	1	1	1	1
-:	5	2	0	2	2	**0**	1	1	**1**	1	0	0	0	1	1	1
s:	6	2	0	2	2	2	**0**	1	**1**	1	1	0	0	0	0	0
h:	7	2	0	2	2	2	2	**0**	**1**	1	1	1	1	1	0	0
a:	8	2	0	2	2	2	2	2	0	**0**	0	0	0	0	0	0
l:	9	2	0	2	2	2	2	2	0	0	**0**	1	1	1	1	1
t:	10	2	0	2	2	2	2	2	0	0	0	**0**	0	0	0	0
-:	11	2	0	2	2	0	2	2	0	0	0	0	**0**	1	1	1
n:	12	2	0	2	2	2	2	2	0	0	0	0	2	**0**	1	1
o:	13	2	0	0	2	2	2	2	0	0	0	0	2	2	**0**	1
t:	14	2	0	2	2	2	2	2	2	0	0	**0**	2	2	2	**0**

Ilustración 17 P8-Parent matrix for edit distance computation, with the optimal alignment path highlighted in bold

3.1.6 Longest Increasing Sequence [P9]

Problema:

We distinguish an increasing sequence from a run, where the elements must be physical neighbors of each other. The selected elements of both must be sorted in increasing order from left to right. For example, consider the sequence

S = {2, 4, 3, 5, 1, 7, 6, 9, 8}

The longest increasing subsequence of *S* has length 5, including {2, 3, 5, 6, 8}. In fact, there are eight of this length. There are four longest increasing runs of length 2: (2, 4), (3, 5), (1, 7) and (6, 9).

The longest increasing sequence containing the *n*th number will be formed by appending it to the longest increasing sequence to the left of *n* that ends on a number smaller than s_n. The following recurrence computes l_i:

$$l_i = \max_{0<j<i} l_j + 1, \text{ where } (s_j < s_i),$$

$$l_0 = 0$$

Here is the table associated with our previous example:

Sequence s_i	2	4	3	5	1	7	6	9	8
Length l_i	1	2	3	3	1	4	4	5	5
Predecessor p_i	-	1	1	2	-	4	4	6	6

3.1.7 Parsing Context-Free Grammars [P10]

Problema:

Programs often contain trivial syntax errors that prevent them from compiling. Given a context-free grammar *G* and input string *S*, find the smallest number of character substitutions you must make to *S* so that the resulting string is accepted by *G*.

Define *M*'[*i,j,X*] to be an *integer* function that reports the minimum number of changes to substring *S*[*i,j*] so it can be generated by non-terminal *X*. This symbol will be generated by some production $x \rightarrow yz$. Some of the changes to *s* may be to the left of the breaking point and some to the right, but all we care about is minimizing the sum. In other words:

$$M'[i,j,X] = \min_{(X \rightarrow YZ) \in G} \left(\min_{i=k}^{j} M'[i,k,Y] + M'[k+1,j,Z] \right)$$

Visualizaciones:

sentence ::= noun–phrase
verb–phrase
noun–phrase ::= article noun
verb–phrase ::= verb noun–phrase
article ::= *the, a*
noun ::= *cat, milk*
verb ::= *drank*

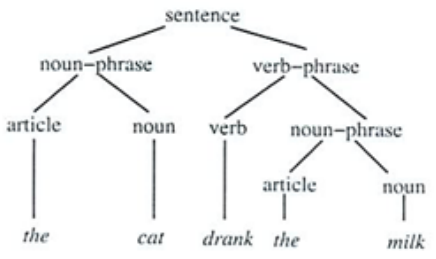


Ilustración 18 P10-A context-free grammar with an associated parse tree

3.1.8 Apareamiento del Máximo Par de Bases de ARN [P11]

Problema:

El ácido ribonucleico (ARN) es una cadena simple de nucleótidos (bases), adenina (*A*), guanina (*G*), citosina (*C*) y uracil (*U*). La secuencia de las bases *A, G, C* y *U* se denomina estructura primaria de un ARN.

En el ARN, *G* y *C* pueden formar un par de bases *G*≡*C* mediante un triple enlace de hidrogeno. *A* y *U* pueden formar un par de bases *A*=*U* mediante un doble enlace de hidrogeno, y *G* y U pueden formar una para de bases *G*-*U* mediante un enlace simple de hidrogeno. Debido a estos enlaces de hidrogeno, la estructura primaria de un ARN puede plegarse sobre sí misma para formar su estructura secundaria. Por ejemplo, suponga que se tiene la siguiente secuencia de ARN:

A-G-G-C-C-U-U-C-C-U

Así, esta estructura puede plegarse sobre sí misma para formar muchas estructuras secundarias posibles, en la naturaleza, no obstante, sólo existe una secuencia secundaria correspondiente a una secuencia de ARN.

El objetivo de la predicción de la estructura secundaria es encontrar una estructura secundaria que tenga la energía libre mínima.

Visualizaciones:

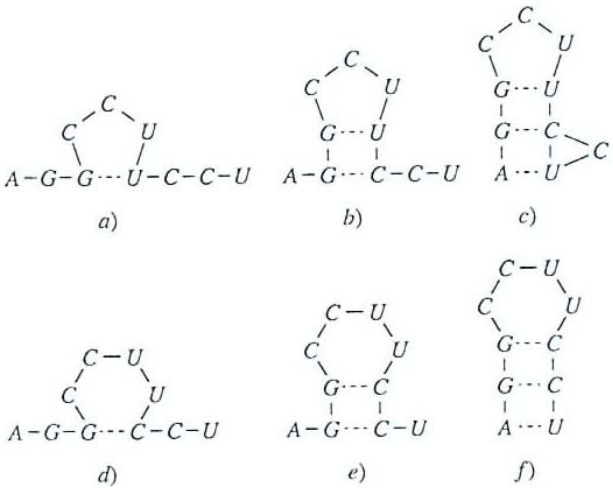


Ilustración 19 P10-Seis estructuras posibles de las secuencias de ARN A-G-G-C-C-U-U-C-C-U (líneas discontinuas representan enlaces de hidrógeno)

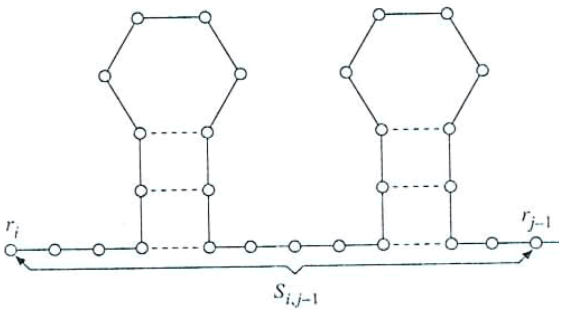


Ilustración 20 P10-Caso1: En la solución óptima r_j no está apareada con ninguna otra base. En este caso, se encuentra una solución óptima para r_j r_{j+1}… r_{j-1} y $M_{ij} = M_{ij}$-1

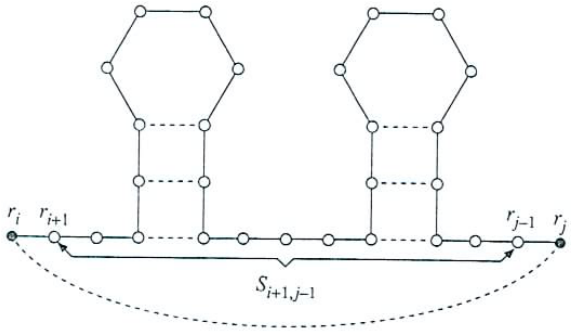


Ilustración 21 P10-Caso2: En la solución óptima r_j está apareada con r_i. En este caso, se encuentra una solución óptima para r_{j+1} r_{j+2}… r_{j-1} y Mij = M_{i+1j-1}

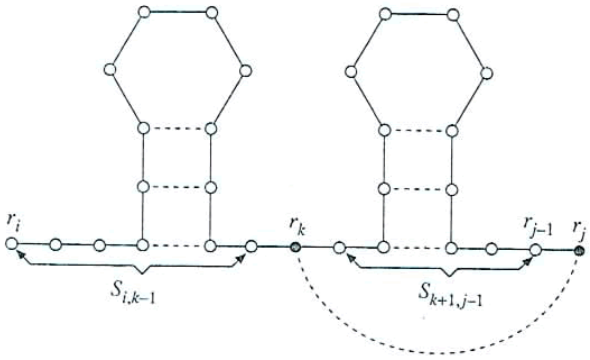


Ilustración 22 P10-Caso3: En la solución óptima r_j está apareada con alguna r_k, donde $i+1 \leq k \leq j-4$. En este caso, se encuentran soluciones óptimas para r_i r_{i+1}… $r_{k+1}r_{k+1}$ ... r_{j-1}, M_{ij} =1+ $M_{i,k-1}$+$M_{k+1,j-1}$

---

i \ j	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	1	2	2	2	3
2	–	0	0	0	0	1	1	2	2	2
3	–	–	0	0	0	0	1	1	1	1
4	–	–	–	0	0	0	0	0	0	0
5	–	–	–	–	0	0	0	0	0	0
6	–	–	–	–	–	0	0	0	0	0
7	–	–	–	–	–	–	0	0	0	0
8	–	–	–	–	–	–	–	0	0	0
9	–	–	–	–	–	–	–	–	0	0
10	–	–	–	–	–	–	–	–	–	0

Ilustración 23 P11 - Cálculo del número máximo de pares de una secuencia de ARN A-G-G-C-C-U-U-C-C-U

3.1.9 Otros Problemas

Problem 407
[Parberry, p. 92, dynamic programming]
A *context-free grammar* in Chomsky Normal Form consists of:

- A set of *nonterminal symbols N*.
- A set of *terminal symbols T*.
- A special nonterminal symbol called the *root*.
- A set of *productions* of the form either $A \rightarrow BC$, or $A \rightarrow a$, where $A, B, C \in N$, $a \in T$.

If $A \in N$, define $L(A)$ as follows:

$L(A) = \{bc \mid b \in L(B), c \in L(C),$ where $A \to BC\} \cup \{a \mid A \to a\}$

The *language* generated by a grammar with root R is defined to be $L(R)$. The *CFL recognition problem* is the following:

For a fixed context-free grammar in Chomsky Normal Form, on input a string of terminals x, determine whether x is in the language generated by the grammar.

Devise an algorithm for the *CFL* recognition problem. Analyze your algorithm.

Problem 408
[Parberry, 1995, p. 92, dynamic programming]
Fill in the tables in the dynamic programming algorithm for the *CFL* recognition problem (see Problem 407) on the following inputs. In each case, the root symbol is *S*.
(a) Grammar: $S \to SS$, $S \to s$. String: *ssssss*.
(b) Grammar: $S \to AR$, $S \to AB$, $A \to a$, $R \to SB$, $B \to b$. String: *aaabbb*.
(c) Grammar: $S \to AX$, $X \to SA$, $A \to a$, $S \to BY$, $Y \to SB$, $B \to b$, $S \to CZ$, $Z \to SC$, $C \to c$. String: *abacbbcaba*.

Problem 410
[Parberry, 1995, p.93, dynamic programming]
A certain string-processing language allows the programmer to break a string into two pieces. Since this involves copying the old string, it costs n units of time to break a string of n characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks are made can affect the total amount of time used.

For example, suppose we wish to break a 20-character string after characters 3, 8, and 10 (numbering the characters in ascending order from the left-hand end, starting from1). If the breaks are made in left-to-right order, then the first breaks costs 20 units of time, the second breaks costs 17 units of time, and the third breaks costs 12 units of time, a total of 49 units of time (see Figure 8.2 in the problem 408). If the breaks are made in right-to-left order, then the first break costs 20 units of time, the second break costs 10 units of time, and the third break costs 8 units of time, a total of 38 units of time.

Devise a dynamic programming algorithm that, when given the numbers of the characters after which to break, determines the cheapest cost of those breaks in time $O(n^3)$.

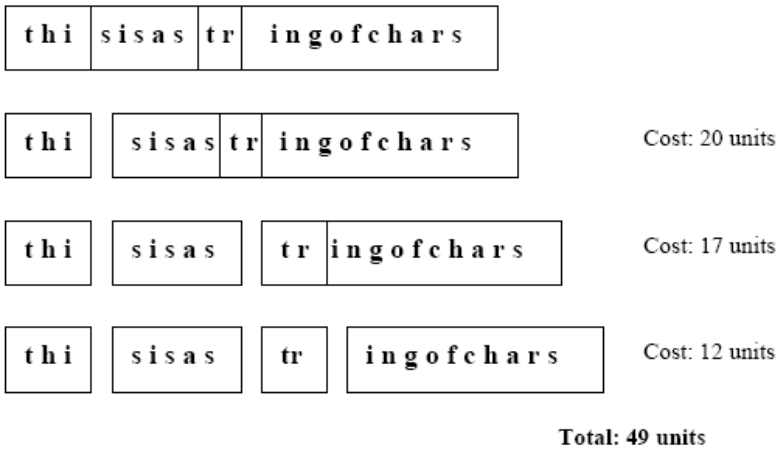


Figure 8.2. The cost of making the breaks in left-to-right order.

Problem 411
[Parberry, 1995, p.93, dynamic programming]
Find counterexamples to the following algorithms for the string-cutting problem introduced in Problem 410. That is, find the length of the string, and several places to cut such that when cuts are made in the order given, the cost in higher than optimal.
(a) Start by cutting the string as close to the middle as possible, and then repeat the same thing recursively in each half.
(b) Start by making (at most) two cuts to separate the smallest substring.

Repeat this until finished. Start by making (at most) two cuts to separate the largest substring. Repeat this until finished.

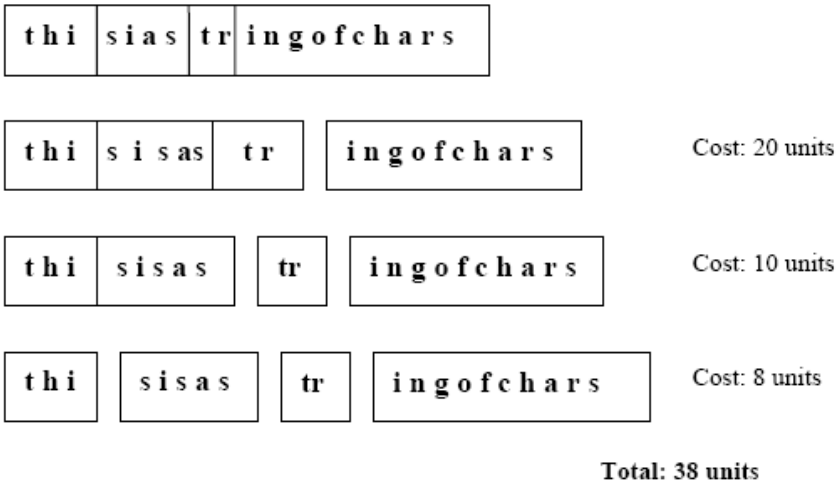


Figure 8.3. The cost of making the breaks in right-to-left order.

Problem 412
[Parberry, 1995, p.94, dynamic programming]

There are two warehouses V and W from which widgets are to be shipped to destinations D_i, $1 \le i \le n$. Let d_i be the demand at D_i, for $1 \le i \le n$, and rV, rW be the number of widgets available at V and W, respectively.

Assume that there are enough widgets available to fill the demand, that is, that

$$rv + rw = \sum_{i=1}^{n} d_i$$

Let v_i be the cost of shipping a widget from warehouse V to destination D_i, and w_i be the cost of shipping a widget from warehouse W to destination D_i, for $1 \le i \le n$. The *warehouse problem* is the problem of finding x_i, $y_i \in$ N for $1 \le i \le n$ such that when x_i widgets are sent from V to D_i and y_i widgets are sent from W to D_i:

The demand at D_i is filled, that is, $x_i + y_i = d_i$,

The inventory at V is sufficient, that is, $\sum_{i=1}^{n} x_i = rv$

The inventory at W is sufficient, that is, $\sum_{i=1}^{n} y_i = rw$

And the total cost of shipping the widgets,

$\sum_{i=1}^{n} v_i x_i + w_i y_i$ is minimized.

Let $g_j(x)$ be the cost incurred when V has an inventory of x widgets, and supplies are sent to destinations Di for all $1 \le i \le j$ in the optimal manner (note that W is not mentioned because knowledge of the inventory for V implies knowledge of the inventory for W, by (8.1).) Write a recurrence relation for $g_j(x)$ in terms of g_{j-1}.

Advance	Move cursor one character to the right
Delete	Delete the character under the cursor, and move the cursor to the next character.
Replace	Replace the character under the cursor with another. The cursor remains stationary.
Insert	Insert a new character before the one under the cursor. The cursor remains stationary.
Kill	Delete all characters from (and including) the one under the cursor to the end of the line. This can only be the last operation.

Ilustración 24 Operations allowed on a smart terminal.

(b) Use this recurrence to devise a dynamic programming algorithm that finds the cost of the cheapest solution to the warehouse problem. Analyze your algorithm.

Problem 413
[Parberry, 1995, p.95, dynamic programming]
Consider the problem of neatly printing a paragraph of text on a printer with fixed-width fonts. The input text is a sequence of n words containing $l_1, l_2, \ldots, l_n$, characters, respectively. Each line on the printer can hold up to M characters. If a printed line contains words i through j, then the number of blanks left at the end of the line (given that there is one blank between each pair of words) is

$$M - j + i - \sum_{k=i}^{j} l_k$$

We wish to minimize the sum, over all lines except the last, of the cubes of the number of blanks at the end of each line (for example, if there are k lines with b_1, b_2, …, b_k blanks at the ends, respectively, then we wish to minimize $b_1^3+b_2^3+ \ldots+b^3_{k-1}$).

Give a dynamic programming algorithm to compute this value. Analyze your algorithm.

Problem 414
[Parberry, 1995, p.95, dynamic programming]
A "smart" terminal changes one string displayed on the screen into another by a series of simple operations. The cursor is initially on the first character of the string. The operations are shown in Table 8.1. For example, one way to transform the string algorithm to the string altruistic is shown in Figure 8.4.

Operation	*String*
Algorithm	
advance	aLgorithm
advance	alGorithm
replace with *t*	alTorithm
advance	altOrithm
delete	altRithm
advance	altrIthm
insert *u*	altruIthm
advance	altruiThm
insert *s*	altruisThm
advance	altruistHm
insert *i*	altruistiHm
insert *c*	altruisticHm
kill	altruistic

Figure 8.4. Transforming algorithm to altruistic using the operations shown in Table 8.1. The cursor position is indicated by a capital letter.

There are many sequences of operations that achieve the same result. Suppose that on a particular brand of terminal, the advance operation takes *milliseconds*, the delete operation takes d milliseconds, the replace operation takes r milliseconds, the insert operation takes i milliseconds, and the kill operation takes k milliseconds. So, for example, the sequence of operations to transform algorithm into altruistic takes time $6a + d + r + 4i + k$.

(a) Show that everything to the left of the cursor is a prefix of the new string, and everything to the right of the cursor is a suffix of the old string.
(b) Devise a dynamic programming algorithm that, given two strings x [1..n] and y[1..n], determines the fastest time needed to transform x to y. Analyze your algorithm.

3.2 Caminos Mínimos

A continuación se ha realizado una recopilación de los problemas sobre caminos mínimos entre todos los pares de nodos, estos problemas suelen recurrir al algoritmo de Floyd así como al algoritmo de Warshall.

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado5 (pág. 215)	Grafo ejemplo y matrices (Figs.7.4/5 pág. 216)	Pseudocódigo (pág. 213)	*The All-Pairs Shortest Path Problem*
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado5 (pág. 301)	Ejemplo de Algoritmo de Floyd (Fig. 8.5 pág. 302) Matriz con resultado (pág. 304)	Pseudocódigo (pág. 303)	*Caminos Mínimos*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado3 (pág. 208)	Grafo con ciclo negativo. (Fig. 5.3 pág. 209) Grafo dirigido, y matrices (Figs. 5.4/5 págs. 210/1)	Pseudocódigo. (pág. 210)	*All Pairs Shortest Path*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 9 (pág. 397)	Esquema de tabla (Fig. 13.7 pág. 423)	Pseudocódigo (pág. 424) Pseudocódigo - nº camino mínimo (pág. 425) Pseudocódigo - algoritmo Warshall (pág. 426)	*Camino mínimo*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 (pág. 253)	La ruta más corta - Grafo explicativo (Figs. 7.2,7.3,7.4,7.5,7.6 págs.254-258)	NO	*Ruta más corta*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett*.**	Capítulo 3 Apartado2 (pág. 94)	Grafo dirigido con pesos. (Fig. 3.2 pág. 95). Tablas pesos y longitudes (Fig. 3.3 pág. 97) Grafo explicativo (Fig. 3.4 pág. 98) Tabla camino mínimo (Fig. 3.5 pág. 102)	Pseudocódigo (pág. 100) Pseudocódigo - caminos más cortos. (pág.101) Pseudocódigo - Imprimir. (pág. 102)	*Floyd´sAlgorithm for shortest paths*
P7	I. Parberry. ***Problems on Algorithms.***	Capítulo 8 Apartado 4 (pág. 91)	NO	Pseudocódigo (pág. 91)	*Floyd´s Algorithm*
P8	S. Sahni ***Data Structures, Algorithms, and Applications in Java.***	Capítulo 20 Apartado 2.3 (pág. 815)	Ejemplo-Matriz (Fig. 20.4 pág. 819)	Pseudocódigo - iterativo (pág. 816) Código Java - iterativo. (págs. 817/8)	*All-Pairs Shortest Paths*
P9	R. Sedgewick, ***Algorithms in Java***	Capítulo 21 Apartado 3 (pág. 308)	Floyd´s algorithm (Fig. 21.14 pág. 310)	Pseudocódigo *Floyd's algorithm shortest path – Dynamic programming (*pág. 308*)*	*Shortest Paths*
P10	S. Sahni ***Data Structures, Algorithms, and Applications in Java.***	Capítulo 20 Apartado 2.4 (pág. 819)	NO	Pseudocódigo – CN iterativo. (pág. 821) Pseudocódigo - CN iterativo - refinado. (pág. 821)	*Single-Source Shortest Paths with Negative Cost. Algorithm Bellman-Ford*

				Pseudocódigo – CN iterativo- final. (pág. 822) Código Java - CN iterativo. (pág. 823)	
P11	*E. Horowitz y S. Sahni , **Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 7 (pág. 231)	Grafo dirigido y matriz. (Fig. 15 pág. 232)	NO	*The traveling sales person problem*
P12	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett*.**	Capítulo 3 Apartado 6 (pág. 123)	Grafo (Fig 3.16 pág. 124) Matriz (Fig. 3.17 pág. 124)	Pseudocódigo (págs. 126/7)	*The traveling sales person problem*

3.2.1 Caminos Mínimos [P1-9]

Problema:

Sea $G = \{N, A\}$ un grafo dirigido; N es el conjunto de nodos y A es el conjunto de aristas. Toda arista tiene asociada una longitud no negativa. Deseamos calcular la longitud del camino más corto entre cada par de nodos.

Es aplicable el principio de optimalidad: si k es un nodo del camino mínimo entre i y j, entonces la parte del camino que va desde i hasta k, y la parte del camino que va desde k hasta j debe ser óptimo también.

Si permitimos que las aristas del grafo tengan longitudes negativas, entonces la noción "camino más corto" pierde gran parte de su significado: si el grafo incluye un ciclo cuya longitud total sea negativa, ¡cuánto más pasemos por el bucle, más corto sería nuestro camino!

No se conoce un algoritmo eficiente para hallar los caminos simples más cortos en grafos que puedan tener aristas de longitud negativa. Este problema es NP-completo.

Visualizaciones:

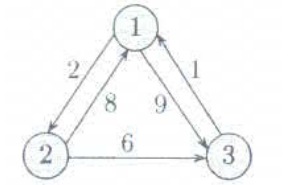
a)

$$d_{i,j}^{k} = \begin{cases} l[i,j] & \text{if } k = 0 \\ \min\{d_{i,j}^{k-1}, d_{i,k}^{k-1} + d_{k,j}^{k-1}\} & \text{if } 1 \leq k \leq n. \end{cases}$$

b)

$$D_k[i,j] = \min\{D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j]\}.$$

Ilustración 25 P1-Función Recursiva (a) y fórmula (b) del problema



An instance of the all pairs shortest path problem

The matrices D0, D1, D2, D3 are:

$$D_0 = \begin{bmatrix} 0 & 2 & 9 \\ 8 & 0 & 6 \\ 1 & \infty & 0 \end{bmatrix} \quad D_1 = \begin{bmatrix} 0 & 2 & 9 \\ 8 & 0 & 6 \\ 1 & 3 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 2 & 8 \\ 8 & 0 & 6 \\ 1 & 3 & 0 \end{bmatrix} \quad D_3 = \begin{bmatrix} 0 & 2 & 8 \\ 7 & 0 & 6 \\ 1 & 3 & 0 \end{bmatrix}.$$

Ilustración 26 P1-Grafo dirigido y sus matrices asociadas

Input: An $n \times n$ matrix $l[1..n, 1..n]$ such that $l[i, j]$ is the length of the edge (i, j) in a directed graph $G = (\{1, 2, \ldots, n\}, E)$.

Output: A matrix D with $D[i, j]$ = the distance from i to j.

```
1. D←l      {copy the input matrix l into D}
2. for k←1 to n
3.     for i←1 to n
4.         for j←1 to n
5.             D[i,j] = min{D[i,j], D[i,k] + D[k,j]}
6.         end for
7.     end for
8. end for
```

Ilustración 27 P1-Pseudocódigo

$$D_0 = L = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix}$$

$$D_1 = \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \qquad D_2 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

$$D_3 = \begin{pmatrix} 0 & 5 & 20 & 10 \\ 45 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \qquad D_4 = \begin{pmatrix} 0 & 5 & 15 & 10 \\ 20 & 0 & 10 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix}$$

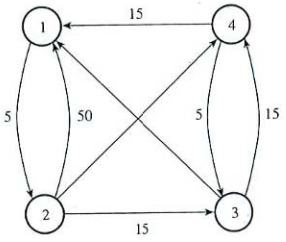
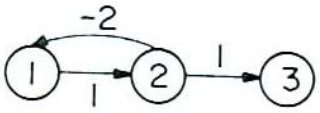


Ilustración 28 P2-Algoritmo de Floyd en funcionamiento

si $D[i, k]+D[k, j]<D[i, j]$ **entonces** $D[i, j] \leftarrow D[i, k]+D[k, j]$
$P[i, j] \leftarrow k$

$$P = \begin{pmatrix} 0 & 0 & 4 & 2 \\ 4 & 0 & 4 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}.$$

Ilustración 29 P2-Formula y matriz resultante que muestran el camino más corto



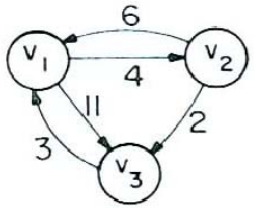
$$\begin{bmatrix} 0, & 1, & \infty \\ -2, & 0, & 1 \\ \infty, & \infty, & 0 \end{bmatrix}$$

Ilustración 30 P3-Grafo con ciclo negativo y matriz asociada

```
    procedure ALL_PATHS(COST, A, n)
      //COST(n, n) is the cost adjacency matrix of a graph with n ver-//
      //tices; A(i, j) is the cost of a shortest path from v_i to v_j//
      //COST(i,i) = 0, 1 ≤ i ≤ n//
      integer i, j, k, n; real COST(n, n), A(n, n)
1     for i ← 1 to n do
2       for j ← 1 to n do
3         A(i, j) ← COST(i, j)   //copy COST into A//
4       repeat
5     repeat
6     for k ← 1 to n do   //for a path with highest vertex index k//
7       for i ← 1 to n do   //for all possible pairs of vertices//
8         for j ← 1 to n do
9           A(i, j) ← min{A(i, j), A(i, k) + A(k, j)}
10        repeat
11      repeat
12    repeat
13  end ALL_PATHS
```

Algorithm 5.3 Procedure to compute lengths of shortest paths

Ilustración 31 P3 –Pseudocódigo.



	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

Ilustración 32 P3-Grafo dirigido y Matriz de costes asociada.

$A^{(0)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	∞	0

$A^{(1)}$	1	2	3
1	0	4	11
2	6	0	2
3	3	7	0

$A^{(2)}$	1	2	3
1	0	4	6
2	6	0	2
3	3	7	0

$A^{(3)}$	1	2	3
1	0	4	6
2	5	0	2
3	3	7	0

Ilustración 33 P3-Matrices A^k producidas por el pseudocódigo para el grafo de la Ilustración 28.

$C^k(i, j)$ = coste *mínimo* para ir de i a j pudiendo utilizar como vértices intermedios aquellos entre 1 y k.

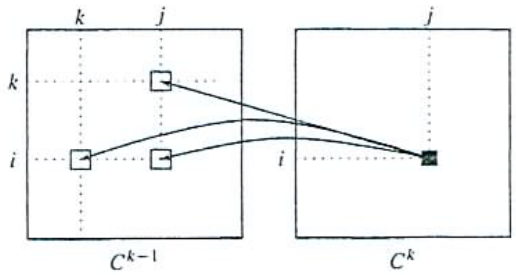


Ilustración 34 P4-Esquema de tabla para el algoritmo de Floyd.

a)

$$W^k(k, j) = W^{k-1}(k, j) \vee (W^{k-1}(k, k) \wedge W^{k-1}(k, j)) = W^{k-1}(k, j),$$

b)

```
fun Warshall(R : grafo[n]) dev W[1..n, 1..n] de bool
    { inicialización }
    W := R
    para i = 1 hasta n hacer W[i, i] := cierto fpara
    { actualizaciones }
    para k = 1 hasta n hacer
        para i = 1 hasta n hacer
            para j = 1 hasta n hacer
                W[i, j] := W[i, j] ∨ (W[i, k] ∧ W[k, j])
            fpara
        fpara
    fpara
ffun
```

Ilustración 35 P4–Fórmula (a) y Pseudocódigo(b) del algoritmo de Warshall.

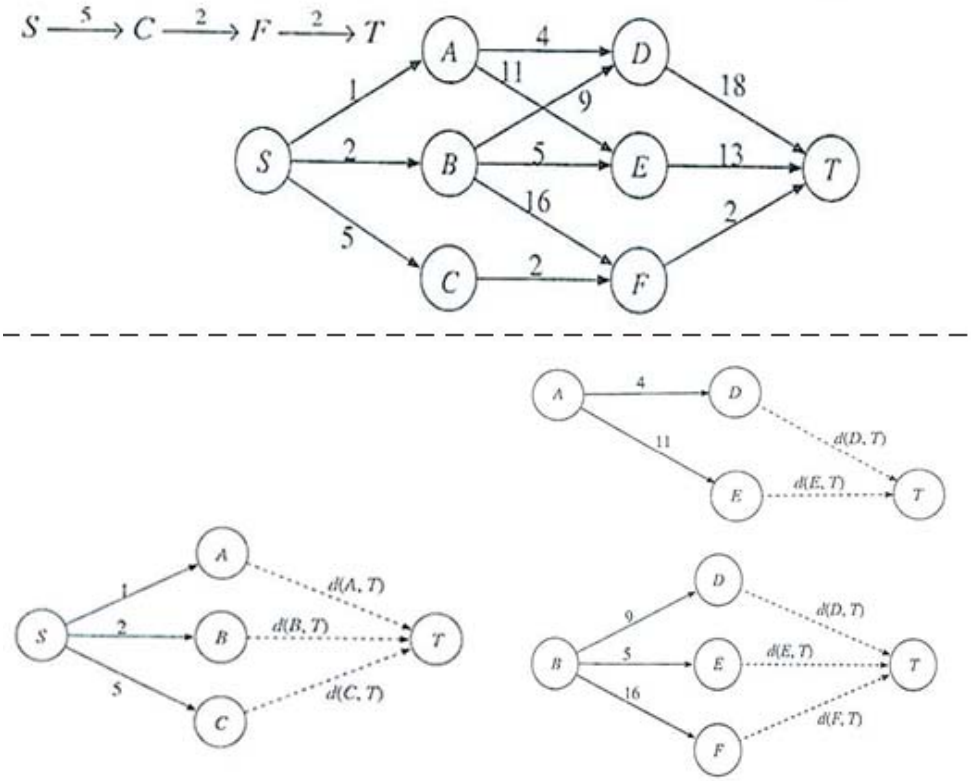


Ilustración 36 P5-Ejemplo de ruta más corta y grafos de cálculo paso a paso.

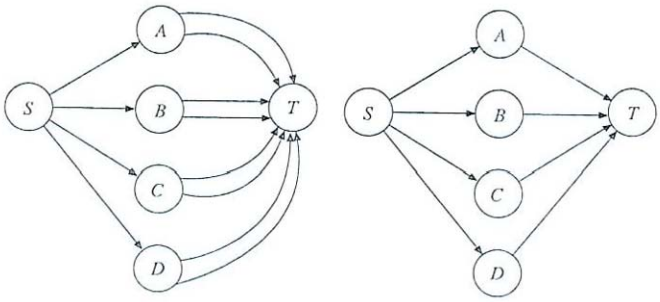
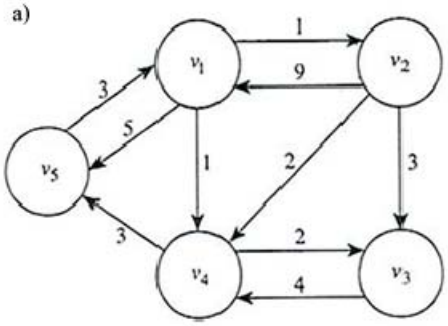


Ilustración 37 P5-Eliminación de soluciones con el método de programación dinámica.

a)



b)

	1	2	3	4	5
1	0	1	∞	1	5
2	9	0	3	2	∞
3	∞	∞	0	4	∞
4	∞	∞	2	0	3
5	3	∞	∞	∞	0

W

	1	2	3	4	5
1	0	1	3	1	4
2	8	0	3	2	5
3	10	11	0	4	7
4	6	7	2	0	3
5	3	4	6	4	0

D

Ilustración 38 P6-Grafo dirigido con pesos (a) y matrices de pesos W y longitudes D asociadas (b).

```
procedure floyd2 (n: integer;
                  W: array[1..n,1..n] of number;
              var D: array[1..n,1..n] of number;
              var P: array[1..n,1..n] of index);
var
  i,j,k: index;
begin
  for i:=1 to n do
    for j:=1 to n do
      P[i,j]:= 0
    end
  end;
  D:= W;
  for k:= 1 to n do
    for i:= 1 to n do
      for j:= 1 to n do
        if D[i,k] + D[k,j] < D[i,j] then
          P[i,j]:= k;
          D[i,j]:= D[i,k] + D[k,j]
        end
      end
    end
  end
end;
```

Ilustración 39 P6-Pseudocódigo que calcula matriz (*P*) con camino más corto.

Problem: Same as in Algorithm 3.3, except shortest paths are also created.

Additional outputs: an array P, where

$$P[i, j] = \begin{cases} \text{highest index of an intermediate vertex on the shortest path} \\ \text{from } v_i \text{ to } v_j \text{ if at least one intermediate vertex exists.} \\ 0 \text{ if no intermediate vertex exists.} \end{cases}$$

	1	2	3	4	5
1	0	0	4	0	4
2	5	0	0	0	4
3	5	5	0	0	4
4	5	5	0	0	0
5	0	1	4	1	0

Ilustración 40 P6-Salida de *P* según pseudocódigo de Ilustración 35 del ejemplo de la Ilustración 34.

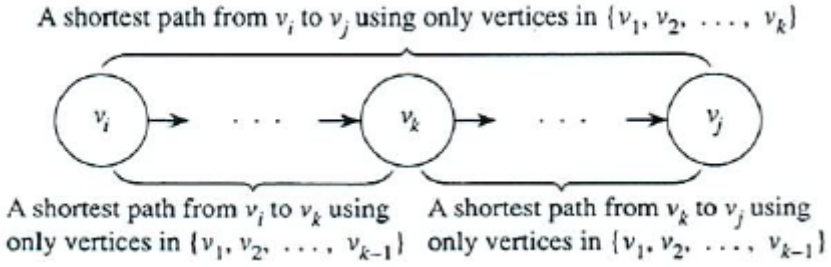
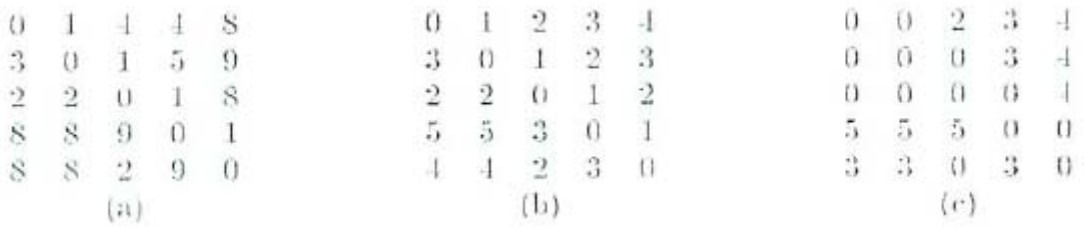


Ilustración 41 P6-Camino más corto



0	1	4	4	8
3	0	1	5	9
2	2	0	1	8
8	8	9	0	1
8	8	2	9	0

(a)

0	1	2	3	4
3	0	1	2	3
2	2	0	1	2
5	5	3	0	1
4	4	2	3	0

(b)

0	0	2	3	4
0	0	0	3	4
0	0	0	0	4
5	5	5	0	0
3	3	0	3	0

(c)

Ilustración 42 P8-Matrices con resultados de ejemplo

```
class GraphSPall
{ private Edge[][] p;
  private double[][] d;
  GraphSPall(Graph G)
    { int V = G.V();
      p = new Edge[V][V]; d = new double[V][V];
      for (int s = 0; s < V; s++)
        for (int t = 0; t < V; t++)
          d[s][t] = maxWT;
      for (int s = 0; s < V; s++)
        for (int t = 0; t < V; t++)
          if (G.edge(s, t) != null)
            { p[s][t] = G.edge(s, t);
              d[s][t] = G.edge(s, t).wt(); }
      for (int s = 0; s < V; s++) d[s][s] = 0.0;
      for (int i = 0; i < V; i++)
        for (int s = 0; s < V; s++)
          if (p[s][i] != null)
            for (int t = 0; t < V; t++)
              if (s != t)
                if (d[s][t] > d[s][i] + d[i][t])
                  { p[s][t] = p[s][i];
                    d[s][t] = d[s][i] + d[i][t]; }
    }
  Edge path(int s, int t)
    { return p[s][t]; }
  double dist(int s, int t)
    { return d[s][t]; }
}
```

Ilustración 43 P9-Pseudocódigo

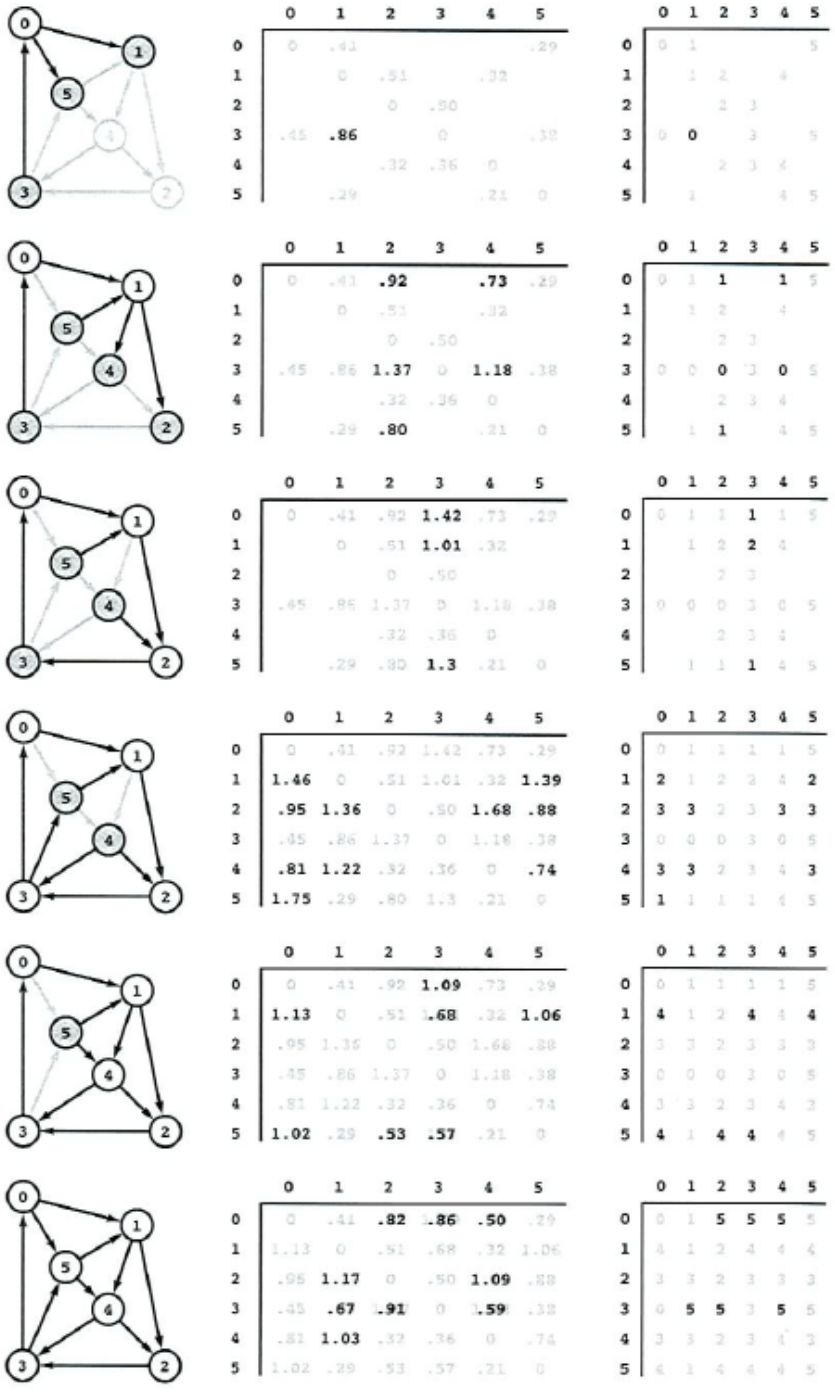


Ilustración 44 P9-Secuencia de imágenes de la construcción del camino

3.2.2 Single-Source Shortest Paths with Negative Cost [P10]

Problema:
Consider a graph in which vertices represent cities, and an edge cost gives the cost of renting a car in one city and dropping it off in another. Most edge costs are positive. However, if there is a net migration into Florida, then Florida will have a surplus of cars and cities that are losing population will have no cars. To fix this imbalance in rental cars, the rental company will actually pay you to drive a car from a select Florida city to one that is short of cars. So some edges have a negative cost. The Web sites shows how to solve a scheduling problem and a system o difference equation by finding shortest paths in a graph that have negative edge cost.

Visualizaciones:

```
initialize d(*) = d(*,0);
put the vertices that are adjacent from the source vertex onto a list list1;

// compute d(*) = d(*, n - 1)
put the source vertex into list1;
for (int k = 1; k < n; k++)
{
   // see if there are vertices whose d value has changed
   if (list1 is empty) break;  // no such vertex
   while (list1 is not empty)
   {
      delete a vertex u from list1;
      for (each edge (u, v))
      {
         d(v) = min{d(v), d(u) + cost(u, v)};
         if (d(v) has changed and v is not on list2) add v to list2;
      }
      list1 = list2;
      make list2 empty;
   }
}
```

Ilustración 45 P10- Pseudocódigo del algoritmo de Bellman-Ford.

3.2.3 The Traveling Salesperson Problem [P11-12]

Problema:
Suppose a salesperson is planning a sales trip that includes 20 cities. Each city is connected to some of other cities by a road. To minimize travel time, we want to determinate a shortest route that starts at the salesperson's home city, visits each of the cities ones, and ends up at the home city. We generalize the problem to include the case where the weight (distance) going in one direction can be different than the weight going in another direction. We assume that the weights are nonnegative numbers.

A tour (also called a Hamiltonian Circuit) in a directed graph is a path from a vertex to itself that passes through each of the other vertices exactly once. An optimal tour in a weighted, directed graph is such a path of minimum length. The Traveling Salesperson Problem is to find an optimal tour in a weighted, directed graph when at least one tour exists.

For $i \neq 1$ and i not in A,

$D[v_i, A] = minimum(W[i, j] + D[v_j, A - \{v_j\}])$ if $A \neq \emptyset$
$D[v_i, \emptyset] = W[i, 1]$

Visualizaciones:

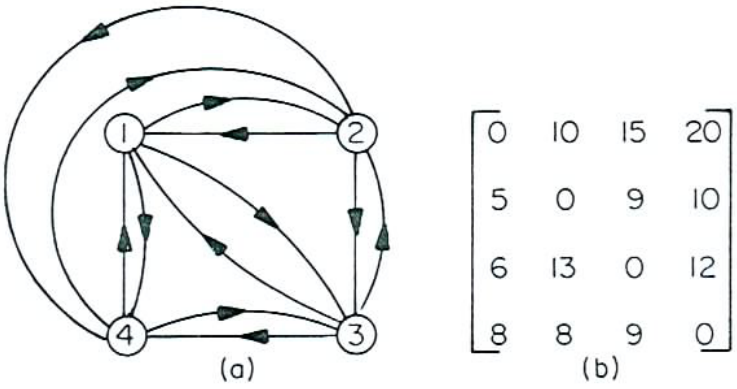
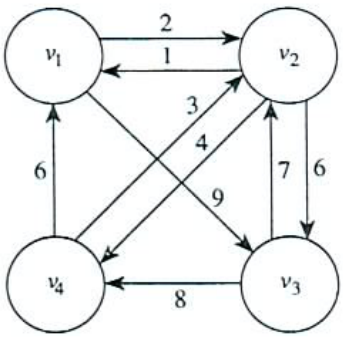


Ilustración 46 P11-Grafo dirigido a) y matriz con longitud de arcos b). El camino óptimo es 1 2 4 3 1.

a)



b)

$length[v_1, v_2, v_3, v_4, v_1] = 22$

$length[v_1, v_3, v_2, v_4, v_1] = 26$

$length[v_1, v_3, v_4, v_2, v_1] = 21$

c)

	1	2	3	4
1	0	2	9	∞
2	1	0	6	4
3	∞	7	0	8
4	6	3	∞	0

Ilustración 47 P12-Grafo dirigido a) longitud de los caminos b) y matriz de adyacencia c) del grafo a). El camino óptimo es v_1 v_3 v_4 v_2 v_1.

```
procedure travel(n: integer;
                 W: array[1..n,1..n] of number;
              var P: array[1..n,subset of V−{v1}] of index;
      var minlength: number);
var
  i,j,k: index;
  D: array[1..n,subset of V−{v1}] of number;
begin
  for i:= 2 to n do
    D[i,∅]:= W[i,1]
  end;
  for k:= 1 to n−2 do
    for all subsets A ⊆ V−{v1} containing k vertices do
      for i such that i≠1 and vi is not in A do
        D[i,A]:= minimum(W[i,j] + D[vj,A−{vj}]);
                 vj∈A
        P[i,A]:= value of j that gave the minimum
      end
    end
  end;
  D[1, V−{v1}]:= minimum(W[1,j] + D[vj, V−{v1}]);
                 2≤j≤n
  P[1, V−{v1}]:= value of j that gave the minimum
  minlength:= D[1, V−{v1}]
end;
```

Ilustración 48 P12-Pseudocódigo

3.3 Coeficientes Binomiales

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 1.1 (pág. 292)	Triángulo de Pascal (Fig. 8.1 pág. 291)	Pseudocódigo - sin pd (pág. 291).	*Cálculo del coeficiente binomial.*
P2	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Capítulo 3 Apartado 1 (pág. 90)	Llenado de array (Fig, 3.1 págs. 92/3)	Código Pascal - divide y vencerás (pág. 91) Código Pascal – pd (pág. 93)	*The Binomial Coefficient*
P3	S. Skiena. ***The Algorithm Design Manual***	Capítulo 8 Apartado 1 (pág. 278)	Triángulo de Pascal (pág. 278) Tablas evaluación orden (Fig. 8.3 pág. 279)	Pseudocódigo –pd (pág. 280) Código C - pd (http://www.cs.sunysb.edu/~skiena/392/programs/binomial.c) Código Java (http://www.cs.sunysb.edu/~skiena/392/javaprograms/)	*Binomail Coefficients*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 1 (pág. 397)	Triángulo de Pascal (pág. 397) Árbol llamadas recursivas (Fig. 13.1 pág. 398)	NO	*Esquema general. Coeficientes binomiales*
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 18 (pág. 447)	Triángulo de Pascal (Tabla 13.2 pág. 448)	Pseudocódigo (pág. 448)	*Número combinatorio*

3.3.1 Cálculo Coeficiente Binomial [P1-4]

Problema:
[G. Brassard y P. Bratley. *Fundamentos de algoritmia*, pág. 292]
Considere el problema consistente en calcular el coeficiente binomial:

$$\binom{n}{k} = \begin{cases} 1 & \text{si } k = 0 \text{ o } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{si } 0 < k < n \\ 0 & \text{en caso contrario} \end{cases}$$

Supongamos que $0 \leq k \leq n$.

Visualizaciones:

a)

función $C(n, k)$
si $k = 0$ o $k = n$ entonces devolver 1
sino devolver $C(n-1, k-1) + C(n-1, k)$

b)

	0	1	2	3	...	k-1	k
0	1						
1	1	1					
2	1	2					
⋮							
n-1						$C(n-1,k-1)$	$C(n-1,k)$
n						+ ↓	
							$C(n,k)$

Ilustración 49 P1-Pseudocódigo cálculo coeficiente binomial (a) y Triángulo de Pascal (b)

---

a)

$$B[i, j] = \begin{cases} B[i-1, j-1] + B[i-1, j] & 0 < j < i \\ 1 & j = 0 \text{ or } j = i \end{cases}$$

b)

	0	1	2	3	4		j	k
0	1							
1	1	1						
2	1	2	1					
3	1	3	3	1				
4	1	4	6	4	1			
						$B[i-1, j-1]$	$B[i-1, j]$	
						→	↓	
i							$B[i, j]$	
n								

Ilustración 50 P2-Función recursiva y llenado del array B empleado en el cálculo del coeficiente binomial.

Problem: Compute the binomial coefficient.

Inputs: nonnegative integers n and k, where $k \leq n$

Outputs: *bin2*, the binomial coefficient $\binom{n}{k}$.

```
function bin2(n, k: integer): integer;
var
   i,j: index;
   B: array[0..n,0..k] of integer;
begin
   for i:= 0 to n do
      for j:= 0 to minimum(i,k) do
         if j = 0 or j = i then
            B[i,j]:= 1
         else
            B[i,j]:= B[i-1,j-1] + B[i-1,j]
         end
      end
   end;
   bin2:= B[n,k]
end;
```

Ilustración 51 P2-Pseucocódigo Coeficiente Binomial con pd

---

a)

b)

m / n	0	1	2	3	4	5
0	A					
1	B	G				
2	C	1	H			
3	D	2	3	I		
4	E	4	5	6	J	
5	F	7	8	9	10	K

m / n	0	1	2	3	4	5
0	1					
1	1	1				
2	1	1	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	**5**	1

c)

```
long binomial_coefficient(n,m)
int n,m;                            /* computer n choose m */
{
        int i,j;                    /* counters */
        long bc[MAXN][MAXN];        /* table of binomial coefficients

        for (i=0; i<=n; i++) bc[i][0] = 1;

        for (j=0; j<=n; j++) bc[j][j] = 1;

        for (i=1; i<=n; i++)
                for (j=1; j<i; j++)
                        bc[i][j] = bc[i-1][j-1] + bc[i-1][j];

        return( bc[n][m] );
}
```

Ilustración 52 P3-Triángulo de Pascal (a). Evaluación del orden de *binomial_coefficient* de M[5,4] e inicialización de condiciones A-K, evaluación de recurrencias 1-10. Contenido de la matriz después de la evaluación (b). Pseudocódigo (c)

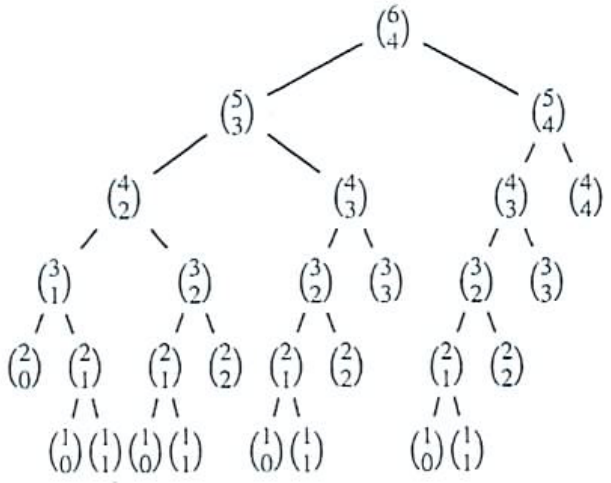


Ilustración 53 P4-Llamadas recursivas

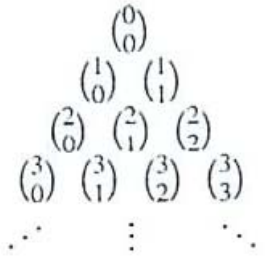


Ilustración 54 P4-Triángulo de Pascal

	0	1	2	...	r
0	1	0	0	...	0
1	1	1	0	...	0
2	1	2	1	...	0
⋮	⋮	⋮	⋮	⋱	⋮
n	1	n	$\binom{n}{2}$	...	$\binom{n}{r}$

Ilustración 55 P6-Triángulo de Pascal

```
{ r < n }
fun pascal(n, r : nat) dev c : nat
var C[0..r] de nat
    C[0] := 1
    C[1..r] := [0]
    para i = 1 hasta n hacer
        para k = r hasta 1 paso − 1 hacer
            C[k] := C[k] + C[k − 1]
        fpara
    fpara
    c := C[r]
ffun
{ c = (n r) }
```

Ilustración 56 P6-Pseudocódigo

3.3.2 Otros Problemas

Ordenación [P7]

Problema:
Se dispone de n objetos que es necesario ordenar empleando las relaciones de < y =. Por ejemplo 3 objetos se tienen 13 ordenaciones posibles:

$a = b = c$	$a = b < c$	$a < b = c$	$a < b < c$	$a < c < b$
$a = c < b$	$b < a = c$	$b < a < c$	$b < c < a$	$b = c < a$
$c < a = b$	$c < a < b$	$c < b < a$		

Desarrollar un algoritmo para calcular en función de ordenaciones posibles, invirtiendo un tiempo que esté en $O(n^2)$ y un coste en espacio auxiliar que esté en $O(n)$.

3.4 Problemas de Árboles

A continuación se recopila diferentes problemas de árboles recogidos de diversas fuentes. En este caso se trata el problema sobre "Construcción óptima de un árbol de búsqueda binario".

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 4 (pág. 466)	Árbol búsqueda binario (Fig. 10.3 pág. 466) Eligiendo K_k como raíz (Fig. 10.4 pág. 469) Coste de computación (Fig. 10.5 pág. 470)	Pseudocódigo (págs. 470/1)	*Constructing Optimal Binary Search Trees*
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Capítulo IV Apartato15.5 (pág. 397)	Ejemplo árboles (Fig. 15.9 pág. 398) Tablas con valores (Fig. 15.10 pág. 403)	Pseudocódigo (pág. 402)	*Optimal Binary Search Trees*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 4 (pág. 211)	Árboles (Fig. 5.6 pág. 212) Árboles - nodos externos (Fig. 5.7 pág. 213) Árboles - ejemplo (Ex. 5.15 pág. 214/5) Árbol - óptimo (Fig. 5.8 pág. 216) Tabla valores ejemplo (Fig. 5.9 pág. 218) Árbol óptimo - ejemplo (Fig. 5.10 pág. 218)	Pseudocódigo (pág. 212) Pseudocódigo - mínimo coste (pág. 219)	*Optimal Binary Search Trees*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 7 (pág. 414)	NO	Pseudocódigo (pág. 416/7)	*Árbol de búsqueda binario óptimo*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 6 (pág. 283)	Ilustraciones (Figs7.19-24 págs. 284-90)	NO	*El problema del árbol binario óptimo*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett.***	Capítulo 3 Apartado 5 (pág. 113)	Árboles binarios (Fig. 3.10 pág. 114) Árboles binarios-ejemplos (Fig. 3.11/12 págs. 117/8) Eligiendo Key_k como raíz (Fig. 3.13 pág. 119) Tabla de cálculo de valores y árbol de un ejemplo (Fig. 3.14/5 pág.122/3)	Pseudocódigo (págs. 115/6) Pseudocódigo óptimo (págs. 120/1) Pseudocódigo construcción (págs. 121/2)	*Optimal Binary Search Trees*
P7	Ian Parberry. ***Problems on Algorithms.***	Capítulo 8 Apartado 3 (pág. 90)	NO	Pseudocódigo – pd (pág. 90)	*Optimal Binary Search Trees*
P8	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 7 (pág. 283)	Ilustraciones (Fig. 7.25/33 págs. 291/9)	Pseudocódigo (págs. 299-300)	*El problema ponderado de dominación perfecta en árboles*

3.4.1 Árboles de Búsqueda Binaria Óptimos [P1-7]

Problema:
Dados n identificadores $a_1<a_2<a_3\ldots a_n$, puede tenerse muchos árboles binarios distintos.

Para un árbol binario dado, los identificadores almacenados cerca de la raíz del árbol pueden buscarse de manera más bien rápida, mientras que para recuperar datos almacenados lejos de la raíz se requieren varios pasos. A cada identificador a_i se asociará una probabilidad P_i, que es la probabilidad de que este identificador sea buscado. Para un identificador que no está almacenado en el árbol, también se supondrá que hay una probabilidad asociada con el identificador. Los datos se separan en n+1 clases de equivalencia. Q_i denota la probabilidad de que X sea buscado donde $a_i<X<a_{i+1}$, en el supuesto de que a_0 es $-\infty$ y a_{n+1} es $+\infty$. Las probabilidades cumplen la siguiente ecuación:

$$\sum_{i=1}^{n} P_i + \sum_{i=0}^{n} Q_i = 1.$$

Sea $L(a_i)$ el nivel del árbol binario donde se almacena a_i. Entonces la recuperación de a_i requiere $L(a_i)$ pasos si se hace que la raíz del árbol esté en el nivel 1.

El costo esperado de un árbol binario puede expresarse como sigue:

$$\sum_{i=1}^{n} P_i L(a_i) + \sum_{i=0}^{n} Q_i (L(E_i) - 1).$$

Un árbol binario óptimo es un árbol binario donde se ha minimizado el costo anterior.

Visualizaciones:

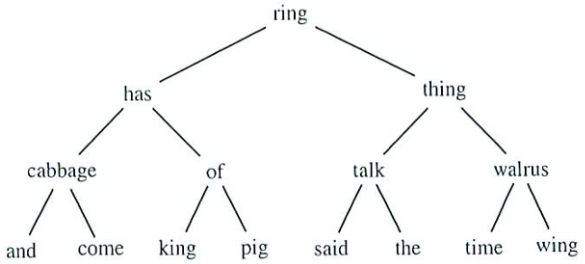


Ilustración 57 P1-Árbol de búsqueda binaria

Key	Number of searches	Probability (p_i)	Comparisons (c_i)	$p_i c_i$
and	30	150	4	.600
cabbage	5	025	3	.075
come	10	050	4	.200
has	5	025	2	.050
king	10	050	4	.200
of	25	125	3	.375
pig	5	025	4	.100
ring	15	075	1	.075
said	15	075	4	.300
talk	10	050	3	.150
the	30	150	4	.600
thing	15	075	2	.150
time	10	050	2	.100
walrus	5	025	3	.075
wing	10	050	4	.200
	Total = 200	Total = 1 000		Total = 3.250

Ilustración 58 P1-Datos y tiempo medio de búsqueda de computación de las claves o nodos

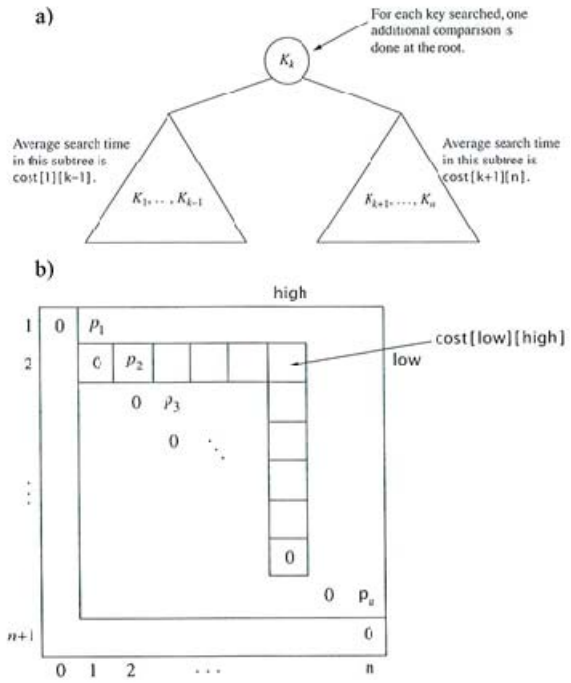
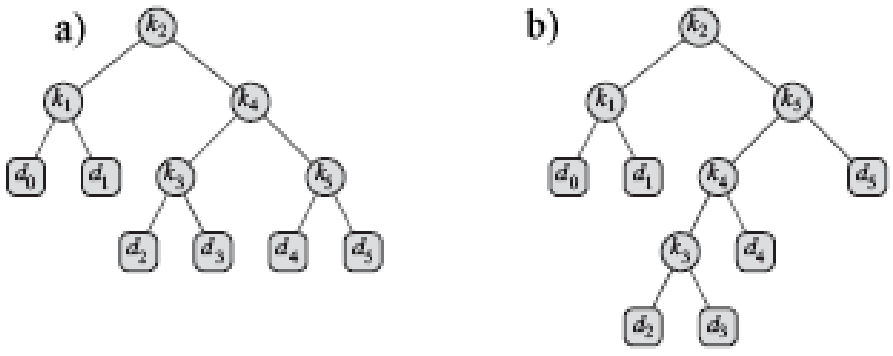


Ilustración 59 P1-Eligiendo K_k como raíz (a) y Computación de cost[low][high]

```
optimalBST(prob, n, cost, root)  // OUTLINE
    for (low = n + 1; low ≥ 1; low --)
        for (high = low-1; high ≤ n; high ++)
            bestChoice(prob, cost, root, low, high);
    return cost;

/** Compute solution of subproblem (low, high) */
bestChoice(prob, cost, root, low, high)  // OUTLINE
    if (high < low)
        bestCost = 0;  // empty tree
        bestRoot = -1;
    else
        bestCost = ∞;
    for (r = low; r ≤ high; r ++)
        rCost = p(low, high) + cost[low][r-1] + cost[r+1][high];
        if (rCost < bestCost)
            bestCost = rCost;
            bestRoot = r;
        // Continue loop
    cost[low][high] = bestCost;
    root[low][high] = bestRoot;
    return;
```

Ilustración 60 P1-Pseudocódigo



i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Ilustración 61 P2-Árboles de búsqueda binaria con n=5 (claves o nodos) y la tabla con sus probabilidades asociadas. a) Árbol de búsqueda binaria con coste de búsqueda esperado 2.80. b) Árbol de búsqueda binaria con coste de búsqueda esperado 2.75, este árbol es óptimo.

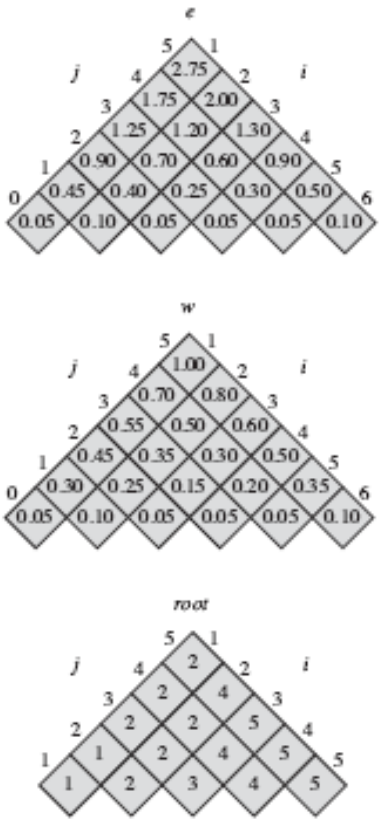


Ilustración 62 P2-Tablas e[i, j], w[i, j], root[i, j] obtenidas por OPTIMAL-BST según la clave de la figura 46. Las tablas se han rotado por lo que las diagonales aparecen en la horizontal.

OPTIMAL-BST(p, q, n)

```
1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
       and root[1..n, 1..n] be new tables
2  for i = 1 to n + 1
3      e[i, i-1] = q_{i-1}
4      w[i, i-1] = q_{i-1}
5  for l = 1 to n
6      for i = 1 to n - l + 1
7          j = i + l - 1
8          e[i, j] = ∞
9          w[i, j] = w[i, j-1] + p_j + q_j
10         for r = i to j
11             t = e[i, r-1] + e[r+1, j] + w[i, j]
12             if t < e[i, j]
13                 e[i, j] = t
14                 root[i, j] = r
15 return e and root
```

Ilustración 63 P2-Pseudocódigo

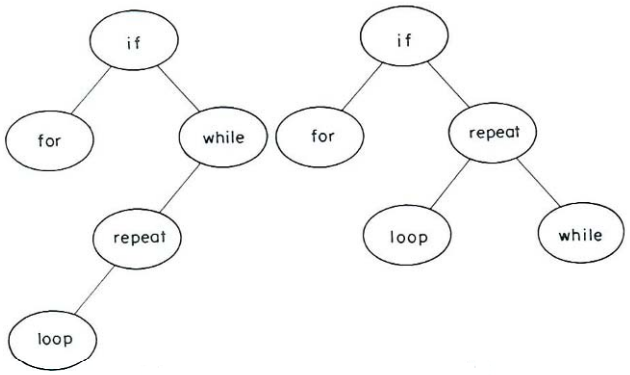


Ilustración 64 P3-Árboles de búsqueda binaria

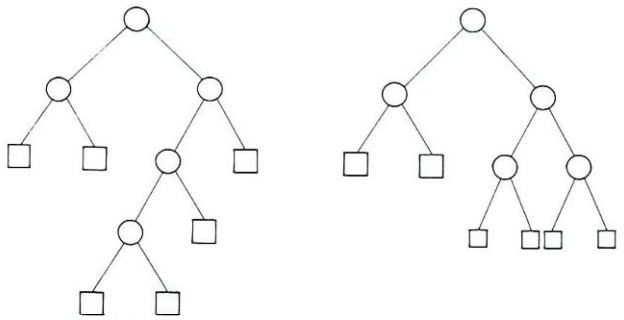


Ilustración 65 P3-Árboles de la Ilustración 49 con nodos externos añadidos

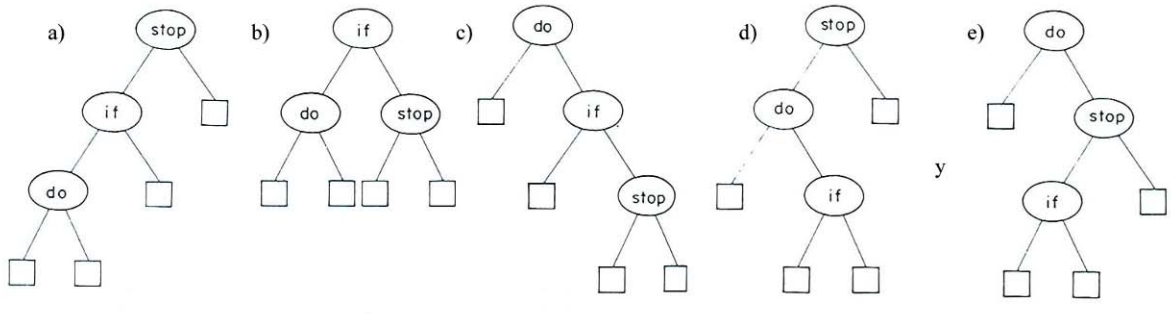


Ilustración 66 P3-Posibles árboles de búsqueda binaria para (a1, a2, a3) = (do, if, stop)

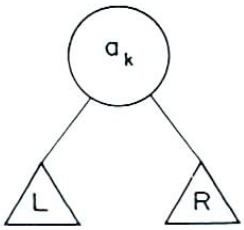


Ilustración 67 P3-Árbol de búsqueda binario óptimo con raíz a_k

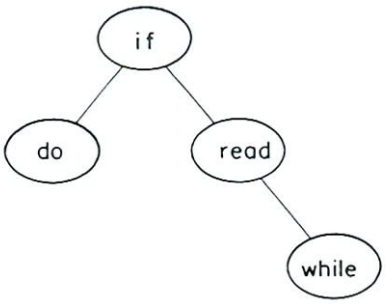


Ilustración 68 P3-Árbol de búsqueda binario óptimo para n = 4 (a_1, a_2, a_3, a_4) = (do, if, read, while)

row ↓ / column →	0	1	2	3	4
0	2, 0, 0	3, 0, 0	1, 0, 0	1, 0, 0	1, 0, 0
1	8, 8, 1	7, 7, 2	3, 3, 3	3, 3, 4	
2	12, 19, 1	9, 12, 2	5, 8, 3		
3	16, 25, 2	11, 19, 2			
4	16, 32, 2				

Ilustración 69 P3-Valores de C(0,4), W(0,4) y R(0,4)

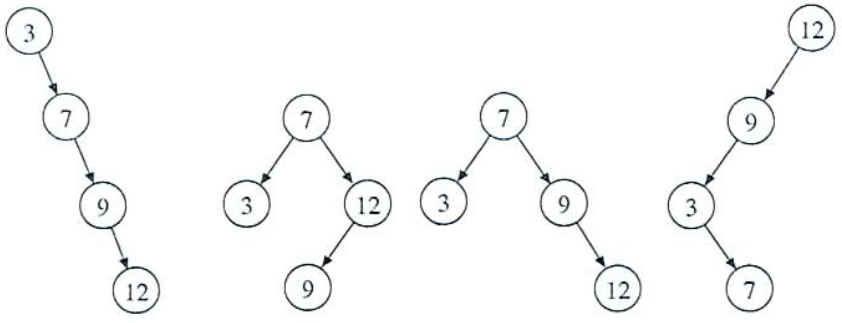


Ilustración 70 P5-Cuatro árboles binarios distintos para el conjunto de datos: 3, 7, 9, 12

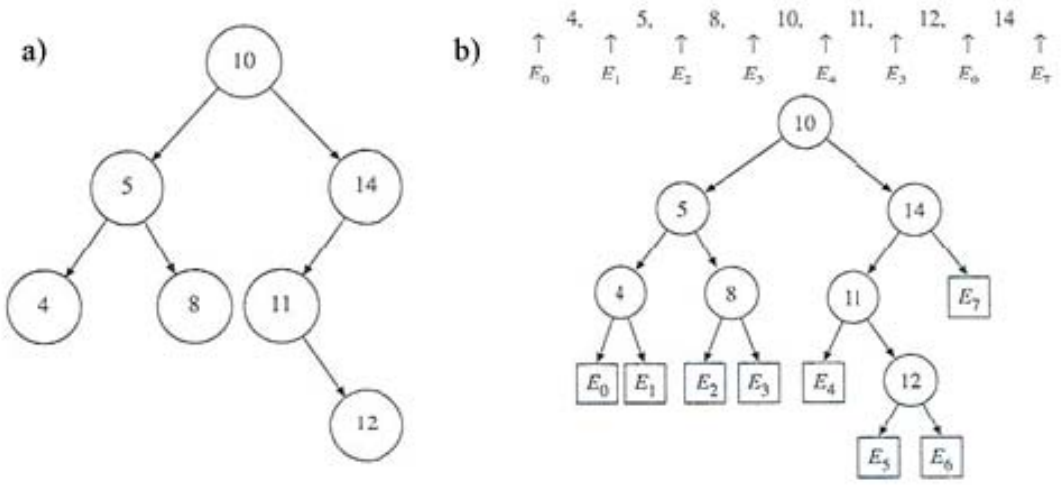
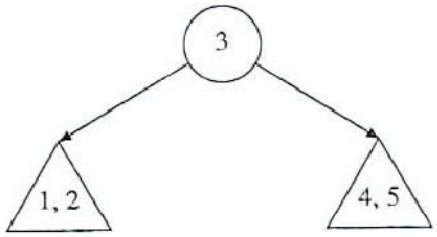


Ilustración 71 P5-Árbol binario(a) y árbol binario con nodos externos agregados (b)



a) Un subárbol que contiene a 1 y 2 con 1 como la raíz.
b) Un subárbol que contiene a 1 y 2 con 2 como la raíz.
c) Un subárbol que contiene a 4 y 5 con 4 como la raíz.
d) Un subárbol que contiene a 4 y 5 con 5 como la raíz.

Ilustración 72 P5-Árbol binario con cierto identificador seleccionado como la raíz

---

$$C(i, j) = \min_{i \le k \le j} \left\{ C(i, k-1) + C(k+1, j) + \sum_{l=i}^{j} (P_l + Q_l) + Q_{i-1} \right\}$$

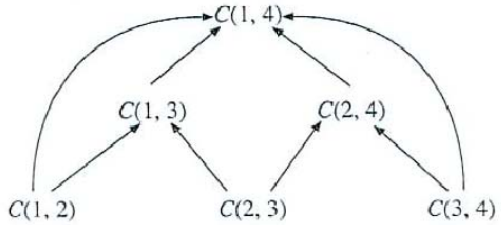


Ilustración 73 P5-Fórmula del cote de árbol binario óptimo y relación de cálculo de subárboles

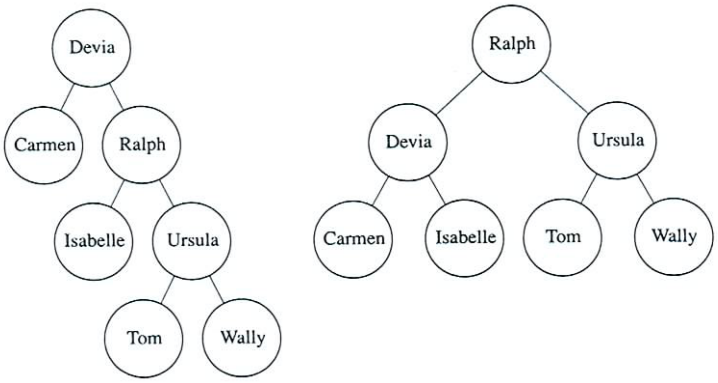


Ilustración 74 P6-Dos árboles de búsqueda binaria con las mismas claves o nodos

---

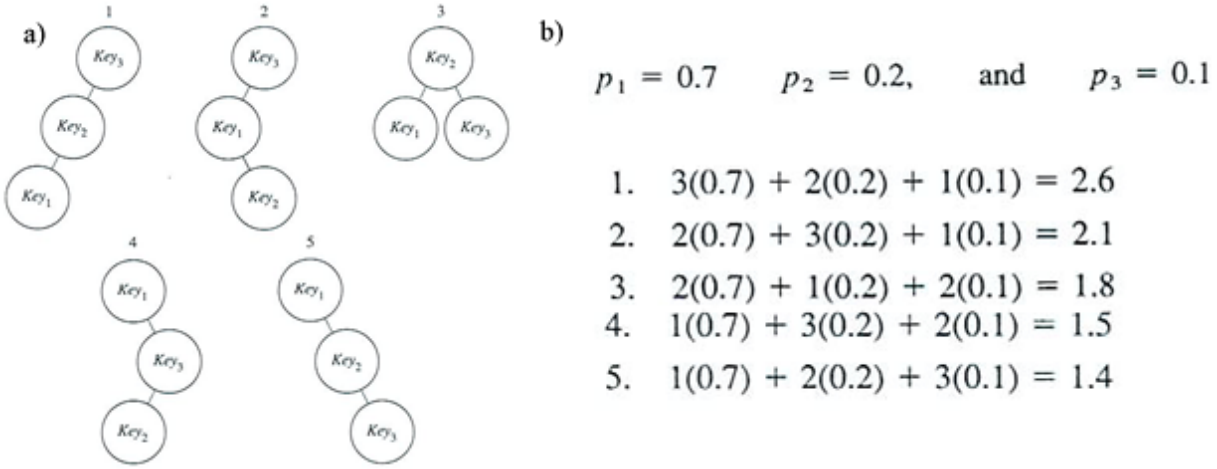


Ilustración 75 P6-Árboles de búsqueda binaria con clave n = 3 (a) y tiempo medio de búsqueda (b) con p1, p2 p3 para los árboles de a). El árbol 5 (a) es óptimo

---

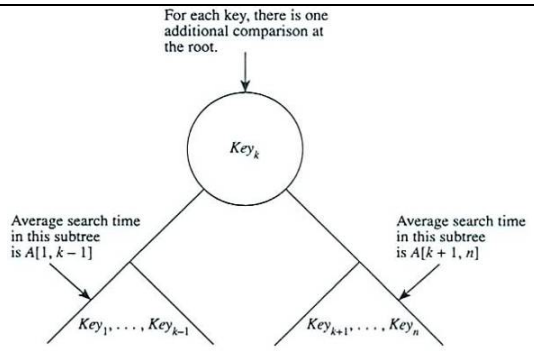


Ilustración 76 P6-Árbol de búsqueda binario óptimo teniendo como raíz Key $_k$

a)

Don	Isabelle	Ralph	Wally
Key[1]	*Key*[2]	*Key*[3]	*Key*[4]

$p_1 = \frac{3}{8} \quad p_2 = \frac{3}{8} \quad p_3 = \frac{1}{8} \quad p_4 = \frac{1}{8}.$

b)

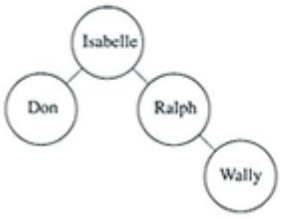
A

	0	1	2	3	4
1	0	$\frac{3}{8}$	$\frac{9}{8}$	$\frac{11}{8}$	$\frac{7}{4}$
2		0	$\frac{3}{8}$	$\frac{5}{8}$	1
3			0	$\frac{1}{8}$	$\frac{3}{8}$
4				0	$\frac{1}{8}$
5					0

R

	0	1	2	3	4
1	0	1	1	2	2
2		0	2	2	2
3			0	3	3
4				0	4
5					0

c)




```
function tree(i,j: index): node_pointer;
var
  k: index;
  p: node_pointer;
begin
  k:= R[i,j];
  if k = 0 then
    tree:= nil
  else
    p:= getnode;
    p↑.key:= Key[k];
    p↑.left:= tree(i,k−1);
    p↑.right:= tree(k+1, j);
    tree:= p
  end
end;
```

Ilustración 77 P6-Con valores del array Key y $p_{1..4}$ a) se obtienen los arrays A y R b) aplicando el algoritmo de la *Ilustración 78*. (c) Árbol obtenido tras aplicar el algoritmo de c) y la *Ilustración 78* con los valores de a).

```
procedure optsearchtree(n: integer:
                        p: array[1..n] of real;
                var minavg: real;
                    var R: array[1..n+1,0..n] of index);
var
  i,j,k,diagonal: index;
  A: array[1..n+1,0..n] of real;
begin
  for i:= 1 to n do
    A[i,i−1]:= 0;
    A[i,i]:= p[i];
    R[i,i]:= i;
    R[i,i−1]:= 0
  end;
  A[n+1,n]:= 0;
  R[n+1,n]:= 0;
  for diagonal:= 1 to n−1 do {Diagonal 1 is just above main diagonal
    for i:= 1 to n−diagonal do
      j:= i+diagonal;
      A[i,j]:= minimum(A[i,k−1] + A[k+1,j]) + Σ_{m=i}^{j} p_m;
               i≤k≤j
      R[i,j]:= a value of k that gave the minimum
    end
  end;
  minavg:= A[1,n]
end;
```

Ilustración 78 P6–Pseudocódigo.

3.4.2 El Problema Ponderado de Dominación Perfecta en Árboles [P8]

Problema:

Se tiene una gráfica $G = (V, E)$ donde cada vértice $v \in V$ tiene un costo $c(v)$ y cada arista $e \in E$ también tiene un costo $c(e)$. Un conjunto dominante perfecto de G es un subconjunto D de vértices de V tal que todo vértice que no está en D es adyacente a exactamente un vértices en D. El costo de un conjunto dominante perfecto D incluye todos los costos de los vértices en D y el costo de $c(u,v)$ si v no pertenece a D, u

pertenece a D y (u,v) es una arista en E. El problema de dominación perfecta consiste en encontrar un conjunto dominante perfecto con un costo mínimo.

La tarea fundamental al aplicar la estrategia de programación dinámica consiste en fusionar dos soluciones factibles en una nueva solución factible.

Visualizaciones:

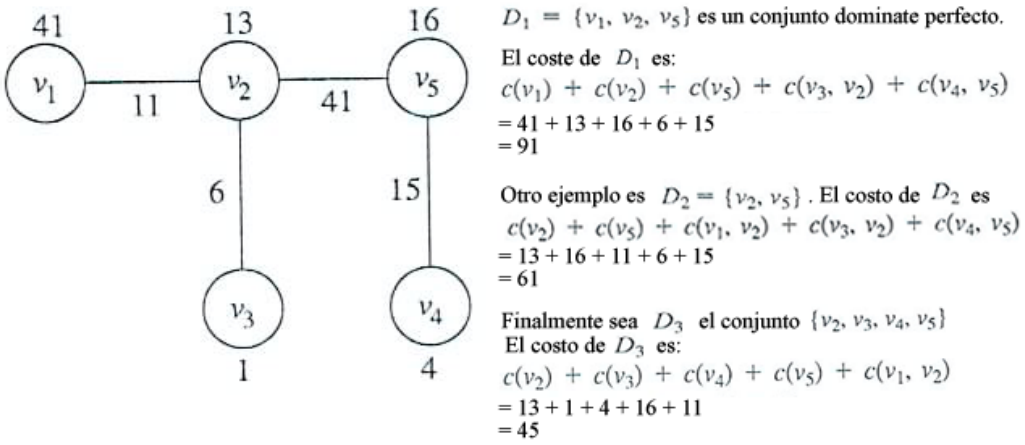


Ilustración 79 P8-Gráfica que ilustra el problema.

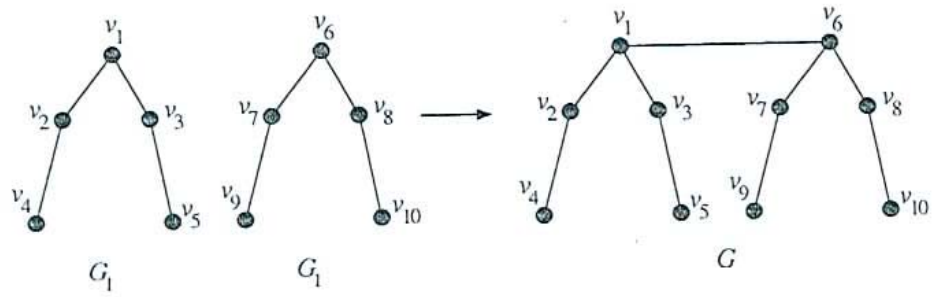


Ilustración 80 P8-Ejemplo que ilustra el esquema de fusionar al resolver el problema ponderado de dominación perfecta.

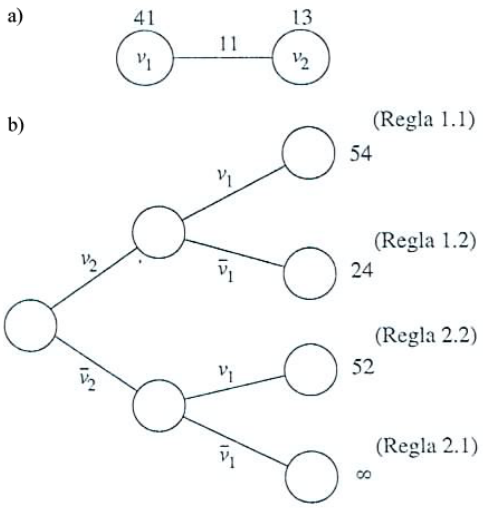


Ilustración 81 P8-Subárbol que contiene a v1 y v2 (a). Cálculo del conjunto dominante perfecto del subárbol que contiene a v1 y v2 (b).

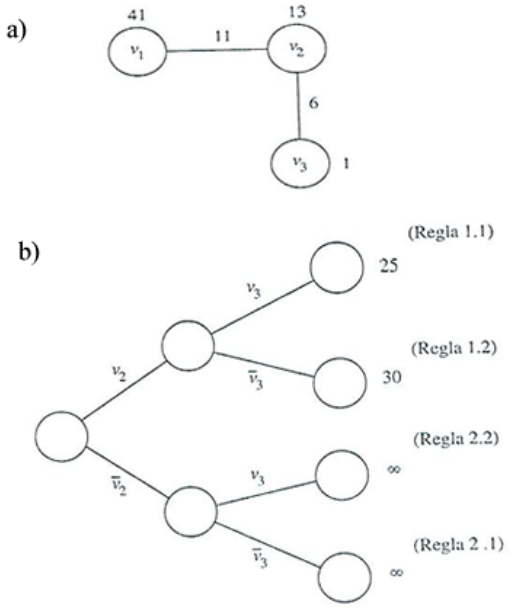


Ilustración 82 P8-Subárbol que contiene a v1, v2 y v3 (a). Cálculo del conjunto dominante perfecto del subárbol que contiene a v1, v2 y v3 (b).

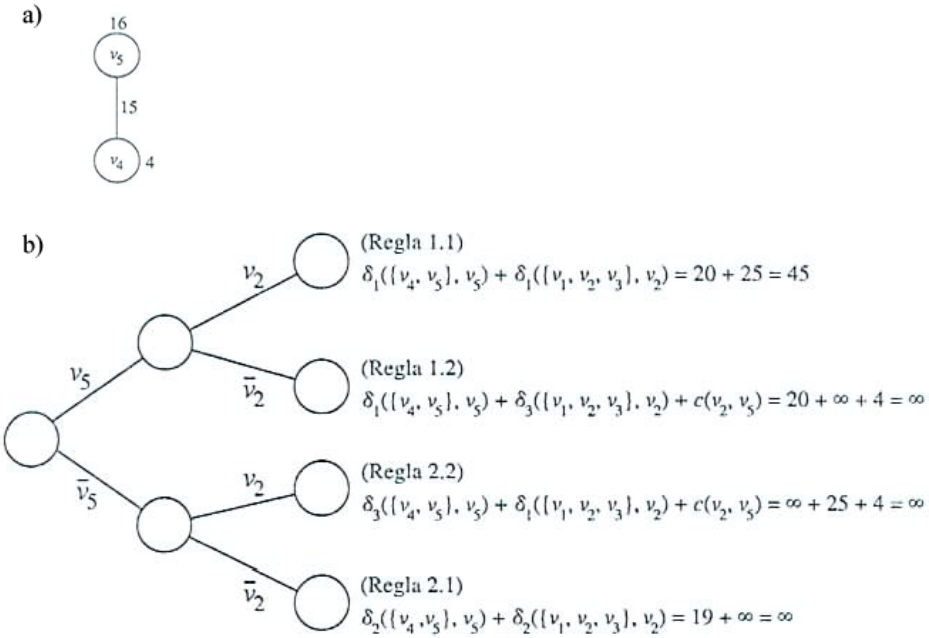


Ilustración 83 P8-Subárbol que contiene a v_4 y v_5 (a). Cálculo del conjunto dominante perfecto de todo el árbol de la *Ilustración 63* (b).

3.5 Problemas de Grafos

A continuación se recogen los problemas encontrados sobre grafos. Se han clasificado en grafos multietapa o multiestado (*multistage graphs*) y sobre subconjunto de redes no cruzadas.

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 2 (pág. 203)	Grafo 5 estados. (Fig. 5.1 pág. 203) Grafo 4 estados - problema de 3 proyectos. (Fig. 5.2 pág. 204)	Pseudocódigo – aproximación hacia delante. (pág. 206) Pseudocódigo – aproximación hacia atrás. (pág. 207)	*Multistage Graphs*
P2	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.5 (pág. 825)	Ejemplo - redes. (Fig. 20.8 pág. 825) Ejemplo - MNS. (Fig. 20.9 pág. 826) Tabla figura 20.8. (Fig. 20.10 pág. 828)	Código Java – iterativo. (pág. 829) Código Java – iterativo MNS. (pág. 830)	*Noncrossing Subset of Nets*
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 10 (pág. 426)	NO	Pseudocódigo (pág. 428)	*Puentes - Islas*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 11 (pág. 428)	NO	Pseudocódigo (pág. 429/30)	*Planificación viaje - canoas*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 8 (pág. 301)	Ejemplos búsquedas (Figs. 7.34/44 págs. 301/9)	NO	*El problema ponderado de búsqueda de bordes en una gráfica en un solo paso*
P6	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 9 (pág. 309)	Ilustraciones (Figs. 7.45/49 págs. 310/14)	NO	*El problema de rutas de m-Vigilantes para polígonos de 1 espiral resuelto para pd*

3.5.1 Multistage Graphs [P1]

Problema:

A multistage graph $G = (V,E)$ is a directed graph in which the vertices are partitioned into $k \geq 2$ disjoint sets V_i, $1 \leq i \leq k$. In addition, if $<u,v>$ is an edge in E then $u \in V_i$ and $v \in V_{i+1}$ for some i, $1 \leq i < k$. The sets V_1 and V_k are such that $|V_1| = |V_k| = 1$. Let s and t respectively be the vertex in V_1 and V_k. S is the source and t the sink. Let $c(i, j)$ be the cost of edge $<i,j>$. The cost of a path from s to t is the sum of the cost of the edges on the path. The multistage graph problem is to find a minimum cost path from s to t. Each set V_i defines a stage in the graph. Because of the constraints on E, every path from s to t starts in stage 1, goes to stage 2, then to stage 3, then to stage 4 etc. and eventually terminates in stage k.

Visualizaciones:

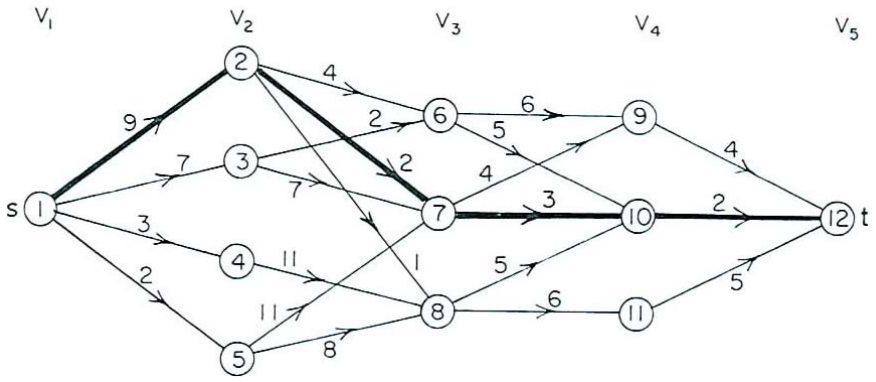


Ilustración 84 P1-Grafo con 5 estados. El coste mínimo del camino s->t esta indicado con los arcos más oscuros.

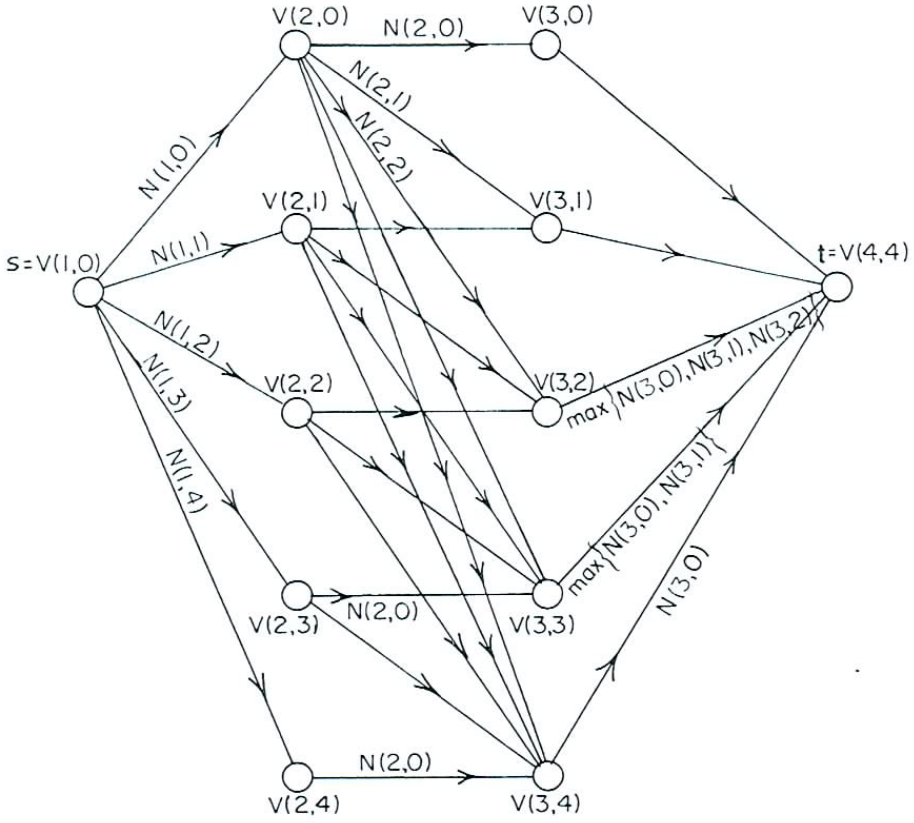


Ilustración 85 P1-Grafo de 4 estados para el problema de 3 proyectos

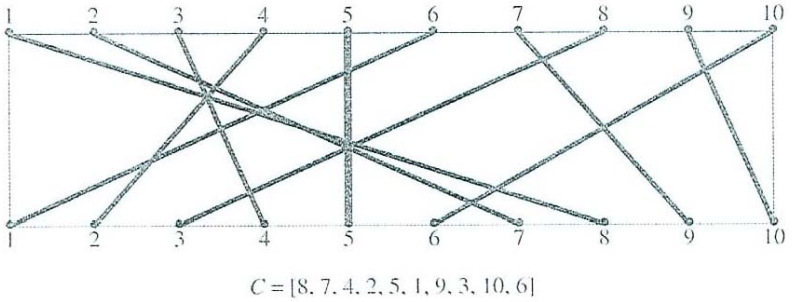
3.5.2 Noncrossing Subset of Nets [P2]

Problema:

We are given a routing channel with n pins on either side and a permutation C. Pin i on the top side of the channel is to be connected to pin C_i on the bottom side, $1 \leq i \leq n$. The pair (i, C_i) is called a net.

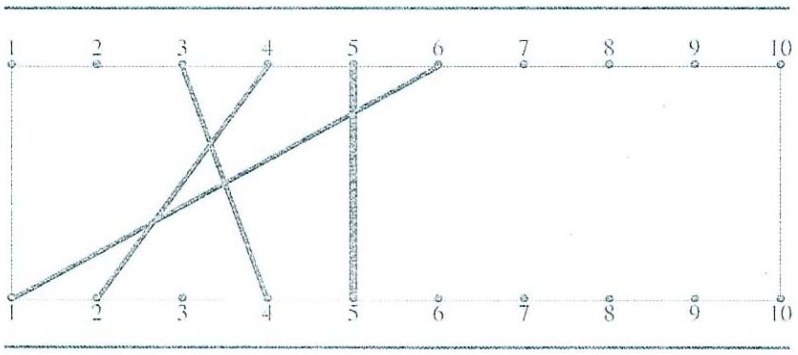
In all we have n nets that are to be connected or routed. Suppose that we have two or more routing layers of which one is a preferred layer. For example, in the preferred layer it may be possible to use much thinner wires, or the resistance in the preferred layer may be considerably less than in other layers. Our task is to route as many nets as possible in the preferred layer. The remaining nets will be routed, at least partially, in the other layers. Since two nets can be routed in the same layer if they do not cross, our task is equivalent to finding a maximum noncrossing subset (MSS) of the nets. Such a subset has the property that no two nets of the subset cross. Since net (i, c_i) is completely specified by i, we may refer to this net as net i.

Visualizaciones:



The nets (1, 8) and (2, 7) (or equivalently, the nets 1and 2) cross and so cannot be routed in the same layer. The nets (1, 8), (7, 9) and (9, 10) do not cross and so can be routed in the same layer. These three nets do not constitute an MNS, as a larger subset of noncrossing nets exists.

Ilustración 86 P2-Ejemplo de redes



The set of four nets (4, 2), (5, 5), (7, 9), (9, 10) is an MNS of routing instance given in Ilustración 87.

Ilustración 87 P2-Redes de la Ilustración 86 que pueden pertenecer a MNS[7,6]

i	j									
	1	2	3	4	5	6	7	8	9	10
1	0	0	0	0	0	0	0	1	1	1
2	0	0	0	0	0	0	1	1	1	1
3	0	0	0	1	1	1	1	1	1	1
4	0	1	1	1	1	1	1	1	1	1
5	0	1	1	1	2	2	2	2	2	2
6	1	1	1	1	2	2	2	2	2	2
7	1	1	1	1	2	2	2	2	3	3
8	1	1	2	2	2	2	2	2	3	3
9	1	1	2	2	2	2	2	2	3	4
10	1	1	2	2	2	3	3	3	3	4

Ilustración 88 P2-Tabla *size(i, j)* de la Ilustración 70

```
public static void mns(int [] theC, int [][] size)
{
   int numberOfNets = theC.length - 1;
   // initialize size[1][*]
   for (int j = 0; j < theC[1]; j++)
      size[1][j] = 0;
   for (int j = theC[1]; j <= numberOfNets; j++)
      size[1][j] = 1;

   // compute size[i][*], 1 < i < numberOfNets
   for (int i = 2; i < numberOfNets; i++)
   {
      for (int j = 0; j < theC[i]; j++)
         size[i][j] = size[i - 1][j];
      for (int j = theC[i]; j <= numberOfNets; j++)
         size[i][j] = Math.max(size[i - 1][j],
                               size[i - 1][theC[i] - 1] + 1);
   }

   size[numberOfNets][numberOfNets] =
       Math.max(size[numberOfNets - 1][numberOfNets],
                size[numberOfNets - 1][theC[numberOfNets] - 1] + 1);
}
```

Ilustración 89 P2-Programa mns, calcula *size*(*i, j*) para todo *i, j*

3.5.3 Puentes - Islas [P3]

Problema:
Un archipiélago consta de unas cuantas islas y varios puentes que unen ciertos pares de islas.

Para casa puente (que puede ser de dirección única), además de saber la isla origen y la isla destino, se conoce su anchura>0. La anchura de un camino formado por una sucesión de puentes es la anchura mínima entre las anchuras de todos los puentes que lo forman. Para cada par de islas de desea saber cuál es el camino de anchura máxima que las une (siempre que exista alguno).

Visualizaciones:

```
fun puentes(anchura[1..n, 1..n] de real∞) dev (A[1..n, 1..n] de real∞, camino[1..n, 1..n] de 0..n)
    A := anchura
    camino[1..n, 1..n] := [0]
    para k = 1 hasta n hacer
        para i = 1 hasta n hacer
            para j = 1 hasta n hacer
                temp := min(A[i, k], A[k, j])
                si temp > A[i, j] entonces { es mejor pasar por k }
                    A[i, j] := temp ; camino[i, j] := k
                fsi
            fpara
        fpara
    fpara
ffun
```

Ilustración 90 P3-Pseudocódigo puentes – islas.

3.5.4 Planificación Viaje – Canoas [P4]

Problema:
Indian Pérez quiere planificar una aventurilla por el Amazonas. A lo largo del río hay *n* poblados indígenas, cuyos habitantes, al observar el creciente auge del turismo rural, han ideado un sistema de alquiler de canoas. En cada poblado se puede alquilar una canoa, la cual puede devolverse en cualquier otro poblado que esté a favor de la corriente. Consultando por Internet los costes de alquileres entre poblados, Indiana ha constatado que el coste del alquiler desde un poblado *i* hasta otro *j* puede resultar mayor que el coste total de una serie de alquileres más breves. En tal caso, es más rentable devolver la primera canoa en alguna aldea *k* entre *i* y *j*, y seguir camino en una segunda canoa, sin cargos adicionales por cambiar de canoa.

(a) Dar un algoritmo eficiente para determinar el coste mínimo de un viaje en canoa desde todos los posibles puntos de partida *i* hasta todos los posibles puntos de llegada *j*.
(b) Modificar el algoritmo de apartado anterior para que, además, proporciona un plan de alquileres para viajar desde el primer poblado (el más cercano al nacimiento del río) hasta el último (el más cercano a la desembocadura).

Visualizaciones:

```
fun canoas1(alquiler[1..n, 1..n] de real+) dev coste[1..n, 1..n] de real+
    { inicialización }
    coste := alquiler
    para d = 2 hasta n − 1 hacer { recorrer diagonales }
        para i = 1 hasta n − d hacer { recorrer elementos de la diagonal }
            j := i + d
            { cálculo del mínimo }
            para k = i + 1 hasta j − 1 hacer
                coste[i, j] := min(coste[i, j], coste[i, k] + coste[k, j])
            fpara
        fpara
    fpara
ffun
```

Ilustración 91 P4-Pseudocódigo apartado a)

```
fun canoas2(alquiler[1..n, 1..n] de real+) dev ⟨coste[1..n, 1..n] de real+, camino[1..n, 1..n] de 0..n⟩
    coste := alquiler
    camino[1..n, 1..n] := [0]
    para d = 2 hasta n − 1 hacer { recorrer diagonales }
        para i = 1 hasta n − d hacer { recorrer elementos de la diagonal }
            j := i + d
            para k = i + 1 hasta j − 1 hacer
                temp := coste[i, k] + coste[k, j]

                si temp < coste[i, j] entonces
                    coste[i, j] := temp ; camino[i, j] := k
                fsi
            fpara
        fpara
    fpara
ffun

proc plan(e camino[1..n, 1..n] de 0..n, e i, j : 1..n)
    k := camino[i, j]
    si k ≠ 0 entonces
        plan(camino, i, k) ; imprimir(k) ; plan(camino, k, j)
    fsi
fproc
```

Ilustración 92 P4-Pseudocódigo apartado b)

3.5.5 El Problema Ponderado de Búsqueda de Bordes en una Gráfica en un Solo Paso [P5]

Problema:

Dado un grafo no dirigido simple $G = (V, E)$ donde cada vértice $v \in V$ está asociado con un peso $wt(v)$. Todos los bordes de G tienen la misma longitud.

Se supone que en alguna arista de G está escondido un fugitivo que puede moverse a cualquier velocidad. En cada arista de G, para buscar al fugitivo se asigna un buscador de aristas. El buscador siempre empieza desde un vértice. El costo de recorrer una arista (u,v) se define como $wt(u)$ si el buscador empieza desde u y como $wt(v)$ si empieza desde v. suponga que todos los buscadores de aristas se desplazan a la misma velocidad. El problema ponderado de búsqueda de aristas en una gráfica simple en un solo paso consiste en disponer las direcciones de búsqueda de los buscadores de aristas de modo que el fugitivo sea atrapado en un paso y se minimice el número de buscadores utilizados.

Visualizaciones:

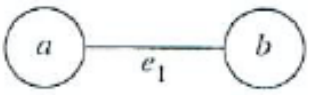


Ilustración 93 P5-Caso en el que no se requieren buscadores adicionales.

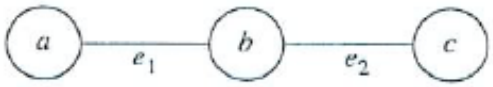
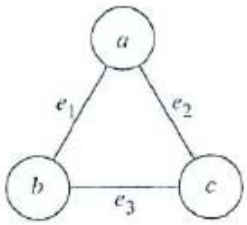


Ilustración 94 P5-Otro caso en el que no se requieren buscadores adicionales.



1. El buscador en la arista e_1 busca de a a b.
2. El buscador en la arista e_2 busca de a a c.
3. El buscador en la arista e_3 busca de b a c.

Ilustración 95 P5-Caso en el que se requieren buscadores adicionales.

v_i es un nodo hoja. Entonces $T(v_i)$ denota a v_i y a su nodo padre.

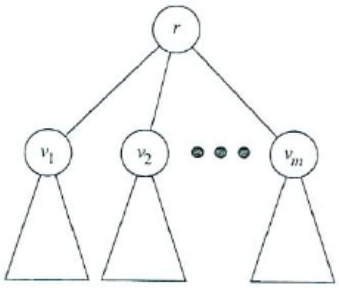
Ilustración 96 P5-Definición de $T(v_i)$ - caso1.

v_i es un nodo interno, pero no el nodo raíz. Así, (v_i) denota al árbol que implica a v_i, al nodo padre v_j y a todos los nodos descendientes de v_i.

Ilustración 97 P5-Definición de $T(v_i)$ - caso2.

v_i es la raíz de un árbol T. Entonces $T(v_i)$ denota a T.

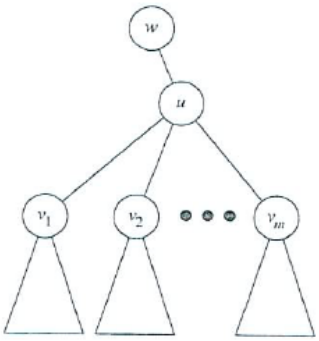
Ilustración 98 P5-Definición de $T(v_i)$ - caso3.



$$C(T(r), \bar{r}) = \min\left\{\sum_{i=1}^{m} C(T(v_i), r, v_i), \sum_{i=1}^{m} C(T(v_i), v_i, r)\right\}$$

$$C(T(r), r) = wt(r) + \sum_{i=1}^{m} \min\{C(T(v_i), r, v_i), C(T(v_i), v_i, r)\}.$$

Ilustración 99 P5-Ilustración Regla2



$$C(T(u), w, u) = \min\{C(T(u), w, u, \bar{u}), C(T(u), w, u, u)\},$$
$$C(T(u), u, w) = \min\{C(T(u), u, w, \bar{u}), C(T(u), u, w, u)\}.$$

Ilustración 100 P5-Ilustración Regla3

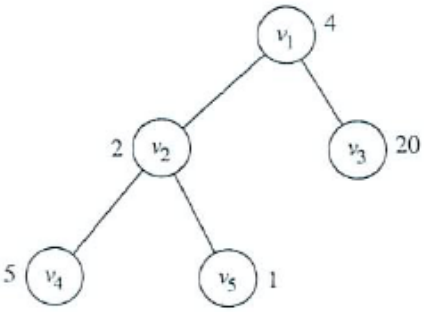


Ilustración 101 P5-Árbol que ilustra el método de programación dinámica.

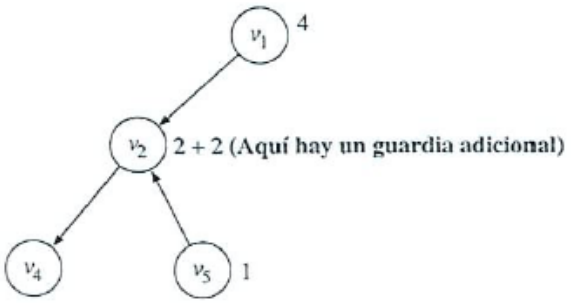


Ilustración 102 P5-Plan de búsqueda en un sólo paso que implica a v_2

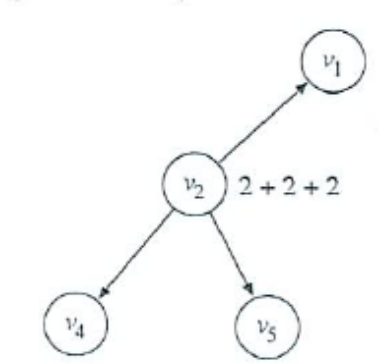


Ilustración 103 P5-Otro plan de búsqueda en un sólo paso que implica a v_2

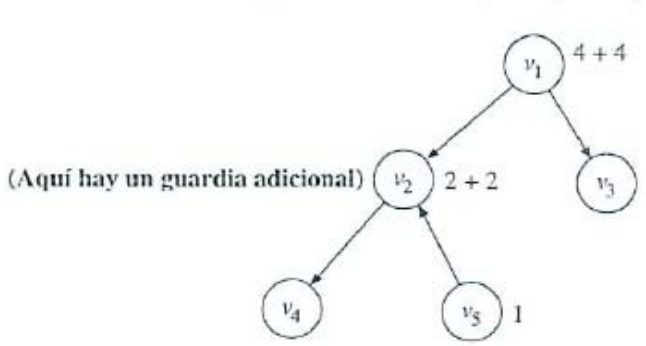


Ilustración 104 P5-Plan de búsqueda en un sólo paso que implica a v_1

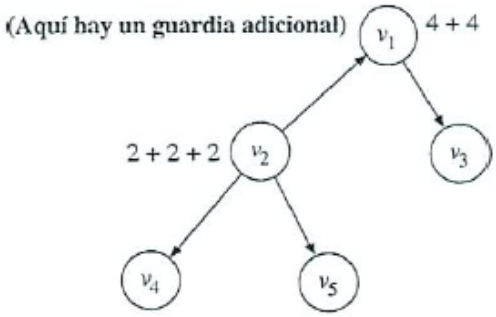


Ilustración 105 P5-Otro plan de búsqueda en un sólo paso que implica a v_1

3.5.6. El Problema de Rutas de m-Vigilantes para Polígonos de 1 Espiral Resuelto con el Método de Programación Dinámica [P6]

Problema:

Se tiene un polígono simple y un entero $m \leq 1$ y se requiere encontrar las rutas para *m*-vigilantes, denominadas las rutas de *m*-vigilantes, de modo que todo punto del polígono sea visto al menos por un vigilante desde alguna posición en su ruta. El objetivo es minimizar la suma de las longitudes de las rutas. Se ha demostrado que el problema de rutas para *m*-vigilantes para polígonos de 1 espiral es NP-difícil.

El problema de rutas de *m*-vigilantes para polígonos de 1 espiral puede resolverse con el método de programación dinámica.

Visualizaciones:

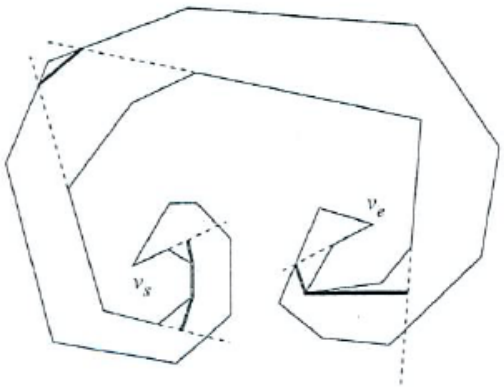


Ilustración 106 P6-Una solución del problema de rutas de 3 vigilantes para un polígono de 1 espiral.

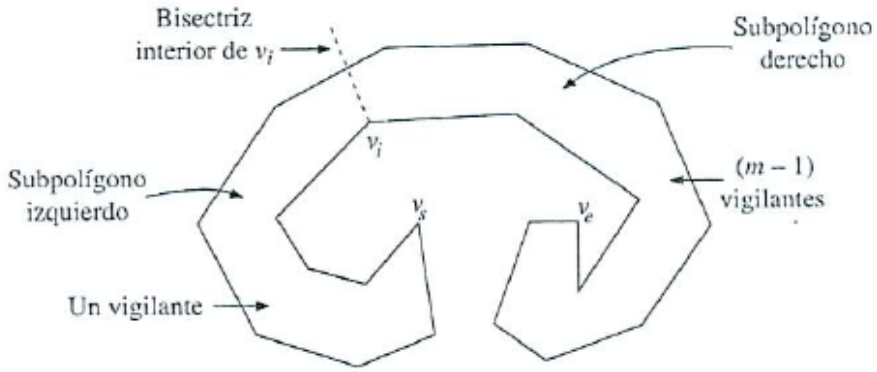


Ilustración 107 P6-Idea básica para resolver el problema de rutas de *m*-vigilantes.

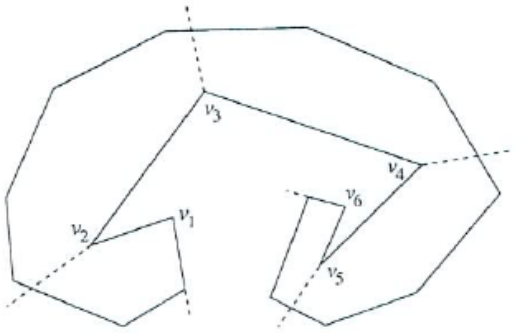


Ilustración 108 P6-Polígono de 1 espiral con seis vértices en la cadena reflex.

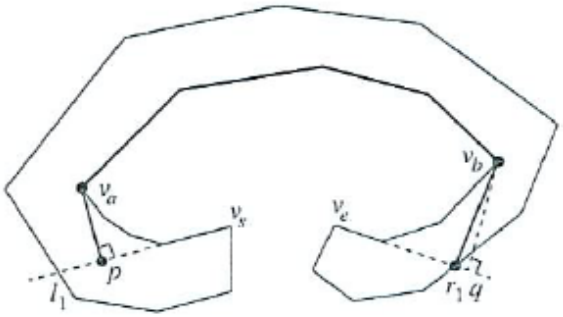
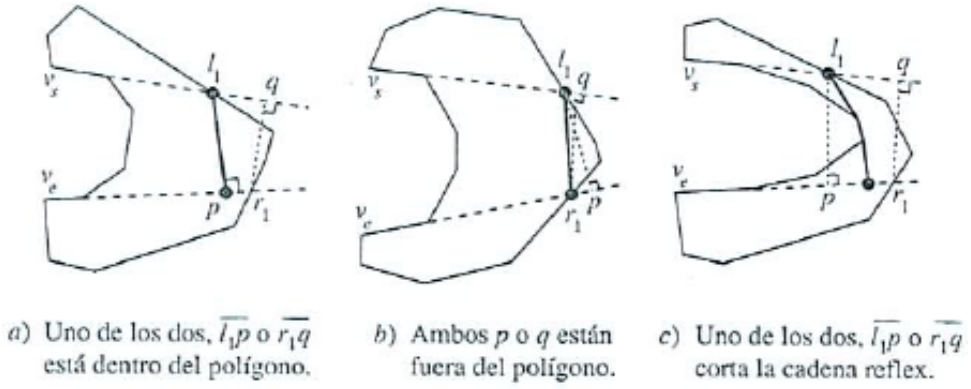


Ilustración 109 P6- Problema típico de ruta de 1 vigilante p, v_a, $C[v_a,v_b]$, v_b, r_1 en un polígono de 1 espiral.



a) Uno de los dos, $\overline{l_1 p}$ o $\overline{r_1 q}$ está dentro del polígono.

b) Ambos p o q están fuera del polígono.

c) Uno de los dos, $\overline{l_1 p}$ o $\overline{r_1 q}$ corta la cadena reflex.

Ilustración 110 P6- Casos especiales del problema de ruta de 1 vigilante en un polígono de 1 espiral.

3.6 Monedas

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 2 (pág. 295)	Tabla (Fig. 8.3 pág. 296)	Pseudocódigo (pág. 297)	Devolver Cambio
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 2 (pág. 398)	Esquema de tabla (Fig. 13.2 pág. 400)	Pseudocódigo -tabla (pág. 400) Pseudocódigo – vector (págs. 401/2)	Devolver Cambio
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 12 (pág. 430)	Esquema Temporal (Fig. 13 pág. 431)	Pseudocódigo (pág. 431)	La cantidad máxima de dinero que se puede obtener haciendo inversiones adecuadas
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 3 (pág. 407)	NO	Pseudocódigo (pág. 408)	Diferentes formas de franquear una carta

3.6.1 Devolver Cambio [P1-2]

Problema:

El problema consiste en desarrollar un algoritmo para pagar una cierta cantidad a un cliente, empleando el menor número posible de monedas. El quid del método consiste en preparar una tabla que contenga resultados intermedios útiles, que serán combinados en la solución del caso que estamos considerando. Supongamos que el sistema monetario que estamos considerando tiene monedas de n denominaciones diferentes, y que una moneda de denominación i, con $1 \leq i \leq n$ tiene un valor de d_i unidades. Supondremos, como es habitual, que todos los d_i>0. Por el momento, supondremos que se tiene un suministro ilimitado de monedas de cada denominación. Por último supongamos que tenemos que dar al cliente monedas por valor de N unidades, empleando el menor número posible de monedas. Para resolver este problema, preparamos una tabla $c[1..n,0..N]$ con una fila para cada denominación posible y una columna para las cantidades que van desde 0 unidades hasta N unidades.

Visualizaciones:

a)

$$c[i, j] = \min(c[i-1, j], 1+c[i, j-d_i])$$

b)

Cantidad:	0	1	2	3	4	5	6	7	8
$d_1 = 1$	0	1	2	3	4	5	6	7	8
$d_2 = 4$	0	1	2	3	1	2	3	4	2
$d_3 = 6$	0	1	2	3	1	2	1	2	2

Ilustración 111 P1-Función recursiva (a) y tabla devolver cambio donde hay que pagar 8 unidades con monedas de 1, 4 y 6 unidades (b).

```
función monedas(N)
   {Devuelve el mínimo número de monedas necesarias
      para cambiar  N unidades. El vector d[1..n]
      especifica las denominaciones: en el ejemplo hay monedas
      de 1, 4 y 6 unidades}
   vector d[1..n] = [1, 4, 6]
   matriz c[1..n, O..N]
   para i ← 1 hasta n hacer c[i, 0] ← 0
   para i ← 1 hasta n hacer
      para j ← 1 hasta N hacer
            si i = 1 y  j < d[i] entonces c[i, j] ← +∞
            sino si i = 1 entonces c[i, j] ← 1+ c[1, j – d[1]]
            sino si j < d[i] entonces c[i, j] ← c[i–1, j]
            sino c[i, j] ← mín(c[i – 1, j],1+c[i, j – d[i]])
   devolver c[n, N]
```

Ilustración 112 P1-Pseudocódigo - Devolver cambio.

a) $$monedas(i, j) = \min(\underbrace{monedas(i-1, j)}_{\text{no cogemos moneda } m_i}, \underbrace{monedas(i, j-m_i)+1}_{\text{sí cogemos moneda } m_i})$$

b)

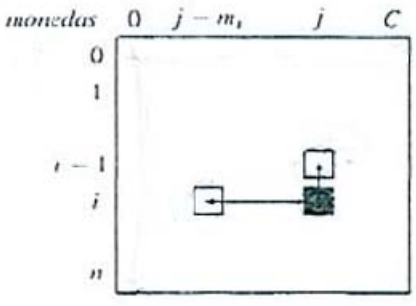
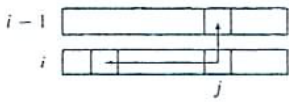


Ilustración 113 P2-Función recursiva (a) y esquema de tabla (b).



Para cada posición sólo se necesita un valor de la fila anterior *i*-1 (en la misma columna) y un valor de la misma fila.

Ilustración 114 P2-Empleando un vector en vez de la matriz.

```
fun devolver-cambio2(M[1..n] de nat+, C : nat) dev número : nat∞
var monedas[0..C] de nat∞
    { inicialización }
    monedas[0] := 0
    monedas[1..C] := [+∞]
    { hacer las actualizaciones }
    para i = 1 hasta n hacer
        para j = M[i] hasta C hacer
            monedas[j] := min(monedas[j], monedas[j − M[i]] + 1)
        fpara
    fpara
    número := monedas[C]
ffun
```

Ilustración 115 P2-Pseudocódigo empleando un vector

3.6.2 La Cantidad Máxima de Dinero que se Puede Obtener Haciendo Inversiones Adecuadas [P3]

Problema:

Mr. Scrooge dispone de una cierta cantidad de dinero M que quiere invertir durante *n* meses. Al principio de cada mes puede elegir una de entre las tres opciones siguientes, destinando a ella todo su dinero disponible en ese momento:

1. Comprar certificados de depósito de un mes del Banco Usureros & Co, cuya comisión fija (no depende de la cantidad invertida) en el tiempo *t* de compra es GCD (*t*), es decir, una cantidad de dinero *x* invertida en el tiempo *t* se convierte en la cantidad (*x*-GCD(*t*))*RCD(*t*) en el tiempo *t*+1.
2. Comprar bonos del tesoro de Corruplandia de seis meses. Los gastos de compra en el tiempo *t* son GBT(*t*) (también fijos) y el correspondiente rendimiento a los seis meses es RBT(*t*).
3. Guardar el dinero en un calcetín y ponerlo debajo del colchón (durante un mes).

Suponiendo que Mr.Scrooge tiene predicciones fiables de GCD; RCD, GBT y RBT para los *n* meses siguientes, desarrollar un algoritmo eficiente para calcular la cantidad máxima de dinero que puede obtener haciendo las inversiones adecuadas.

Visualizaciones:

a)

$inversiones(n)$ = *máxima* cantidad obtenible después de invertir n meses, es decir, en el instante n.

$$inversiones(i) = \begin{cases} \text{máx}\{inversiones(i-1), \\ \quad (inversiones(i-1) - GCD[i-1]) * RCD[i-1]\} & \text{si } 1 \leq i < 6 \\ \text{máx}\{inversiones(i-1), \\ \quad (inversiones(i-1) - GCD[i-1]) * RCD[i-1], \\ \quad (inversiones(i-6) - GBT[i-6]) * RBT[i-6]\} & \text{si } i \geq 6 \end{cases}$$

b)

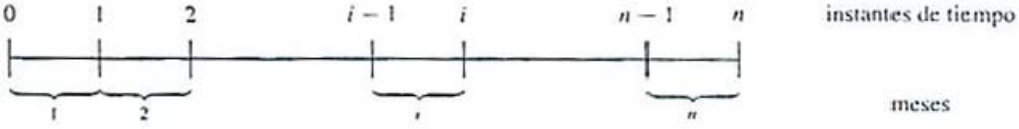


Ilustración 116 P3-Función recursiva (a) y esquema temporal (b).

```
fun inversión(M : real+, GCD[0..n − 1], RCD[0..n − 1], GBT[0..n − 1], RBT[0..n − 1] de real+)
              dev máx-gan : real+
var inversiones[0..n]
    inversiones[0] := M     { empezamos con una cantidad M }
    para i = 1 hasta mín(5, n) hacer
        inversiones[i] := máx(inversiones[i − 1],
                              (inversiones[i − 1] − GCD[i − 1]) * RCD[i − 1])
    fpara
    para i = 6 hasta n hacer
        inversiones[i] := máx(inversiones[i − 1],
                              máx((inversiones[i − 1] − GCD[i − 1]) * RCD[i − 1],
                                  (inversiones[i − 6] − GBT[i − 6]) * RBT[i − 6]))
    fpara
    máx-gan := inversiones[n]
ffun
```

Ilustración 117 P3-Pseudocódigo inversiones.

3.6.3 Diferentes Formas de Franquear una Carta [P4]

Problema:

El país de Fanfanisflán emite n sellos diferentes de valores naturales positivos s_1, s_2,..., s_n. Se quiere enviar una carta y se sabe que la correspondiente tarifa postal es T.

¿De cuántas formas diferentes se puede franquear exactamente la carta, si el orden de los sellos no importa?

Visualizaciones:

a)

$$formas(n, T) = \text{número de formas de franquear } T \text{ con } n \text{ tipos de sellos.}$$

$$formas(i, j) = \begin{cases} formas(i-1, j) & \text{si } s_i > j \\ formas(i-1, j) + formas(i, j-s_i) & \text{si } s_i \leq j \end{cases}$$

donde $1 \leq i \leq n$ y $1 \leq j \leq T$.

b)

```
fun sellos(S[1..n] de nat+, T : nat+) dev núm-formas : nat
var formas[0..T] de nat
    { inicialización }
    formas[0] := 1
    formas[1..T] := [0]
    { actualizaciones del vector }
    para i = 1 hasta n hacer
        para j = S[i] hasta T hacer
            formas[j] := formas[j] + formas[j - S[i]]
        fpara
    fpara
    núm-formas := formas[T]
ffun
```

Ilustración 118 P4-Fórmula recursiva (a) y pseudocódigo (b).

3.6.4 Otros Problemas

Problem 416
[Parberry, 1995, p.96, dynamic programming]

Arbitrage is the use of discrepancies in currency-exchange rates to make a profit. For example, there may be a small window of time during which 1 U.S. dollar buys 0.75 British pounds, 1 British pound buys 2 Australian dollars, and 1 Australian dollar buys 0.70 U.S. dollars. Then, a smart trader can trade one U.S. dollar and end up with $0.75 \times 2 \times 0.7 = 1.05$ U.S. dollars, a profit of 5%. Suppose that there are n currencies c_1, …, c_n, and an $n \times n$ table R of exchange rates, such that one unit of currency c_i buys $R[i, j]$ units of currency c_j. Devise and analyze a dynamic programming algorithm to determine the maximum value of $R[c_1, c_{i1}] \cdot R[c_{i1}, c_{i2}] \cdot \cdot \cdot R[c_{ik-1}, c_{ik}] \cdot R[c_{ik}, c_1]$.

Problem 417
[Parberry, 1995, p.97, dynamic programming]

You have \$1 and want to invest it for n months. At the beginning of each month, you must choose from the following three options:

(a) Purchase a savings certificate from the local bank. Your money will be tied up for one month. If you buy it at time t, there will be a fee of $CS(t)$ and after a month, it will return $S(t)$ for every dollar invested. That is, if you have \$$k$ at time t, then you will have \$$(k - CS(t))S(t)$ at time $t + 1$.

(b) Purchase a state treasury bond. Your money will be tied up for six months. If you buy it at time t, there will be a fee of $CB(t)$ and after six months, it will return.

3.7 Multiplicación de Matrices

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 3 (pág. 208)	Tabla (Fig. 7.2 pág. 211)	Pseudocódigo (pág. 212)	*Matrix Chain Multiplication*
P2	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 3 (pág. 457)	Árbol ejemplo (Fig. 10.2 pág. 464)	Pseudocódigo (pág. 462) Pseudocódigo-orden óptimo (pág. 465)	*Multiplying a Sequence of Matrices*
P3	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 6 (pág. 304)	Tabla (Fig. 8.6 pág. 308)	Pseudocódigo – recursión y memorización (pág. 312)	*Multiplicación encadenada de matrices*
P4	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Capítulo 15 Apartado 2 (pág. 370)	Tabla (Fig. 15.5 pág. 376) Árbol de recursión. (Fig. 15.7 pág. 385)	Pseudocódigo - sin pd. (pág. 371) Pseudocódigo - ordenación sin pd. (pág. 375) Pseudocódigo – paréntesis óptimos(pág. 377) Pseudocódigo – recursivo (pág. 385) Pseudocódigo - memoización (pág. 388)	*Matrix chain multiplication*
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 6 (pág. 411)	Esquema tabla (Fig. 13.4 pág. 412) Recorrido tabla (Fig. 13.5 pág. 413)	Pseudocódigo (pág. 413-14)	*Producto de matrices*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Capítulo 3 Apartado 4 (pág. 105)	Tabla ejemplo (Fig. 3.8 pág. 108) Tabla – p ejemplo (Fig. 3.9 pág. 112)	Pseudocódigo (págs. 110/1) Pseudocódigo - Orden (págs. 112/3)	*Chained Matrix Multiplication*
P7	Ian Parberry. ***Problems on Algorithms***	Capítulo 8 Apartado 1 (pág. 87)	NO	Pseudocódigo (pág. 87)	*Iterated Matrix Product*
P8	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.2 (pág. 807)	NO	Pseudocódigo - recursivo (pág. 811) Pseudocódigo - recursivo con tabla (pág. 813) Pseudocódigo – iterativo (pág. 814)	*Matrix Multiplication Chains*
P9	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.1 (pág. 499)	Modo en el que se emplean los elementos de la matriz para rellenar el array N (Fig. 12.9 pág. 502)	Pseudocódigo (pág. 501)	*Matrix-chain-product*

3.7.1 Multiplicación de Matrices [P1-9]

Problema:

[N. Martí Oliet, Y. Ortega y J. A. Verdejo. Estructuras de datos y métodos algorítmicos: ejercicios resueltos, pág. 411]

El producto de una matriz A_{pxq} y una matriz B_{qxr} es una matriz C_{pxr} cuyos elementos son:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Por tanto, se necesitan *pqr* multiplicaciones entre escalares ara calcular *C*. Si ahora se quiere multiplicar una secuencia de matrices $M_1 M_2 \ldots M_n$, donde cada matriz M_i tiene dimensiones $d_{i-1} \times d_j$, el orden de las matrices no se puede alterar, pero sí el de los productos a realizar, ya que la multiplicación de matrices es asociativa. Desarrollar un algoritmo que inserte paréntesis en la secuencia de matrices de forma que el número total de multiplicaciones entre escalares sea mínimo.

Visualizaciones:

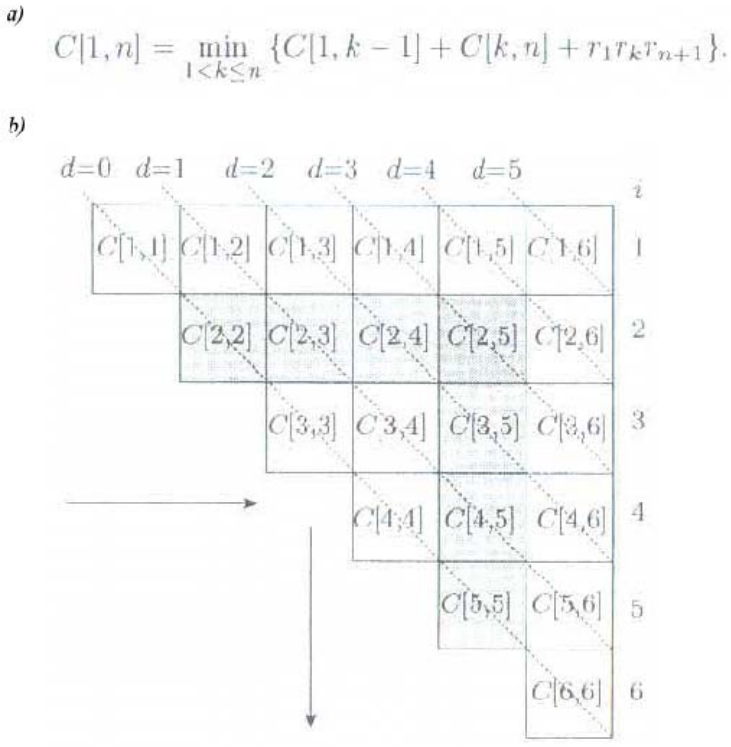


Ilustración 119 P1 -Fórmula recursiva (a) y tabla de multiplicación de matrices (b).

a) $M_1 : 5 \times 10, \quad M_2 : 10 \times 4, \quad M_3 : 4 \times 6, \quad M_4 : 6 \times 10, \quad M_5 : 10 \times 2.$

b)

$C[1,1] = 0$	$C[1,2] = 200$	$C[1,3] = 320$	$C[1,4] = 620$	$C[1,5] = 348$
	$C[2,2] = 0$	$C[2,3] = 240$	$C[2,4] = 640$	$C[2,5] = 248$
		$C[3,3] = 0$	$C[3,4] = 240$	$C[3,5] = 168$
			$C[4,4] = 0$	$C[4,5] = 120$
				$C[5,5] = 0$

Ilustración 120 P1-Ejemplo de multiplicación de 5 matrices (a) encadenadas, mostrando la tabla correspondiente (b).

```
1.  for i ← 1 to n        {Fill in diagonal d_0}
2.      C[i, i] ← 0
3.  end for
4.  for d ← 1 to n − 1    {Fill in diagonals d_1 to d_{n−1}}
5.      for i ← 1 to n − d    {Fill in entries in diagonal d_i}
6.          j ← i + d
7.          comment: The next three lines compute C[i, j]
8.          C[i, j] ← ∞
9.          for k ← i + 1 to j
10.             C[i, j] ← min{C[i, j], C[i, k − 1] + C[k, j] + r[i]r[k]r[j + 1]}
11.         end for
12.     end for
13. end for
14. return C[1, n]
```

Ilustración 121 P1-Pseudocódigo multiplicación de matrices encadenadas.

a)

$$A_1 \times A_2 \times A_3 \times A_4$$
$$30 \times 1 \quad 1 \times 40 \quad 40 \times 10 \quad 10 \times 25$$

b)

$$\text{cost} = \begin{bmatrix} . & 0 & 1200 & 700 & 1400 \\ . & . & 0 & 400 & 650 \\ . & . & . & 0 & 10000 \\ . & . & . & . & 0 \\ . & . & . & . & . \end{bmatrix} \qquad \text{last} = \begin{bmatrix} . & -1 & 1 & 1 & 1 \\ . & . & -1 & 2 & 3 \\ . & . & . & -1 & 3 \\ . & . & . & . & -1 \\ . & . & . & . & . \end{bmatrix}$$

c)

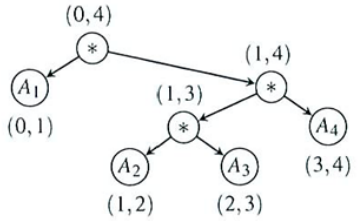


Ilustración 122 P2-Ejemplo de multiplicación de cinco matrices (a), tablas producidas según las matrices de a) y los valores d_0=30d_1=1 d_2=40 d_3=10 y d_4=25 aplicando el algoritmo matrixOrder. (c) Árbol de expresión aritmétrica correspondiente.

a)

$s = 0$: $m_{ii} = 0$ $\quad i = 1, 2, \ldots, n$

$s = 1$: $m_{i,i+1} = d_{i-1} d_i d_{i+1}$ $\quad i = 1, 2, \ldots, n-1$

$1 < s < n$: $m_{i,i+s} = \min_{i \le k < i+s} (m_{ik} + m_{k+1,\,i+s} + d_{i-1} d_k d_{i+s})$ $\quad i = 1, 2, \ldots, n-s$

b)

i \ j	1	2	3	4	
1	0	5.785	1.530	2.856	$s = 3$
2		0	1.335	1.845	$s = 2$
3			0	9.078	$s = 1$
4				0	$s = 0$

Ilustración 123 P3-Función recursiva (a) y tabla de ejemplo (b).

---

a)

matriz	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

b)

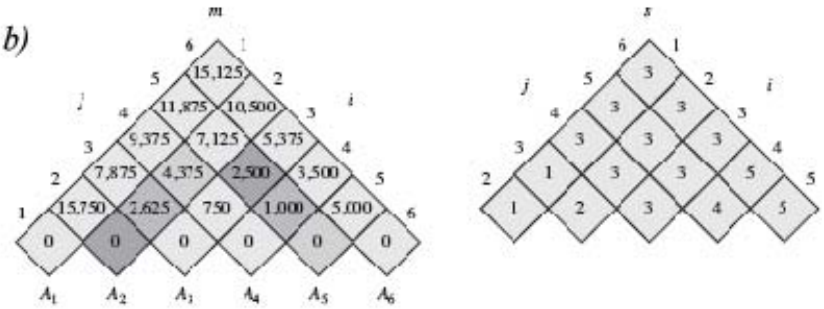


Ilustración 124 P4-Tablas m y s (b) obtenidas de ejecutar el código de Matrix-Chain-Order para n=6 y las dimensiones de las matrices mostradas en a).

---

a)

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\,, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\,. \end{cases}$$

b)

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$
$$= 7125\,.$$

Ilustración 125 P4-Función recursiva (a) y cálculo de m[2,5] para los valores de la Ilustración 124.

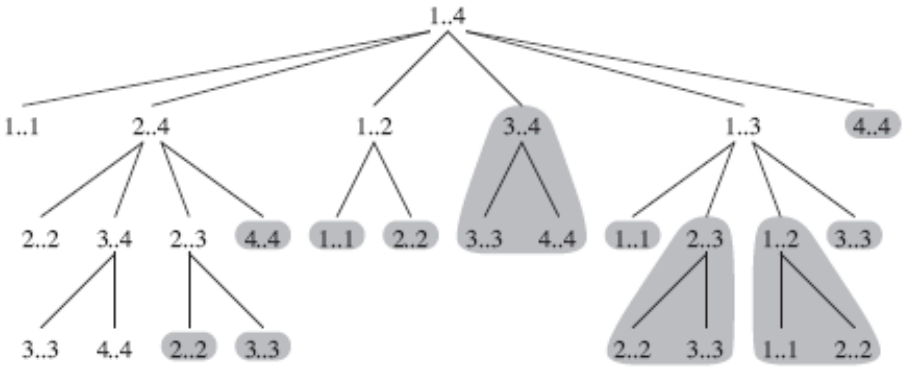


Ilustración 126 P4 -Árbol de recursión del cómputo de Recursive-Matrix-Chain (p, 1, 4).

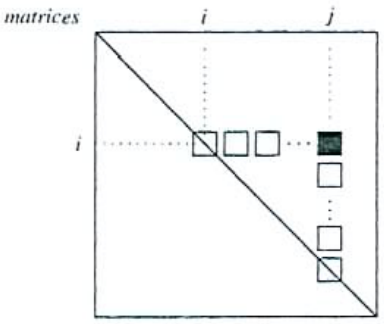


Ilustración 127 P5 -Esquema de tabla.

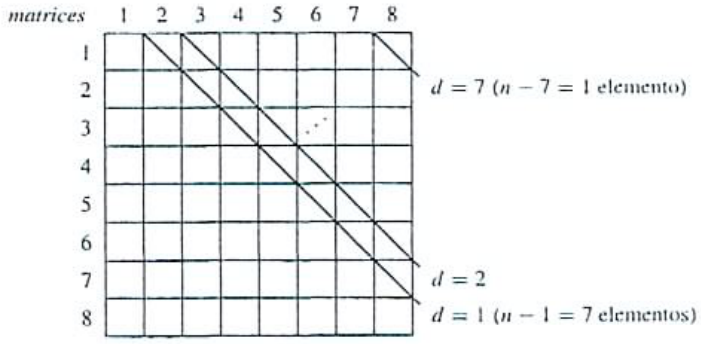


Ilustración 128 P5 -Cómo recorrer una tabla por diagonales.

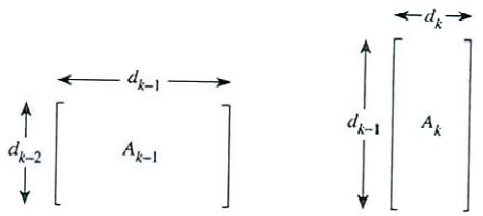
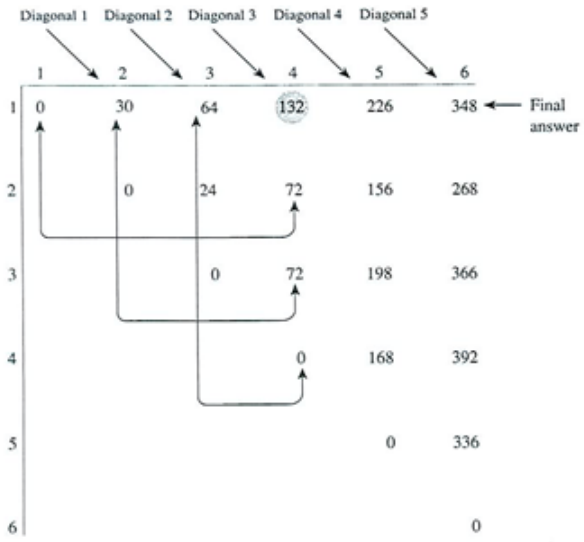


Ilustración 129 P6 -Número de columnas en A_{k-1} es el mismo número de filas en A_k



Compute diagonal 0:

$M[i, i] = 0 \quad \text{for } 1 \le i \le 6.$

Compute diagonal 1:

$$M[1, 2] = \underset{1 \le k \le 1}{minimum}(M[1, k] + M[k + 1, 2] + d_0 d_k d_2)$$
$$= M[1, 1] + M[2, 2] + d_0 d_1 d_2$$
$$= 0 + 0 + 5 \times 2 \times 3 = 30$$

Compute diagonal 2:

$$M[1, 3] = \underset{1 \le k \le 2}{minimum}(M[1, k] + M[k + 1, 3] + d_0 d_k d_3)$$
$$= minimum(M[1, 1] + M[2, 3] + d_0 d_1 d_3,$$
$$M[1, 2] + M[3, 3] + d_0 d_2 d_3)$$
$$= minimum(0 + 24 + 5 \times 2 \times 4, 30 + 0 + 5 \times 3 \times 4) =$$

Compute diagonal 3:

$$M[1, 4] = \underset{1 \le k \le 3}{minimum}(M[1, k] + M[k + 1, 4] + d_0 d_k d_4)$$
$$= minimum(M[1, 1] + M[2, 4] + d_0 d_1 d_4,$$
$$M[1, 2] + M[3, 4] + d_0 d_2 d_4,$$
$$M[1, 3] + M[4, 4] + d_0 d_3 d_4)$$
$$= minimum(0 + 72 + 5 \times 2 \times 6, 30 + 72 + 5 \times 3 \times$$
$$64 + 0 + 5 \times 4 \times 6) = 132$$

. . .

Compute diagonal 5:

$M[1, 6] = 348.$

Ilustración 130 P6 -Cálculo del valor de M[1,4], se le ha hecho un círculo en la tabla, obtenido del cómputo del resto de datos indicados con las flechas.

---

a) $A_1 \times A_2 \times A_3 \times A_4 \times A_5 \times A_6$

$5 \times 2 \quad 2 \times 3 \quad 3 \times 4 \quad 4 \times 6 \quad 6 \times 7 \quad 7 \times 8$

$d_0 \; d_1 \quad d_1 \; d_2 \quad d_2 \; d_3 \quad d_3 \; d_4 \quad d_4 \; d_5 \quad d_5 \; d_6$

b)

P	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5

Ilustración 131 P6-Array P (b) obtenido tras aplicar Minimuns-Multiplications con los datos de a).

Algorithm MatrixChain($d_0, \ldots, d_n$):

Input: Sequence $d_0, \ldots, d_n$ of integers

Output: For $i, j = 0, \ldots, n-1$, the minimum number of multiplications $N_{i,j}$ needed to compute the product $A_i \cdot A_{i+1} \cdots A_j$, where A_k is a $d_k \times d_{k+1}$ matrix

for $i \leftarrow 0$ to $n-1$ **do**
  $N_{i,i} \leftarrow 0$
for $b \leftarrow 1$ to $n-1$ **do**
  for $i \leftarrow 0$ to $n-b-1$ **do**
    $j \leftarrow i+b$
    $N_{i,j} \leftarrow +\infty$
    for $k \leftarrow i$ to $j-1$ **do**
      $N_{i,j} \leftarrow \min\{N_{i,j}, N_{i,k} + N_{k+1,j} + d_i d_{k+1} d_{j+1}\}$.

Ilustración 132 P9-Pseudocódigo

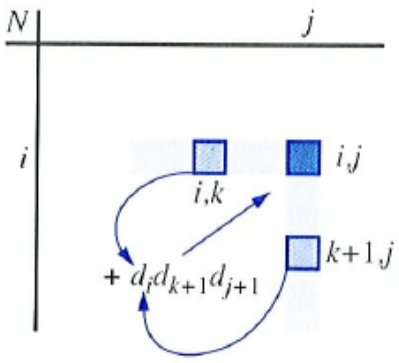


Ilustración 133 P9-Modo en el que en el producto de matrices se rellena el array N.

3.8 Planificación

En este apartado se recogen diferentes tipos de problemas que tratan todos ellos sobre la planificación de recursos. Entre los problemas más conocidos están el problema de la mochila, planificación de tareas…

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 6 (pág. 217)	Tabla ejemplo (Fig. 7.5 pág. 219)	Pseudocódigo (pág. 218)	*The Knapsack Problem*
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 4 (pág. 299)	Tabla ejemplo (Fig. 8.4 pág. 300)	NO	*El problema de la mochila*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 5 (pág. 219)	Ejemplo valores mochila (Fig. 5.11 pág. 221)	Pseudocódigo - algoritmo informal (pág. 223) Pseudocódigo. (pág. 225)	*0/1 knapsack*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 2 (pág. 402)	Esquema de tabla. (Fig. 13.3 pág. 404)	Pseudocódigo - matriz. (pág. 404) Pseudocódigo – riquezas ilimitadas (págs. 405/6) Pseudocódigo - pesos reales (pág. 407)	*El problema de la mochila(Ali Babá y los cuarenta ladrones)*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 5 (pág. 282)	Método (Fig. 7.18 pág. 283)	NO	*Problema 0/1 de la mochila*
P6	R. Sedgewick. ***Algorithms in Java***	Capítulo 5. Apartado 3.5 (pág. 219)	Ejemplo (Fig. 5.16 pág. 223) Estructura recursiva (Fig. 5.17 pág. 224) Estructura pd top-down (Fig. 5.18 pág. 226)	Pseudocódig - recursivo (pág. 223) Pseudocódig - pd y recursivo (pág. 225)	*Knapsack Problem*
P7	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.1 (pág. 801)	Árbol recursivo (Fig. 20.1 pág. 803) Tabla – ejemplo (Fig. 20.2 pág. 804)	Pseudocódigo – recursivo (pág. 802) Pseudocódigo - recursivo con tabla (pág. 804) Pseudocódigo – iterativo (pág. 805) Pseudocódigo – iterativa computación de x (pág. 806)	*Knapsack Problem*
P8	R. Neapolitan y K. Naimipou. ***Foundations of Algorithms***	Capítulo 4 Apartado 4.4 (pág. 169)	Elementos y solución voraz y óptima (Fig. 4.10 pág. 167)	NO	*A Dynamic Programming Approach to the 0-1 Knapsack Problem*
P9	I. Parberry. ***Problems on Algorithms***	Capítulo 8 Apartado 2 (pág. 89)	NO	Pseudocódigo (pág. 89)	*The knapsack problem*
P10	M.T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.4 (pág. 507)	Ejemplo de llenado de la mochila (Fíg. 12.12 pág. 508)	Pseudocódigo (pág. 509)	*The* 0-1 *knapsack problem*
P11	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to***	Capítulo 15 Apartado 1	Ejemplo tabla - longitud / precio	Pseudocódigo - recursivo top-down no pd (pág. 363)	*Rod cutting*

	Algorithms	(pág. 360)	(Fig. 15.1 pág. 360) Ejemplo diferentes formas de cortar barra (Fig. 15.2 pág. 361) Árbol de recursión (Fig. 15.3 pág. 364)	Pseudocódigo memorización top-down (págs. 365/6) Pseudocódigo – memorización bottom-up (pág. 366) Pseudocódigo - lista tamaño piezas memorización (pág. 369)	
P12	*E. Horowitz y S. Sahni* , ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 8 (pág. 234)	Esquema de ejemplo (Fig. 5.15/6 págs. 234/5)	NO	*Flow Shop Scheduling (jobs-processors)*
P13	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 15 (pág. 438)	Asignación de tarea – solución 1 (Fig. 13.11 pág. 440) Esquema de tabla - solución 2 (Fig. 13.12 pág. 441)	Pseudocódigo –solución 1 ineficiente (págs. 439) Pseudocódigo –solución 2 más eficiente (págs. 441/42)	*Tareas - Procesadores*
P14	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 14 (pág. 435)	Esquema de tabla – solución 1 (Fig. 13.10 pág. 436)	Pseudocódigo –solución 2 más eficiente (págs. 436/37) Pseudocódigo –solución 3 límites más sencillos (págs. 437/38)	*Programas - Cintas*
P15	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 1 (pág. 259)	Primera etapa (Fig. 7.7 pág. 260) Dos primeras etapas (Fig. 7.8 pág. 260) Gráfica multietapas (Fig. 7.9 pág. 261) Rutas más largas (Fig. 7.10/12 págs. 261/2)	NO	*Asignación - Recursos*

3.8.1 El problema de la Mochila [P1-10]

Problema:
Sean n objetos y una mochila. Cada objeto i, $1 \leq i \leq n$, tiene un peso positivo p_i. Si se introduce en la mochila, produce un valor positivo o beneficio b_i. La mochila puede llevar un peso que no sobrepase w.

Nuestro objetivo es llenar la mochila de tal manera que se maximice el beneficio producido por los objetos introducidos respetando la limitación de capacidad impuesta.

Por ejemplo, supongamos que están disponibles tres objetos, el primero de los cuales pesa 6 unidades y tiene un valor de 8, mientras que los otros dos pesan 5 unidades cada uno y tienen un valor de 5 cada uno. Si la mochila puede llevar 10 unidades, entonces la carga óptima incluye a los dos objetos más ligeros, con un valor total de 10. El algoritmo voraz, por otra parte, comenzaría por seleccionar el objeto que pesa 6 unidades, puesto que es el que tiene un mayor valor por unidad de peso.

Sin embargo, si los objetos no se pueden romper, el algoritmo no podrá utilizar la capacidad restante de la mochila. La carga que produce, por tanto, consta de un solo objeto, y tiene un valor de 8 nada más.

Existen dos variantes del problema de la mochila que son:

Problema de la mochila. Se pueden romper los objetos en trozos pequeños, de manera que podamos decidir llevar solamente una fracción x_i.

Problema de la mochila 0/1. Los objetos no se pueden fragmentar en trozos pequeños, así que podemos decidir si tomamos un objeto o lo dejamos.

Visualizaciones:

a)

$$V[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ V[i-1,j] & \text{if } j < s_i \\ \max\{V[i-1,j], V[i-1,j-s_i] + v_i\} & \text{if } i > 0 \text{ and } j \geq s_i. \end{cases}$$

b)

Knapsack of capacity : 9
Pack with items:
Sizes-> 2 3 4 5
Values->3 4 5 7
GOAL: Pack the knapsack with many items as possible in way that maximizes the total values without exceeding the knapsack capacity

c)

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	3	3	3	3	3	3	3	3
2	0	0	3	4	4	7	7	7	7	7
3	0	0	3	4	4	7	8	9	9	12
4	0	0	3	4	5	7	8	10	11	12

Ilustración 134 P1-Función recursiva (a), tabla con valores (c) según los datos del ejemplo (b).

a)

$$V[i, j] = \text{máx}(V[i-1, j], V[i-1, j-w_i] + v_i)$$

b)

Knapsack of capacity : 9
Pack with items:
Sizes-> 1 2 5 6 7
Values->1 6 18 22 28
GOAL: Pack the knapsack with many items as possible in way that maximizes the total values without exceeding the knapsack capacity

c)

Límite de peso: 0	1	2	3	4	5	6	7	8	9	10	11
$w_1 = 1, v_1 = 1$	0	1	1	1	1	1	1	1	1	1	1	1
$w_2 = 2, v_2 = 6$	0	1	6	7	7	7	7	7	7	7	7	7
$w_3 = 5, v_3 = 18$	0	1	6	7	7	18	19	24	25	25	25	25
$w_4 = 6, v_4 = 22$	0	1	6	7	7	18	22	24	28	29	29	40
$w_5 = 7, v_5 = 28$	0	1	6	7	7	18	22	28	29	34	35	40

Ilustración 135 P2-Función recursiva (a), tabla con valores (c) según los datos del ejemplo (b).

---

a)

$$f_i(X) = \max\{f_{i-1}(X), f_{i-1}(X - w_i) + p_i\}$$

b)

Knapsack of capacity : 6
Pack with items:
Sizes-> 2 3 4
Values->1 2 5
GOAL: Pack the knapsack with many items as possible in way that maximizes the total values without exceeding the knapsack capacity

c)

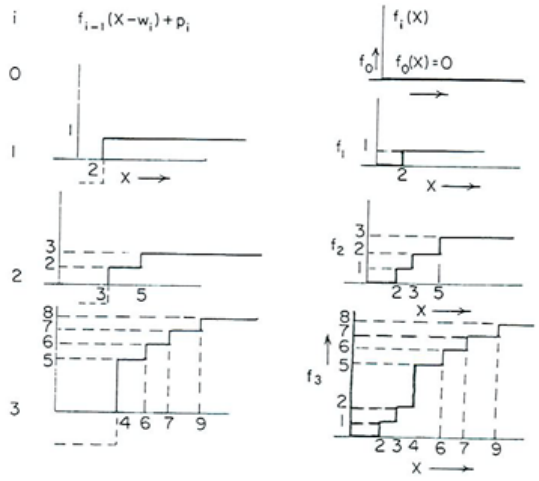


Ilustración 136 P3-Función recursiva (a), graficas que muestra los diferentes estados de la mochila (c) según los datos del ejemplo (b).

a)

$$mochila(i, j) = \text{máx}\{\underbrace{mochila(i-1, j)}_{\text{no cogemos el objeto } i}, \underbrace{mochila(i-1, j-p_i) + v_i}_{\text{sí cogemos el objeto } i}\}$$

b)

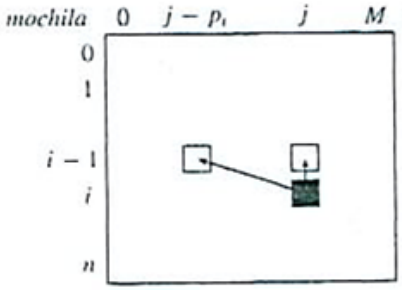


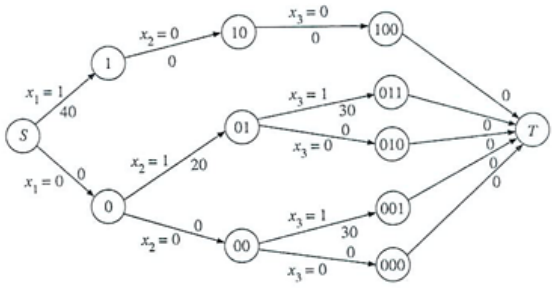
Ilustración 137 P4 -Función recursiva (a), esquema de tabla (b).

---

a)

i	W_i	P_i
1	10	40
2	3	20
3	5	30

b)



c) *La ruta más larga es:*

$S \rightarrow 0 \rightarrow 01 \rightarrow 011 \rightarrow T$

Que se corresponde con: $X_1 = 0,$ $X_2 = 1$ $X_3 = 1$

Y tiene un coste de: 20 + 30=50

Ilustración 138 P5-Método para resolver el problema b) según las datos de a) y dando como resultado c).

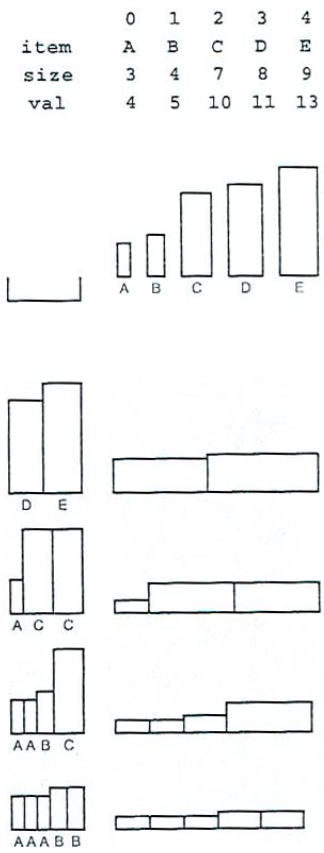


Ilustración 139 P6–Ejemplo de llenado de la mochila.

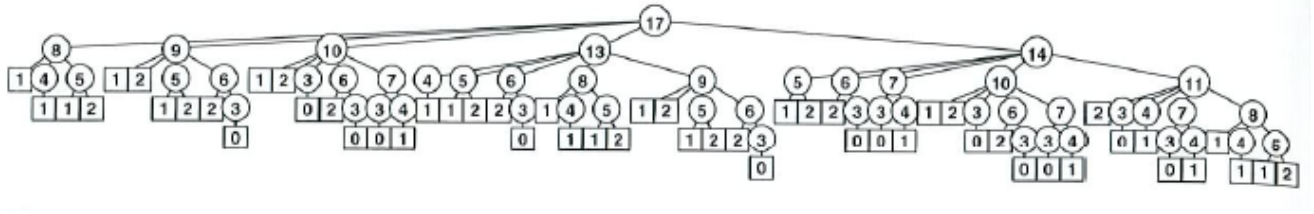


Ilustración 140 P6-Estructura recursiva obtenida al ejecutarse con el algoritmo Knapsack-Problem (sin pd).

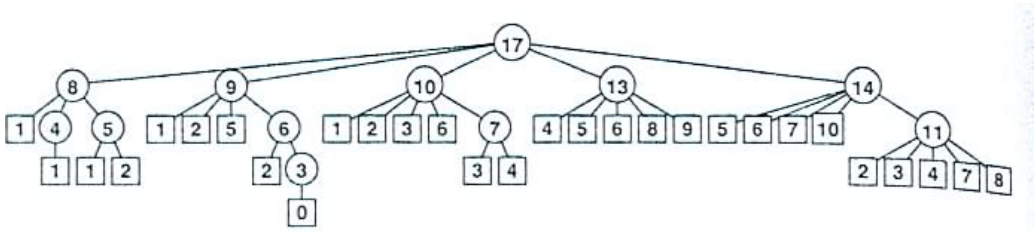


Ilustración 141 P6-Estructura recursiva obtenida al ejecutarse con el algoritmo Knapsack-Problem (con pd –top down).

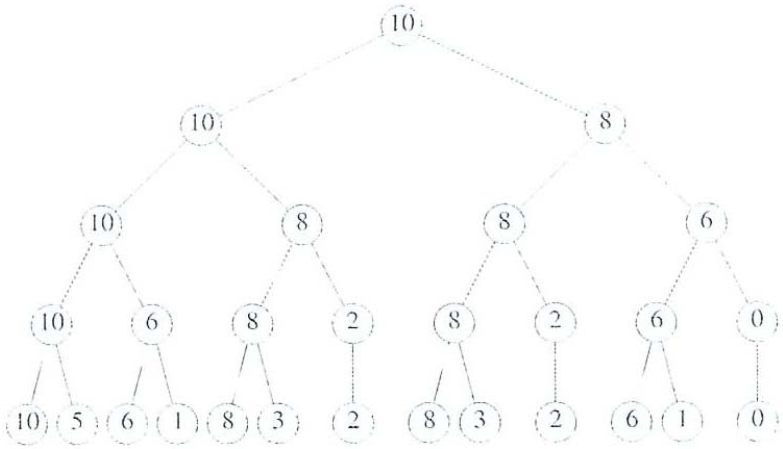


Ilustración 142 P-Árbol de recursión.

Knapsack of capacity : 10
Pack with items:
Sizes-> 6 3 5 4 6
Values->2 2 6 5 4
GOAL: Pack the knapsack with many items as possible in way that maximizes the total values without exceeding the knapsack capacity

	y										
i	0	1	2	3	4	5	6	7	8	9	10
5	0	0	0	0	6	6	6	6	6	6	6
4	0	0	0	0	6	6	6	6	6	10	10
3	0	0	0	0	6	6	6	6	6	10	11
2	0	0	3	3	6	6	9	9	9	10	11

Ilustración 143 P7-Función/array de ejemplo.

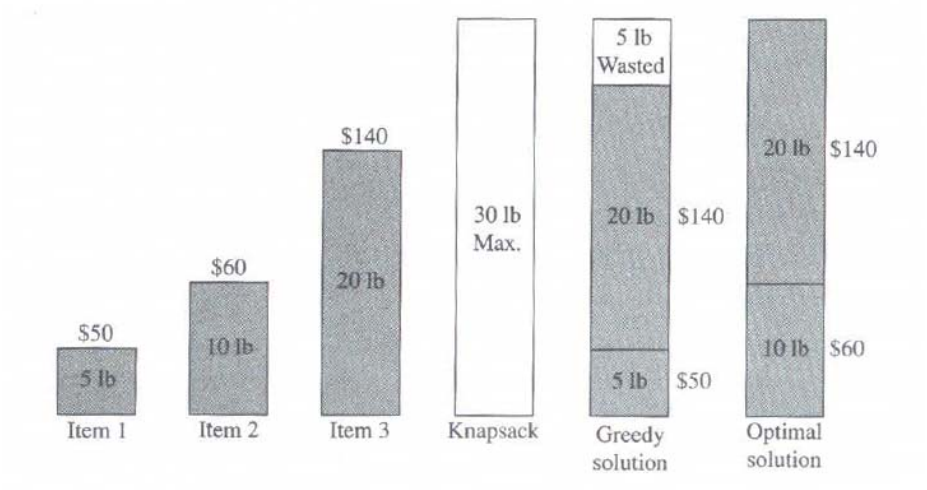


Ilustración 144 P8-Solución voraz y solución óptima del problema de la mochila

---

```
function knapsack(s_1, s_2, ..., s_n, S)
1.    t[0,0] :=true
2.    for j := 1 to S do t[0,j] :=false
3.    for i := 1 to n do
4.       for j := 0 to S do
5.          t[i,j] := t[i-1,j]
6.          if j - s_i ≥ 0 then t[i,j] := t[i,j] ∨ t[i-1, j-s_i]
7.    return(t[n,S])
```

Ilustración 145 P9–Pseudocódigo.

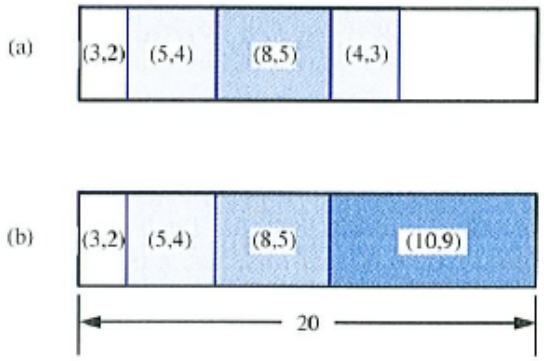


Ilustración 146 P10-Ejemplo de una aproximación para definir un problema de la mochila que no funciona: (a) mejor solución con los primeros cuatro elementos; (b) Mejor solución con los primeros cinco elementos.

Algorithm 01Knapsack$(W, w_1, \ldots, w_n, b_1, \ldots, b_n)$:

Input: Set of n items, such that item i has integer weight w_i, and real-number benefit b_i, and an integer W

Output: For $w = 0, \ldots, W$, maximum benefit $B[w]$ of a subset of S with total weight w

```
for w ← 0 to W do
  B[w] ← 0
for k ← 1 to n do
  for w ← w_k to W do
    if B[w − w_k] + b_k > B[w] then
      B[w] ← B[w − w_k] + b_k
```

Ilustración 147 P10-Pseudocódigo

3.8.2 Rod Cutting [P11]

Problema:

Serling Enterprises buys long steel rods and cuts them into shorter rods, which it then sells. Each cut is free. The management of Serling Enterprises wants to know the best way to cut up the rods.

We assume that we know, for i = 1, 2, …, the price p_i in dollars that Serling Enterprises charges for a rod of length i inches. Rod lengths are always an integral number of inches. The rod-cutting problem is the following. Given a rod of length n inches and a table of prices p_i for i = 1, 2, …, n, determine the maximum revenue r_n obtainable by cutting up the rod and selling the pieces. Note that if the price p_n for a rod of length n is large enough, an optimal solution may require no cutting at all.

We can cut up a rod of length n in 2^{n-1} different ways, since we have an independent option of cutting, or not cutting, at distance i inches from the left end.

Visualizaciones:

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Ilustración 148 **P11- Ejemplo del precio según la longitud de la barra de acero a cortar.**

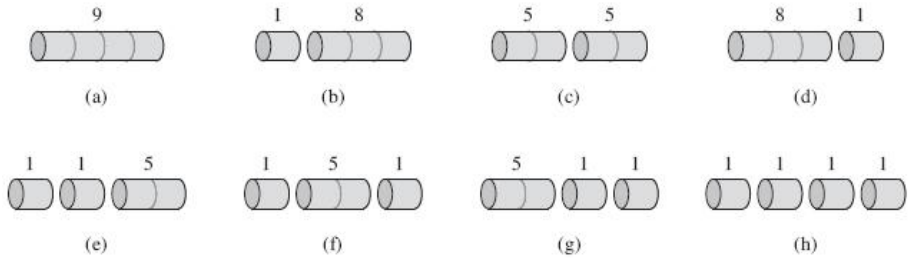


Ilustración 149 P11-Ocho posibles formas de cortar barra de acero. La óptima para longitud 2 es la c).

a)

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1)$$

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

b)

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

Ilustración 150 P11-Fórmula a) y código (recursivo y top-down) b) que implementa la función.

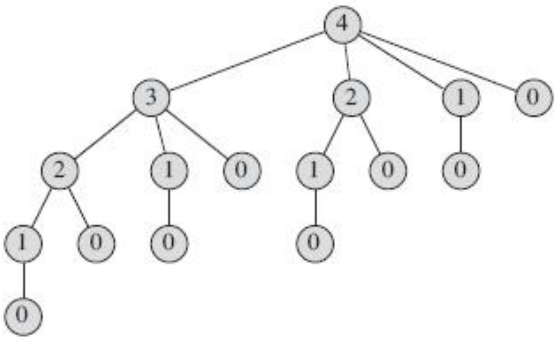


Ilustración 151 P11-Árbol de recursión resultante de CUT-ROD (p, n) n=4 de la Ilustración 150.

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

Ilustración 152 P11-Pseudocódigo con memorización y top-down.

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

Ilustración 153 P11-Pseudocódigo versión botton up.

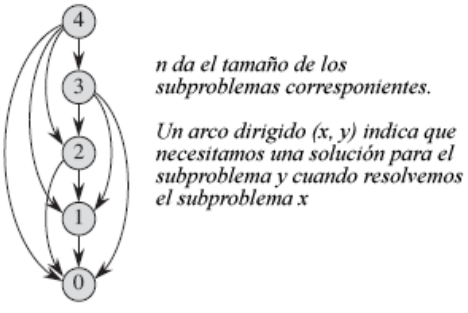


Ilustración 154 P11-Grafo del subproblema para n=4.

3.8.3 Flow Shop Scheduling (jobs-processors) [P12]

Problema:
Often, the processing of a job requires the performance of several distinct tasks. Computer programs run in a multiprogramming environment are input, then executed. Following the execution, the job is queued for output and the output eventually printed. In a general flow shop we may have n jobs each requiring m tasks T_{1i}, T_{2i},…, T_{mi}, $1 \le i \le n$ to be performed. Task T_{ji} is to be performed on processor P_j, $1 \le j \le m$, the time required to complete task T_{ji} is t_{ji}. A schedule for the n jobs is an assignment of tasks to time intervals on the processors. Task T_{ji} must be assigned to processor P_j. No processor may have more than one task assigned to it in any time interval. Additionally, for any job i the processing of task T_{ji}, $j>1$ cannot be started until task $T_{j-1,i}$ has been completed.

Visualizaciones:

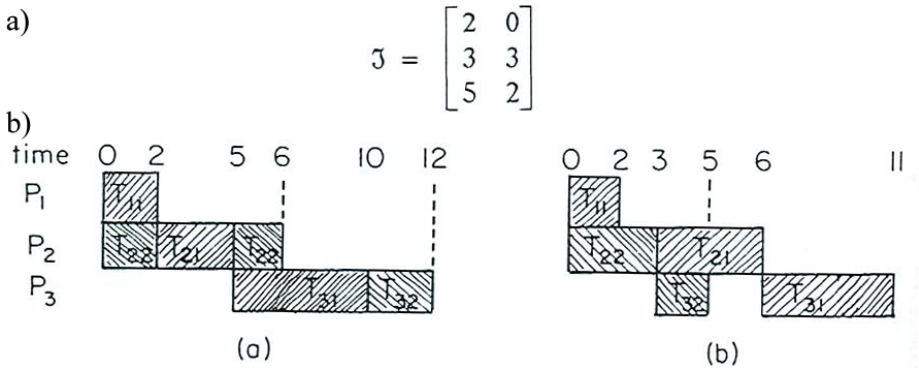


Ilustración 155 P12-Dos trabajos se planifican en dos procesadores. En la matriz a) se muestran los tiempos de las tareas y en b) dos posibles planificaciones.

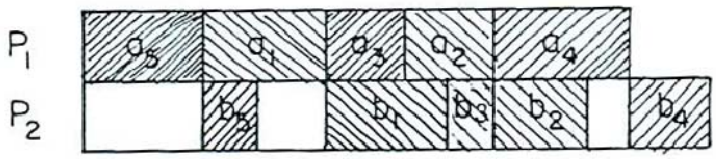


Ilustración 156 P12-Planificación de tareas para dos procesadores donde $a_i = t_{1i}$ $b_i = t_{2i}$

3.8.4 Tareas – Procesadores [P13]

Problema:
Una serie de n tareas ha de ser procesada en un sistema que cuenta con dos procesadores A y B. Para cada tarea i se conocen los tiempos a_i, b_i que cada uno de los procesadores necesita para realizarla.

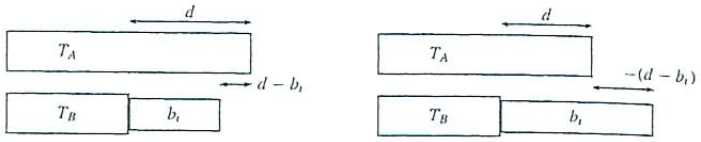
Debido a las características de los procesadores y de las tareas es posible que para una tarea i se tenga $a_i \geq b_i$ mientras que para otra tarea $j \neq i$ sea $a_i < b_i$. Nótese que una tarea no puede dividirse entre los procesadores. Obtener un procedimiento para asignar las tareas a los procesadores de forma que se minimice el tiempo necesario para terminar todas ellas.

Visualizaciones:

$tiempo(i, T_A, T_B)$ = tiempo mínimo para procesar las i primeras tareas, cuando el procesador A ya está ocupado un tiempo T_A y el procesador B un tiempo T_B.

$tiempo(i, T_A, T_B) = \min\{tiempo(i-1, T_A + a_i, T_B), tiempo(i-1, T_A, T_B + b_i)\}$,
con $1 \leq i \leq n$, $0 \leq T_A \leq \sum_{l=1}^{n} a_l$ y $0 \leq T_B \leq \sum_{l=1}^{n} b_l$

$tiempo(0, T_A, T_B) = \max\{T_A, T_B\}$.



(a) $b_i \leq d$ (b) $b_i \geq d$

Ilustración 157 P13-Solución 1 asignando una tarea al procesador B

$tiempo(i, d)$ = tiempo mínimo para procesar las i primeras tareas, partiendo del tiempo inicial $(d, 0)$ si $d \geq 0$, o del tiempo inicial $(0, -d)$ si $d < 0$.

- si $d \geq 0$

$tiempo(i, d) = \min\{tiempo(i-1, d + a_i), tiempo(i-1, d - b_i) + \min\{d, b_i\}\}$

- si $d < 0$

$tiempo(i, d) = \min\{tiempo(i-1, d + a_i) + \min\{-d, a_i\}, tiempo(i-1, d - b_i)\}$

> Caso básico, no hay más tareas que asignar:

$tiempo(0, d) = |d|$,

matriz $tiempo[0..n, -S_B..S_A]$ donde $S_A = \sum_{l=1}^{n} a_l$ y $S_B = \sum_{l=1}^{n} b_l$

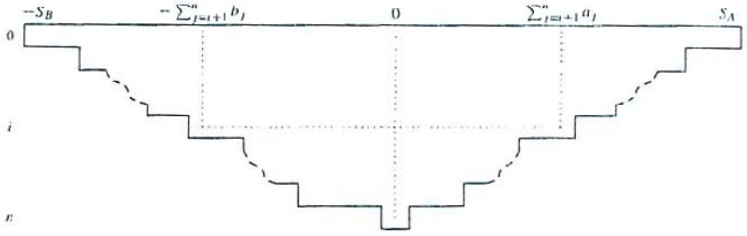


Ilustración 158 P13-Solución 2 esquema de tabla para problema de dos procesadores

3.8.5 Programas – Cintas [P14]

Problema:

Tenemos que almacenar n programas en dos cintas, cada una de longitud L, siendo l_i la longitud de cinta necesaria para almacenar el programa i. Suponemos que:

$$(\sum_{i=1}^{n} l_i \leq L)$$

Un programa puede ser almacenado en cualquiera de las dos cintas.

Si S_1 es el conjunto de programas en la cinta 1, el tiempo de acceso en el peor caso a un programa cualquiera es proporcional a:

$$\max\{\sum_{i \in S_1} l_i, \sum_{i \notin S_1} l_i\}$$

Una asignación óptima de programas a cinta minimiza el tiempo de acceso en el peor caso.

Desarrollar un algoritmo para determinar el tiempo de acceso en el peor caso de una asignación óptima y dicha asignación.

Visualizaciones:

$tiempo(i, L_1, L_2)$ = tiempo de acceso óptimo cuando faltan por asignar los programas del 1 al i, la ocupación de la primera cinta es L_1 y la de la segunda es L_2.

Caso recursivo:

$$tiempo(i, L_1, L_2) = \text{mín}\{\underbrace{tiempo(i-1, L_1 + l_i, L_2)}_{\text{programa } i \text{ en cinta 1}}, \underbrace{tiempo(i-1, L_1, L_2 + l_i)}_{\text{programa } i \text{ en cinta 2}}\}$$

con $1 \leq i \leq n$, $0 \leq L_1, L_2 \leq L'$

Caso base:

$$tiempo(0, L_1, L_2) = \text{máx}\{L_1, L_2\} \quad 0 \leq L_1, L_2 \leq L'.$$

Esquema de tabla:

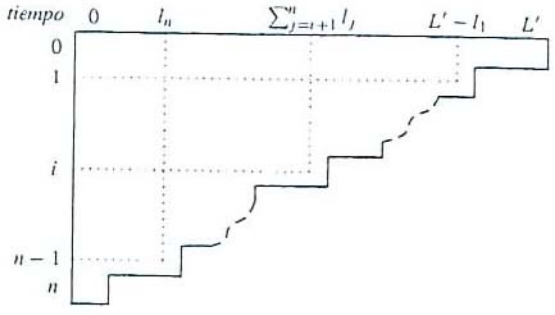


Ilustración 159 P14-Solución 1 esquema de tabla

3.8.6 El Problema de Asignación de Recursos [P15]

Problema:
Se tienen m recursos y n proyectos. Se obtiene una ganancia $P(i, j)$ si al proyecto i se asignan j, $0 \leq j \leq m$ recursos. El problema consiste en encontrar una asignación de recursos que maximice la ganancia total.

Visualizaciones:

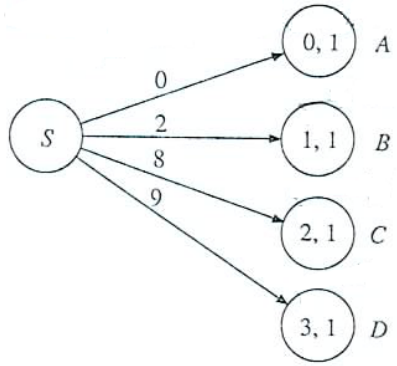


Ilustración 160 P15-Decisiones de la primera etapa.

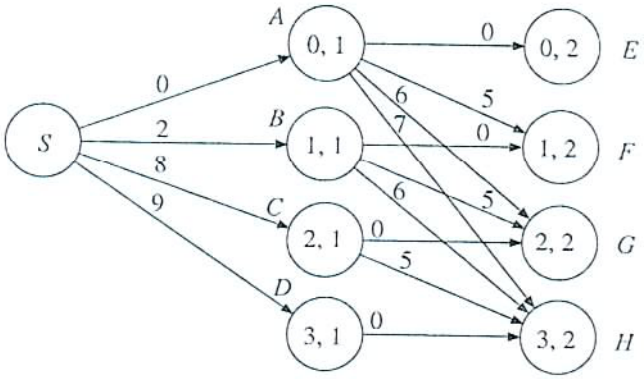


Ilustración 161 P15-Decisiones de las dos primeras etapas.

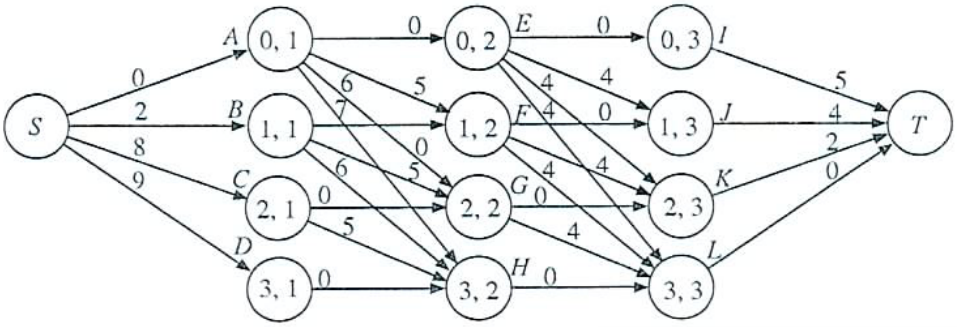


Ilustración 162 P15-Grafo Multietapa.

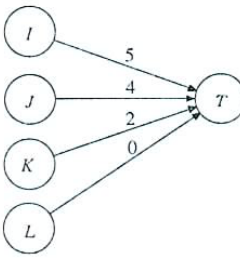


Ilustración 163 P15-Rutas más largas de I J K y L a T.

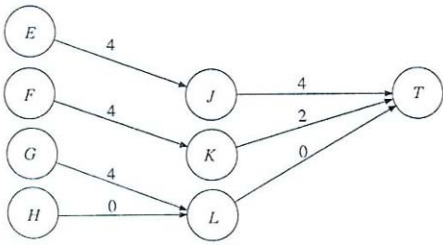


Ilustración 164 P15-Rutas más largas de E F G H a T.

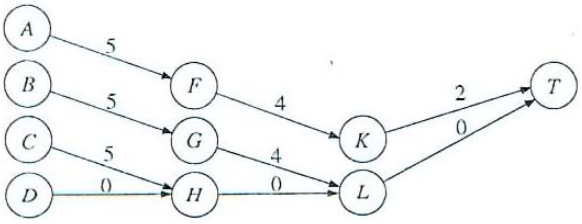


Ilustración 165 P15-Rutas más largas de A B C D a T.

3.9 Sucesión de Fibonacci

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 1 (pág. 452)	Grafo-subproblema (Fig. 10.1 pág. 454)	Pseudocódigo (pag.456)	*Fibonacci function (Introdución)*
P2	R. Sedgewick, ***Algorithms in Java***	Capítulo 5 Apartado 3 (pág. 219)	Estructura alg. recursivo (Fig. 5.14 pág. 221) Árbol pd top-down (Fig. 5.15 pág. 222)	Pseudocódigo recursivo (pág. 220) Pseudocódigo pd y recursivo (áag. 222)	*Fibonacci numbers*
P3	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado1 (pág. 273)	Árbol recursivo. (Fig. 8.1 pág. 275) Árbol recursivo - almacenamiento. (Fig. 8.2 pág. 276)	Pseudocódigo - recursivo (pág. 274) Pseudocódigo - recursivo con almacenamiento (pág. 276) Pseudocódigo – pd (pág. 277) Pseudocódigo – pd almacenando sólo los dos últimos valores (pág. 278)	*Fibonacci numbers*
P4	William McAllister, ***Data Structures and Algorithms using Java***	Capítulo 6 Apartado 4 (pág. 327)	NO	Pseudocódigo (pág. 329)	*Dynamic Programming Applied to Recursion, Fibonacci Sequence*

3.9.1 Sucesión de Fibonacci [P1-4]

Problema:
[S. Skiena, *The Algorithm Design Manual*]
The Fibonacci numbers were originally defined by the Italian mathematician Fibonacci in the thirteenth century to model the growth of rabbit populations. Rabbits breed, well, like rabbits. Fibonacci surmise that the number of pairs of rabbits born in a given year is equal to the number of pairs of rabbits born in each of the two previous years, starting from one pair of rabbits in the first year. To count the number of rabbits born in the *n*th year, he defined the recurrence relation:

$$F_n = F_{n-1} + F_{n-2}$$

With basis cases:

$$F_0 = 0 \text{ and } F_1 = 1$$

Visualizaciones:

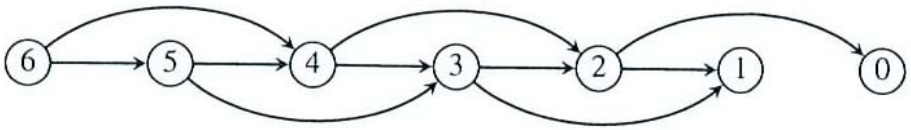


Ilustración 166 P1-Grafo del subproblema fib(6).

```
8 F(6)
  5 F(5)
    3 F(4)
      2 F(3)
        1 F(2)
          1 F(1)
          0 F(0)
        1 F(1)
      1 F(2)
        1 F(1)
          0 F(0)
    2 F(3)
      1 F(2)
        1 F(1)
        0 F(0)
      1 F(1)
  3 F(4)
    2 F(3)
      1 F(2)
        1 F(1)
        0 F(0)
      1 F(1)
    1 F(2)
      1 F(1)
        0 F(0)
```

Ilustración 167 P2-Estructura de algoritmo recursivo para los números de Fibonacci.

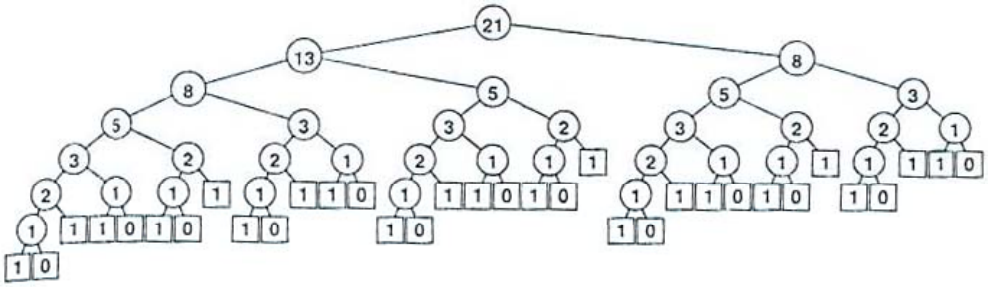


Ilustración 168 P2-Árbol de llamadas recursivas de números de Fibonacci.

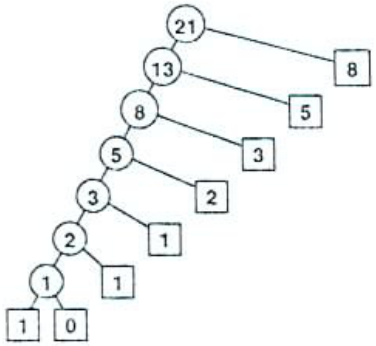


Ilustración 169 P2-Árbol llamadas de números de Fibonacci con programación dinámica top down.

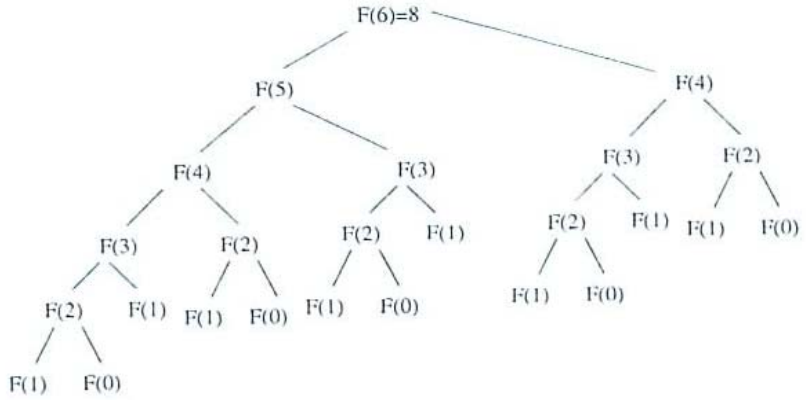


Ilustración 170 P3-Árbol recursivo para f(6).

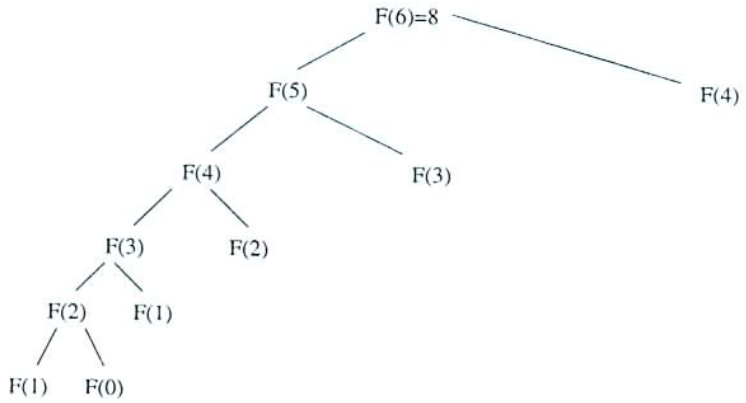


Ilustración 171 P3-Árbol de llamadas con almacenamiento de valores.

```
1.  public class MainFibonacciTerms
2.  {  public static void main(String[] args)
3.     {  int lastTerm = 45;
4.        for(int i = 1; i <= lastTerm; i++)
5.        {  System.out.println("fibonacci" + i + " " + fibonacci(i));
6.        }
7.     } // end main
8.
9.     public static long fibonacci(int n)
10.    {  if(n == 1 || n == 2) // one of 2  base cases
11.          return 1;
12.       else
13.       {  long rp1 = fibonacci(n - 1); // first reduced problem
14.          long rp2 = fibonacci(n - 2); // second reduced problem
15.          long gs = rp1 + rp2;  // general solution
16.          return gs;
17.       }
18.    } // end fibonacci method
19. } // end class MainFibonacciTerms
```

Ilustración 1728-P4 Pseudocódigo

3.10 Otros Problemas

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 1.2 (pág. 293)	Llamadas recursivas (Fig. 8.2 pág. 294)	Pseudocódigo (pág. 295)	*El Campeonato Mundial*
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 5 (pág. 410)	NO	Pseudocódigo (pág. 411)	*El Campeonato Mundial (con y sin empate)*
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 4 (pág. 408)	Vectores en el calculo de recurrencia (pág. 409)	Pseudocódigo (págs. 409/10)	*Reparto de botín*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 16 (pág. 442)	NO	Pseudocódigo (pág. 443)	*Reparto Almacenes*
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 17 (pág. 444)	Esquema de tabla (Fig. 13.13 pág. 445)	Pseudocódigo (págs. 445/6/7)	*Vacas-pienso*
P6	Verlag S. Skiena, **The *Algorithm Design Manual***	Capítulo 8 Apartado 5 (pág. 294)	Matrices con ejemplos de entrada (Fig. 8.8 pág. 297)	Pseudocódigo (pág. 296)	*The Partition Problem*
P7	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 6.1 (pág. 300)	Ejemplo triangulaciones (Fig. 8.10 pág. 300) Seleccionando vértice (Fig. 8.11 pág. 301)	Pseudocódigo (pág. 301)	*Minimum weight triangulation.*
P8	William McAllister, ***Data Structures and Algorithms using Java***	Capítulo 6 Apartado 4.1 (pág. 327)	Cambio contenido array n! (Fig. 6.14 pág. 328)	NO	*Dynamic Programming Applied to recursión, n!.*

3.10.1 El Campeonato Mundial [P1- 2]

Problema:

[G. Brassard y P. Bratley. *Fundamentos de algoritmia pág. 293*]

Considere una competición en la cual hay dos equipos A y B que juegan un máximo de $2n$-1 partidas, y en donde el ganador es el primer equipo que consiga n victorias. Suponemos que no hay empates, que los resultados de todos los partidos son independientes, y que para cualquier partido dado hay una probabilidad constante p de que el equipo A sea el ganador, y por tanto una probabilidad constante q=1-p de que gane el equipo B.

Sea $P(i, j)$ la probabilidad de que el equipo A gane el campeonato, cuando todavía necesitan i victorias más para conseguirlo, mientras que el equipo B necesita j victorias más para ganar. Por ejemplo, antes del primer partido del campeonato la probabilidad de que gane el equipo A es $P(n, n)$: ambos equipos necesitan todavía n victorias para ganar el campeonato. Si el equipo A necesita cero victorias más,

entonces lo cierto es que ya han ganado el campeonato, y por tanto $P(0,i) = 1$, con $1 \leq i \leq n$.

De manera similar, si el equipo *B* necesita 0 victorias más, entonces ya han ganado el campeonato, y por tanto $P(i,0) = 1$, con $1 \leq i \leq n$. Como no puede producirse una situación en la que ambos equipos hayan ganado todos los partidos que necesitaban, $p(0,0)$ carece de significado.

Por último dado que el equipo A gana cualquier partido con una probabilidad *p* y pierde con una probabilidad *q*:

$$P(i, j) = pP(i\text{-}1, j) + qP(i, j\text{-}1),\ i \geq 1, j \geq 1$$

Calcular $P(i, j)$.

Visualizaciones:

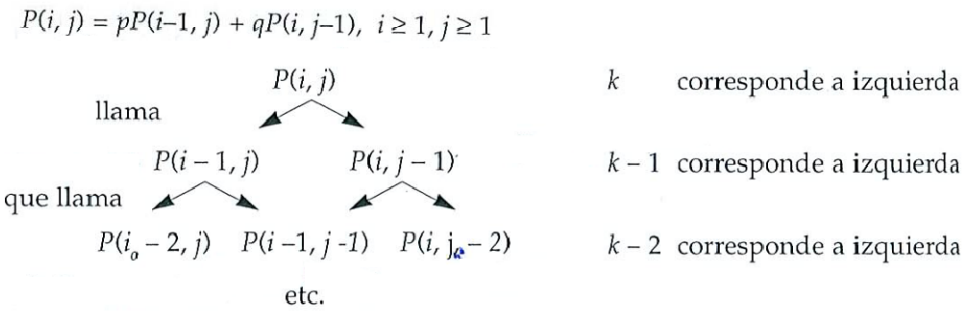


Ilustración 173 P1-Llamadas recursivas efectuadas por una llamada a $P(i, j)$.

$P(i, j)$ = probabilidad de que A gane la competición si le faltan i victorias para ganar y a B le faltan j victorias.

Caso Recursivo:

$$P(i, j) = pP(i - 1, j) + qP(i, j - 1),$$

Casos Base:

$$P(0, j) = 1 \qquad 1 \leq j \leq n,$$

$$P(i, 0) = 0 \qquad 1 \leq i \leq n,$$

Ilustración 174 P2-Funciones.

```
{ 0 ≤ p ≤ 1 }
fun competición(n : nat+, p : real) dev probabilidad-A : real
var P[0..n] : real
    P[0] := 0
    P[1..n] := [1]
    para i = 1 hasta n hacer
        para j = 1 hasta n hacer
            P[j] := p * P[j] + (1 − p) * P[j − 1]
        fpara
    fpara
    probabilidad-A := P[n]
ffun
```

Ilustración 175 P2-Pseudocódigo.

Posibilidad de empatar con probabilidad r,

$$P(i, j) = pP(i-1, j) + qP(i, j-1) + rP(i, j),$$

$$P(i, j) = \frac{p}{1-r} P(i-1, j) + \frac{q}{1-r} P(i, j-1),$$

Ilustración 176 P2-Funciones contemplando la posibilidad de empate.

3.10.2 Reparto de Botín [P3]

Problema:

El Maqui y el Popeye acaban de desvalijar la reserva nacional de oro. Los lingotes están empaquetados en *n* cajas de diferentes pesos naturales positivos p_i para *i* entre 1 y *n*. Como no tienen tiempo de desempaquetarlos para dividir el botín, deciden utilizar los pesos de cada una de las cajas para distribuir el botín a medias. Al cabo de un buen rato todavía no han conseguido repartirse el botín, por lo cual acuden al Teclas para saber si el botín se puede dividir en dos partes iguales sin desempaquetar las cajas con los lingotes.

Visualizaciones:

se-puede(i, j) = booleano que indica si es posible sumar la cantidad j eligiendo algunas de las i primeras cajas.

Caso Recursivo:

$$\textit{se-puede}(i, j) = \begin{cases} \textit{se-puede}(i-1, j) & \text{si } p_i > j \\ \textit{se-puede}(i-1, j) \vee \textit{se-puede}(i-1, j-p_i) & \text{si } p_i \leq j \end{cases}$$

donde $1 \leq i \leq n$ y $1 \leq j \leq P/2$;

Casos Base:

$$\textit{se-puede}(0, j) = \text{falso} \qquad 1 \leq j \leq P/2$$
$$\textit{se-puede}(i, 0) = \text{cierto} \qquad 0 \leq i \leq n.$$

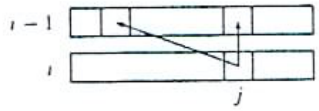


Ilustración 177 P3-Función.

```
{ P' = P/2 }
fun repartir-botin-par-pd(peso[1..n] de nat+, P' : nat+) dev respuesta : bool
var se-puede[0..P'] de bool
    { inicialización }
    se-puede[0] := cierto
    se-puede[1..P'] := [falso]
    { actualizaciones del vector }
    para i = 1 hasta n hacer
        para j = P' hasta peso[i] paso − 1 hacer
            se-puede[j] := se-puede[j] ∨ se-puede[j − peso[i]]
        fpara
    fpara
    respuesta := se-puede[P']
ffun
{ P = Σ_{i=1}^{n} peso[i] }
fun repartir-botin-pd(peso[1..n] de nat+, P : nat+) dev respuesta : bool
    si impar(P) entonces respuesta := falso
    si no respuesta := repartir-botin-par-pd(peso, P div 2)
    fsi
ffun
```

Ilustración 178 P3-Pseudocódigo.

3.10.3 Reparto Almacenes [P4]

Problema:

Se tienen dos almacenes A_1 y A_2 en los que se dispone respectivamente de a_1 y a_2 unidades (indivisibles) de un determinado producto. Desde estos dos almacenes se ha de abastecer a n comercios c_1,... ,c_n, en cada uno de los cuales se precisa una determinada cantidad c_j (para j entre 1 y n) de unidades del producto, de forma que

$$\sum_{j=1}^{n} c_j = a_1 + a_2$$

El coste del transporte desde A_i hasta c_j viene dado por una función t_{ij} (x) donde x es el número de unidades del producto transportadas.

Se desea determinar las cantidades de unidades x_{ij} que cada almacén debe enviar a cada comercio de forma que el coste total del transporte $\sum_{ij} t_{ij}$ (x_{ij}) sea mínima.

Visualizaciones:

$coste(k, x)$ = coste mínimo cuando el almacen A_1 tiene x unidades disponibles y A_2 tiene $(\sum_{j=1}^{k} c_j) - x$ para abastecer a los comercios del C_1 al C_k.

Caso Recursivo:

$$coste(k, x) = \min_{m \le j \le m'} \{coste(k-1, x-j) + t_{1k}(j) + t_{2k}(c_k - j)\}$$

$m = \max\{0, x - \sum_{l=1}^{k-1} c_l\}$ y $m' = \min\{c_k, x\}$

Casos Base:

$coste(0, x) = 0$.

Ilustración 179 P4-Función.

```
fun reparto(C[1..n] de nat+, T1[1..n, 0..a1], T2[1..n, 0..a2] de real, a1, a2 : nat+)
            dev (coste-óptimo : real, solución[1..2, 1..n] de nat)
var coste[0..n, 0..a1] de real∞, cantidad-A1[1..n, 0..a1] de nat
    { inicialización }
    coste[0, 0..a1] := [0] ; suma := 0
    { rellenamos la matriz }
    para k = 1 hasta n hacer
        { suma = Σ_{l=1}^{k-1} C[l] }
        para x = 0 hasta a1 hacer
            { cálculo del mínimo }
            coste[k, x] := +∞
            para j = máx(0, x − suma) hasta mín(C[k], x) hacer
                temp := coste[k − 1, x − j] + T1[k, j] + T2[k, C[k] − j]
                si temp < coste[k, x] entonces
                    coste[k, x] := temp ; cantidad-A1[k, x] := j
                fsi
            fpara
        fpara
        suma := suma + C[k]
    fpara
    coste-óptimo := coste[n, a1]
    { recuperar la solución }
    x := a1
    para k = n hasta 1 paso −1 hacer
        solución[1, k] := cantidad-A1[k, x]
        solución[2, k] := C[k] − solución[1, k]
        x := x − solución[1, k]
    fpara
ffun
```

Ilustración 180 P4-Pseudocódigo.

3.10.4 Pienso Vacas [P5]

Problema:

El Tío Antonio tiene en su granja dos vacas: Devoradora y Listilla. Antes de ordeñarlas las alimenta llenando una hilera de n cubos con pienso (*n* es par). Cada cubo *i* (para *i* entre 1 y *n*) contiene una cantidad p_i de pienso indicada en el cubo (todas las cantidades son distintas). Cada vaca en su turno debe elegir el cubo de uno de los extremos y comerse su contenido. El cubo se retira y el turno pasa a la otra vaca. Se sigue así hasta agotar los cubos. La vaca que comienza comiendo se determina a priori por un procedimiento cualquiera. El objetivo de ambas vacas es comer en total lo máximo posible. La estrategia de Devoradora consiste en pensar poco y escoger el cubo de los extremos que este más lleno. En cambio, Listilla

prefiere pensárselo un poco más, para lo que ha adquirido lo último en computadoras vacunas portátiles.

(a) Demostrar que la estrategia de Devoradora no es óptima, incluso cuando le toca escoger primero.
(b) Listilla ha cursado un máster en informática por la escuela superior Bovina, pero en dicha escuela no estudian la programación dinámica, por lo que pide ayuda para diseñar un algoritmo utilizando dicha técnica, suponiendo que le toca empezar a escoger.
(c) Diseñar un algoritmo de coste lineal de forma que Listilla, siempre que comience comiendo ella, coma al menos tanto como Devoradora, independientemente de la estrategia que siga esta última. ¿Es óptima la nueva estrategia?

Visualización:

$vacas(i, j)$ = cantidad *máxima* que come *Listilla* con los cubos desde el p_i hasta el p_j cuando le toca empezar a comer.

Caso Recursivo:

$$vacas(i, j) = \max\{\ p_i + \begin{cases} vacas(i+1, j-1) & \text{si } p_j > p_{i+1} \\ vacas(i+2, j) & \text{si } p_{i+1} > p_j \end{cases}$$

$$p_j + \begin{cases} vacas(i+1, j-1) & \text{si } p_i > p_{j-1} \\ vacas(i, j-2) & \text{si } p_{j-1} > p_i \end{cases}$$

Casos Base:

$d = 1$

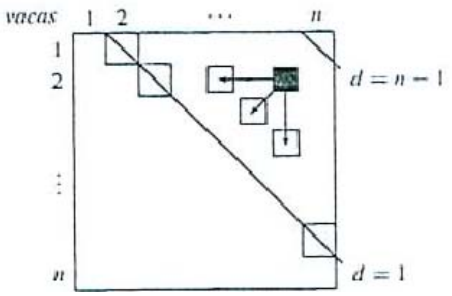


Ilustración 181 P5-Función y esquema de tabla

3.10.5 The Partition Problem [P6]

Problema:

Suppose that three workers are given the task of scanning through a shelf of books in search of a given piece of information. To get the job done fairly and efficiently, the books are to be partitioned among the three workers. To avoid the need to rearrange the books or separate them into piles, it would be simplest to divide the shelf into three regions and assign each region to one worker. But what is the fairest way to divide the shelf up?

Input: A given arrangement *S* of non-negative numbers and an integer *k*.
Output: Partition *S* into *k* ranges, so as to minimize the maximum sum over all the ranges. This so-called linear partition.

Visualizaciones:

M	*k*		
n	1	2	3
1	1	1	1
1	2	1	1
1	3	2	1
1	4	2	2
1	5	3	2
1	6	3	2
1	7	4	3
1	8	4	3
1	9	5	3

D	*k*		
n	1	2	3
1	–	–	–
1	–	1	1
1	–	1	2
1	–	2	2
1	–	2	3
1	–	3	4
1	–	3	4
1	–	4	5
1	–	4	6

M	*k*		
n	1	2	3
1	1	1	1
2	3	2	2
3	6	3	3
4	10	6	4
5	15	9	6
6	21	11	9
7	28	15	11
8	36	21	15
9	45	24	17

D	*k*		
n	1	2	3
1	–	–	–
2	–	1	1
3	–	2	2
4	–	3	3
5	–	3	4
6	–	4	5
7	–	5	6
8	–	5	6
9	–	6	7

Ilustración 182 P6-Matrices de programación dinámica M y D para dos ejemplos de entrada.

```
partition(int s[], int n, int k)
{
        int m[MAXN+1][MAXK+1];                  /* DP table for values */
        int d[MAXN+1][MAXK+1];                  /* DP table for dividers */
        int p[MAXN+1];                          /* prefix sums array */
        int cost;                               /* test split cost */
        int i,j,x;                              /* counters */

        p[0] = 0;                               /* construct prefix sums */
        for (i=1; i<=n; i++) p[i]=p[i-1]+s[i];

        for (i=1; i<=n; i++) m[i][1] = p[i];    /* initialize boundaries */
        for (j=1; j<=k; j++) m[1][j] = s[1];

        for (i=2; i<=n; i++)                    /* evaluate main recurrence */
                for (j=2; j<=k; j++) {
                        m[i][j] = MAXINT;
                        for (x=1; x<=(i-1); x++) {
                                cost = max(m[x][j-1], p[i]-p[x]);
                                if (m[i][j] > cost) {
                                        m[i][j] = cost;
                                        d[i][j] = x;
                                }
                        }
                }

        reconstruct_partition(s,d,n,k);         /* print book partition */
}
```

Ilustración 183 P6-Pseudocódigo Partición

```
reconstruct_partition(int s[],int d[MAXN+1][MAXK+1], int n, int k)
{
        if (k==1)
                print_books(s,1,n);
        else {
                reconstruct_partition(s,d,d[n][k],k-1);
                print_books(s,d[n][k]+1,n);
        }
}


print_books(int s[], int start, int end)
{
        int i;                  /* counter */

        for (i=start; i<=end; i++) printf(" %d ",s[i]);
        printf("\n");
}
```

Ilustración 184 P6-Reconstruir partición**.**

3.10.6 Minimum Weight Triangulation [P7]

Problema:
A *triangulation* of a polygon $p = \{v_1, \ldots, v_n, v_1\}$ is a set of non-intersecting diagonals that partitions the polygon into triangles. We say that the *weight* of a triangulation is the sum of the lengths of its diagonals. Any given polygon may have many different triangulations. For any given polygon, we seek to find its minimum weight triangulation. Triangulation is a fundamental component of most geometric algorithms.

Visualizaciones:

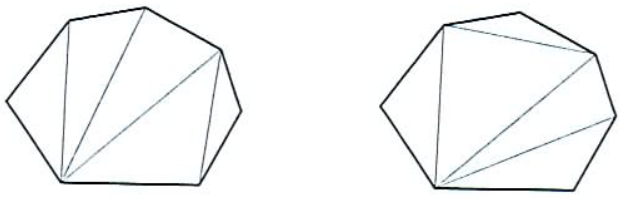


Ilustración 185 P7-Dos triangulaciones diferentes de un heptágono convexo dado.

a)

$$T[i,j] = \min_{k=i+1}^{j-1} (T[i,k] + T[k,j] + d_{ik} + d_{kj})$$

La condición base se aplica cuando i y j son inmediatamente vecinos, $T[i, i+1] = 0$

b)

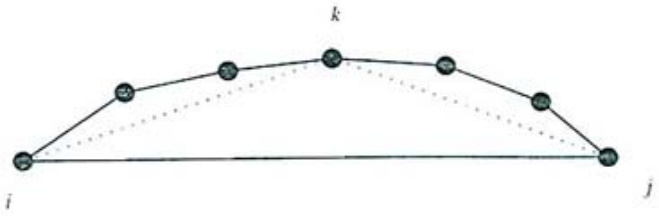


Ilustración 186 P7-Función a) y Seleccionando el vértice k para el par de borde (i,j) en el polígono.

```
Minimum-Weight-Triangulation(P)
    for i = 1 to n - 1 do T[i, i + 1] = 0
    for gap = 2 to n - 1
        for i = 1 to n - gap do
            j = i + gap
            T[i, j] = min_{k=i+1}^{j-1} (T[i, k] + T[k, j] + d_{P_i,P_k} + d_{P_k,P_j})
    return T[1, n]
```

Ilustración 187 P7-Pseudocódigo.

3.10.7 Dynamic Programming Applied to Recursión, n! [P8]

Problema:

Consider a program that calculates the values of 4! and the 5! with two invocations to a method that implements a recursive algorithm the factorial function. In the process of calculating the value 4! The recursive algorithm also calculates the value of 3!, 2! and 1! A dynamic version of this algorithm would store these calculated values in a table during the first invocation of the method.

Visualización:

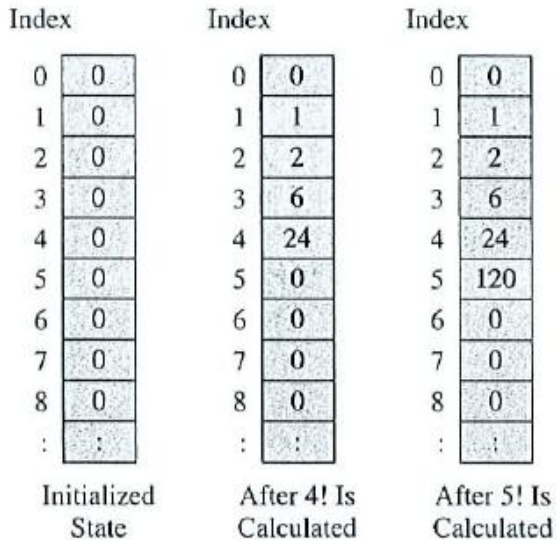


Ilustración 188 P8-Cambios del contenido del array usado en el algoritmo recursivo dinámico para n!

4 Tabla de Resumen de Problemas

1. Cadenas Caracteres					
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 2 (pág. 205)	Tabla con resultado de ejemplo (Fig. 7.1 pág. 207)	Pseudocódigo (pág. 207)	*The longest common subsequence problem*
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to Algorithms***	Capítulo 15 Apartado 4 (pág. 390)	Tabla con resultado de ejemplo (Fig. 15.8 pág. 395)	Pseudocódigo (págs. 394/5)	*Longest common subsequence*
P3	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 2 (pág. 263)	Método(Fig.7.13 pág. 265) Tabla con resultado (Fig. 7.2 pág. 66)	NO	*El problema de la subsecuencia común más larga*
P4	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.3 (pág. 503)	Dos casos del algoritmo (Fig. 12.10 pág. 504) Tabla de construcción (Fig. 12.11 pág. 506)	Pseudocódigo (pág. 505)	*The longest common subsequence*
P5	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 5 (pág. 471)	NO	Pseudocódigo (pág. 474)	*Separating sequences of words into lines*
P6	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 13 (pág. 432)	Esquema de tabla (Fig. 13.9 pág. 433)	Pseudocódigo (pág. 434)	*Transformar A en B*
P7	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 2 (pág. 266)	Tablas con resultados (Tabla 7.3 / 7.4 pág. 268)	NO	*El problema de alineación de dos secuencias*
P8	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 3 (pág. 289)	Matriz editando cadenas. (Fig. 8.4 pág. 284) Matriz padre de editando cadenas. (Fig. 8.5 pág. 285)	Código C - editando cadena, recursivo (págs. 281/2) Código C - editando cadena, pd (pág. 283) Código C - reconstruyendo cadena pd (págs. 285/6) Código C - variedades-tabla de inicialización pd (pág. 286) Código C - variedades-coste de penalización pd (pág. 287) Código C - variedades-Identificación de la celda objetivo pd (pág. 287) Código C - variedades-acciones de rastreo pd (pág. 287) Código C - variedades-Correspondencia de substring pd (pág. 288) Código C - variedades-Subsecuencia común más larga pd (pág. 288) Código C - Comparación de string, pd (http://www.cs.sunysb.edu/~s	*Approximate String Matching*

				kiena/392/programs/editdistance.c) Código Java (http://www.cs.sunysb.edu/~skiena/392/javaprograms/)	
P9	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 3 (pág. 289)	Tabla con valores de ejemplo (pág. 291)	NO	*Longest increasing sequence*
P10	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 6 (pág. 298)	Gramática libre de contexto y árbol. (Fig. 8.9 pág. 298)	NO	*Parsing Context-Free Grammars*
P11	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 3 (pág. 270)	Estructura secundaria (Fig. 7.14 pág. 271) Ilustraciones de casos (Fig. 7.15-17 pág. 273/4) Tabla con resultado ejemplo (Tabla 7.5 pág. 275)	Pseudocódigo (págs. 275-6)	*Problema de apareamiento del máximo par de bases de ARN*
2. Caminos Mínimos					
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 5 (pág. 215)	Grafo ejemplo y matrices (Fig. 7.4/5 pág. 216)	Pseudocódigo (pág. 213)	*The All-Pairs Shortest Path Problem*
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado5 (pág. 301)	Ejemplo de Algoritmo de Floyd (Fig. 8.5 pág. 302) Matriz con resultado (pág. 304)	Pseudocódigo (pág. 303)	*Caminos Mínimos*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 3 (pág. 208)	Grafo con ciclo negativo. (Fig. 5.3 pág. 209) Grafo dirigido, y matrices (Figs. 5.4/5 págs. 210/1)	Pseudocódigo. (pág. 210)	*All Pairs Shortest Path*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 9 (pág. 397)	Esquema de tabla (Fig. 13.7 pág. 423)	Pseudocódigo (pág. 424) Pseudocódigo - nº camino mínimo (pág. 425) Pseudocódigo - algoritmo Warshall (pág. 426)	*Camino mínimo*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 (pág. 253)	La ruta más corta - Grafo explicativo (Figs. 7.2/6 págs. 254/8)	NO	*Ruta más corta*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett.***	Capítulo 3 Apartado 2 (pág. 94)	Grafo dirigido con pesos. (Fig. 3.2 pág.95). Tablas pesos y longitudes (Fig. 3.3 pág. 97) Grafo explicativo (Fig. 3.4 pág. 98) Tabla camino mínimo (Fig. 3.5 pág. 102)	Pseudocódigo (pág. 100) Pseudocódigo - caminos más cortos. (pág. 101) Pseudocódigo - Imprimir. (pág. 102)	*Floyd´sAlgorithm for shortest paths*
P7	I. Parberry. ***Problems on Algorithms.***	Capítulo 8 Apartado 4 (pág. 91)	NO	Pseudocódigo (pág. 91)	*Floyd´s Algorithm*
P8	S. Sahni ***Data Structures, Algorithms, and Applications in Java.***	Capítulo 20 Apartado 2.3 (pág. 815)	Ejemplo-Matriz (Fig. 20.4 pág. 819)	Pseudocódigo - iterativo (pág. 816) Código Java - iterativo. (págs. 817/8)	*All-Pairs Shortest Paths*

P9	R. Sedgewick, ***Algorithms in Java***	Capítulo 21 Apartado 3 (pág. 308)	Floyd´s algorithm (Fig. 21.14 pág. 310)	Pseudocódigo *Floyd's algorithm shortest path – Dynamic programming* (pág. 308)	*Shortest Paths*
P10	S. Sahni ***Data Structures, Algorithms, and Applications in Java.***	Capítulo 20 Apartado 2.4 (pág. 819)	NO	Pseudocódigo – CN iterativo. (pág. 821) Pseudocódigo - CN iterativo - refinado. (pág. 821) Pseudocódigo – CN iterativo- final. (pág. 822) Código Java - CN iterativo. (pág. 823)	*Single-Source Shortest Paths with Negative Cost. Algoritmo Bellman-Ford*
P11	*E. Horowitz y S. Sahni* , ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 7 (pág. 231)	Grafo dirigido y matriz. (Fig. 15 pág. 232)	NO	*The traveling sales person problem*
P12	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett.***	Capítulo 3 Apartado 6 (pág. 123)	Grafo (Fig. 3.16 pág. 124) Matriz (Fig. 3.17 pág. 124)	Pseudocódigo (págs. 126/7)	*The traveling sales person problem*

3 Coeficientes Binomiales

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 1.1 (pág. 292)	Triangulo de Pascal (Fig. 8.1 pág. 291)	Pseudocódigo - sin pd (pág. 291).	*Cálculo del coeficiente binomial.*
P2	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Capítulo 3 Apartado 1 (pág. 90)	Llenado de array (Fig. 3.1 págs. 92/3)	Código Pascal - divide y vencerás (pág. 91) Código Pascal – pd (pág. 93)	*The Binomial Coefficient*
P3	S. Skiena. ***The Algorithm Design Manual***	Capítulo 8. Apartado 1 (pág. 278)	Triangulo de Pascal (pág. 278) Tablas evaluación orden. (Fig. 8.3 pág. 279)	Pseudocódigo –pd (pág. 280) Código C - pd (http://www.cs.sunysb.edu/~skiena/392/programs/binomial.c) Código Java (http://www.cs.sunysb.edu/~skiena/392/javaprograms/)	*Binomail Coefficients*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 1 (pág. 397)	Triangulo de Pascal (pág. 397) Árbol llamadas recursivas (Fig. 13.1 pág. 398)	NO	*Esquema general. Coeficientes binomiales*
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 18 (pág. 447)	Triangulo de Pascal (Tabla 13.2 pág. 448)	Pseudocódigo (pág. 448)	*Número combinatorio*

4 Problemas de Árboles

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 4 (pág. 466)	Árbol búsqueda binario (Fig:10.3 pág. 466) Eligiendo K_k como raíz	Pseudocódigo (págs. 470/1)	*Constructing Optimal Binary Search Trees*

			(Fig. 10.4 pág. 469) Coste de computación (Fig. 10.5 pág. 470)		
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Capítulo IV Apartato15.5 (pág. 397)	Ejemplo árboles (Fig. 15.9 pág. 398) Tablas con valores (Fig. 15.10 pág. 403)	Pseudocódigo (pág. 402)	*Optimal Binary Search Trees*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 4 (pág. 211)	Árboles (Fig. 5.6 pág. 212) Árboles - nodos externos (Fig. 5.7 pág. 213) Árboles - ejemplo (Ex. 5.15 pág. 214/5) Árbol - óptimo (Fig. 5.8 pág. 216) Tabla valores ejemplo (Fig. 5.9 pág. 218) Árbol óptimo - ejemplo (Fig. 5.10 pág. 218)	Pseudocódigo (pág. 212). Pseudocódigo - mínimo coste. (pág. 219)	*Optimal Binary Search Trees*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo**. *Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 7 (pág. 414)	NO	Pseudocódigo (pág. 416/7)	*Árbol de búsqueda binario óptimo*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Capítulo 7 Apartado 6 (pág. 283)	Ilustraciones (Figs7.19-24 págs. 284-90)	NO	*El problema del árbol binario óptimo*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Capítulo 3 Apartado 5 (pág. 113)	Árboles binarios (Fig. 3.10 pág. 114) Árboles binarios-ejemplos (Figs. 3.11/2 págs. 117/8) Eligiendo Key_k como raíz (Fig. 3.13 pág. 119) Tabla de cálculo de valores y árbol de un ejemplo (Figs. 3.14/5 págs. 122/3)	Pseudocódigo (págs. 115/6) Pseudocódigo óptimo. (págs. 120/1) Pseudocódigo construcción (págs. 121/2)	*Optimal Binary Search Trees*
P7	Ian Parberry. ***Problems on Algorithms***	Capítulo 8 Apartado 3 (pág. 90)	NO	Pseudocódigo – pd (pág. 90)	*Optimal Binary Search Trees*
P8	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 7 (pág. 283)	Ilustraciones (Figs. 7.25/33 págs. 291/9)	Pseudocódigo (págs. 299-300)	*El problema ponderado de dominación perfecta en árboles*
5 Problemas de Grafos					
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 2 (pág. 203)	Grafo 5 estados. (Fig. 5.1 pág. 203) Grafo 4 estados - problema de 3 proyectos. (Fig. 5.2 pág. 204)	Pseudocódigo – aproximación hacia delante. (pág. 206) Pseudocódigo – aproximación hacia atrás. (pág. 207)	*Multistage Graphs*
P2	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.5 (pág. 825)	Ejemplo - redes. (Fig. 20.8 pág. 825) Ejemplo - MNS. (Fig. 20.9 pág. 826) Tabla figura 20.8.	Código Java – iterativo. (pág. 829) Código Java – iterativo MNS. (pág. 830)	*Noncrossing Subset of Nets*

			(Figura 20.10 pág. 828)		
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 10 (pág. 426)	NO	Pseudocódigo (pág. 428)	*Puentes - Islas*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 11 (pág. 428)	NO	Pseudocódigo (pág. 429/30)	*Planificación viaje - canoas*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 8 (pág. 301)	Ejemplos búsquedas (Fígs.7.34/44 págs. 301/9)	NO	*El problema ponderado de búsqueda de bordes en una gráfica en un solo paso*
P6	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 9 (pág. 309)	Ilustraciones (Figs. 7.45/49 págs. 310/14)	NO	*El problema de rutas de m-Vigilantes para polígonos de 1 espiral resuelto para pd.*
6 Monedas					
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 2 (pág. 295)	Tabla (Fig. 8.3 pág. 296)	Pseudocódigo (pág. 297)	*Devolver Cambio*
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 2 (pág. 398)	Esquema de tabla (Fig. 13.2 pág. 400)	Pseudocódigo -tabla (pág. 400) Pseudocódigo – vector (págs. 401/2)	*Devolver Cambio*
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 12 (pág. 430)	Esquema Temporal (Fig. 13 pág. 431)	Pseudocódigo (pág. 431)	*La cantidad máxima de dinero que se puede obtener haciendo inversiones adecuadas.*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 3 (pág. 407)	NO	Pseudocódigo (pág. 408)	*Diferentes formas de franquear una carta.*
7 Multiplicación de Matrices					
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 3 (pág. 208)	Tabla (Fig. 7.2 pág. 211)	Pseudocódigo (pág. 212)	*Matrix Chain Multiplication*
P2	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 3 (pág. 457)	Árbol ejemplo (Fig. 10.2 pág. 464)	Pseudocódigo (pág. 462) Pseudocódigo-orden óptimo (pág. 465)	*Multiplying a Sequence of Matrices*
P3	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 6 (pág. 304)	Tabla (Fig. 8.6 pág. 308)	Pseudocódigo – recursión y memorización (pág. 312)	*Multiplicación encadenada de matrices*
P4	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Capítulo 15 Apartado 2 (pág. 370)	Tabla (Fig. 15.5 pág. 376) Árbol de recursión. (Fig. 15.7 pág. 385)	Pseudocódigo - sin pd. (pág. 371) Pseudocódigo - ordenación sin pd. (pág. 375) Pseudocódigo – paréntesis óptimos(pág. 377) Pseudocódigo – recursivo (pág. 385) Pseudocódigo -	*Matrix chain multiplication*

				memoización (pág. 388)	
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 6 (pág. 411)	Esquema tabla (Fig. 13.4 pág. 412) Recorrido tabla (Fig. 13.5 pág. 413)	Pseudocódigo (pág. 413/4)	*Producto de matrices*
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Capítulo 3 Apartado 4 (pág. 105)	Tabla ejemplo (Fig. 3.8 pág. 108) Tabla – p ejemplo (Fig. 3.9 pág. 112)	Pseudocódigo (págs. 110/1) Pseudocódigo - Orden (págs. 112/3)	*Chained Matrix Multiplication*
P7	Ian Parberry. ***Problems on Algorithms***	Capítulo 8 Apartado 1 (pág. 87)	NO	Pseudocódigo (pág. 87)	*Iterated Matrix Product*
P8	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.2 (pág. 807)	NO	Pseudocódigo - recursivo (pág. 811) Pseudocódigo - recursivo con tabla (pág. 813) Pseudocódigo – iterativo (pág. 814)	*Matrix Multiplication Chains*
P9	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.1 (pág. 499)	Modo en el que se emplean los elementos de la matriz para rellenar el array N (Fig. 12.9 pág. 502)	Pseudocódigo (pág. 501)	*Matrix-chain-product*

8 Planificación

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Capítulo 7 Apartado 6 (pág. 217)	Tabla ejemplo (Fig. 7.5 pág. 219)	Pseudocódigo (pág. 218)	*The Knapsack Problem*
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 4 (pág. 299)	Tabla ejemplo (Fig. 8.4 pág. 300)	NO	*El problema de la mochila*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 5 (pág. 219)	Ejemplo valores mochila (Fig. 5.11 pág. 221)	Pseudocódigo - algoritmo informal (pág. 223) Pseudocódigo (pág. 225)	*0/1 knapsack*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 2 (pág. 402)	Esquema de tabla. (Fig. 13.3 pág. 404)	Pseudocódigo - matriz. (pág. 404) Pseudocódigo – riquezas ilimitadas (págs. 405/6) Pseudocódigo - pesos reales (pág. 407)	*El problema de la mochila(Ali Babá y los cuarenta ladrones)*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 5 (pág. 282)	Método (Fig. 7.18 pág. 283)	NO	*Problema 0/1 de la mochila*
P6	R. Sedgewick. ***Algorithms in Java***	Capítulo 5. Apartado 3.5 (pág. 219)	Ejemplo (Fig. 5.16 pág. 223) Estructura recursiva (Fig. 5.17 pág. 224) Estructura pd top-down (Fig. 5.18 pág. 226)	Pseudocódig - recursivo (pág. 223) Pseudocódig - pd y recursivo (pág. 225)	*Knapsack Problem*
P7	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***	Capítulo 20 Apartado 2.1 (pág. 801)	Árbol recursivo (Fig. 20.1 pág.803) Tabla – ejemplo (Fig.20.2 pág. 804)	Pseudocódigo – recursivo (pág. 802) Pseudocódigo - recursivo con tabla (pág. 804) Pseudocódigo – iterativo (pág. 805)	*Knapsack Problem*

				Pseudocódigo – iterativa computación de x (pág. 806)	
P8	R. Neapolitan y K. Naimipou. ***Foundations of Algorithms***	Capítulo 4 Apartado 4.4 (pág. 169)	Elementos y solución voraz y óptima (Fig. 4.10 pág. 167)	NO	*A Dynamic Programming Approach to the 0-1 Knapsack Problem*
P9	I. Parberry. ***Problems on Algorithms***	Capítulo 8 Apartado 2 (pág. 89)	NO	Pseudocódigo (pág. 89)	*The knapsack problem*
P10	M.T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Capítulo 12 Apartado 3.4 (pág. 507)	Ejemplo de llenado de la mochila (Fig. 12.12 pág. 508)	Pseudocódigo (pág. 509)	*The* 0-1 *knapsack problem*
P11	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to Algorithms***	Capítulo 15 Apartado 1 (pág. 360)	Ejemplo tabla - longitud / precio (Fig. 15.1 pág. 360) Ejemplo diferentes formas de cortar barra (Fig. 15.2 pág. 361) Árbol de recursión (Fig. 15.3 pag. 364)	Pseudocódigo - recursivo top-down no pd (pág. 363) Pseudocódigo memoización top-down (págs. 365/6) Pseudocódigo – memoización bottom-up (pág. 366) Pseudocódigo - lista tamaño piezas memoización (pág. 369)	*Rod cutting*
P12	*E. Horowitz y S. Sahni ,* ***Fundamentals of Computer Algorithms***	Capítulo 5 Apartado 8 (pág. 234)	Esquema de ejemplo (Fig. 5.15/6 págs. 234/5)	NO	*Flow Shop Schedulin (jobs-processors)*
P13	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 15 (pág. 438)	Asignación de tarea – solución 1 (Fig. 13.11 pág. 440) Esquema de tabla - solución 2 (Fig. 13.12 pág. 441)	Pseudocódigo –solución 1 ineficiente (págs. 439) Pseudocódigo –solución 2 más eficiente (págs. 441/42)	*Tareas - Procesadores*
P14	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 14 (pág. 435)	Esquema de tabla – solución 1 (Fig. 13.10 pág. 436)	Pseudocódigo –solución 2 más eficiente (págs. 436/37) Pseudocódigo –solución 3 límites más sencillos (págs. 437/38)	*Programas - Cintas*
P15	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Capítulo 7 Apartado 1 (pág. 259)	Primera etapa (Fig. 7.7 pág. 260) Dos primeras etapas (Fig. 7.8 pág. 260) Gráfica multietapas (Fig. 7.9 pág. 261) Rutas más largas (Fig. 7.10/12 págs. 261/2)	NO	*Asignación - Recursos*

9 Sucesión de Fibonacci

	Libro	Capítulo / Apartado	Visualización	Implementación	Nomenclatura
P1	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***	Capítulo 10 Apartado 1 (pág. 452)	Grafo-subproblema (Fig. 10.1 pág. 454)	Pseudocódigo (pág. 456)	*Fibonacci function (Introdución)*
P2	R. Sedgewick, ***Algorithms in Java***	Capítulo 5 Apartado 3 (pág. 219)	Estructura alg. recursivo (Fig. 5.14 pág. 221) Árbol pd top-down (Fig. 5.15 pág. 222)	Pseudocódigo recursivo (pág. 220) Pseudocódigo pd y recursivo (pág. 222)	*Fibonacci numbers*
P3	S. Skiena, ***The Algorithm Design***	Capítulo 8	Árbol recursivo.	Pseudocódigo - recursivo	*Fibonacci numbers*

	Manual	Apartado1 (pág. 273)	(Fig. 8.1 pág. 275) Árbol recursivo - almacenamiento. (Fig. 8.2 pág. 276)	(pág. 274) Pseudocódigo - recursivo con almacenamiento (pág. 276) Pseudocódigo – pd (pág. 277) Pseudocódigo – pd almacenando sólo los dos últimos valores (pág. 278)	
P4	William McAllister, ***Data Structures and Algorithms using Java.***	Capítulo 6 Apartado 4 (pág. 327)	NO	Pseudocódigo (pág. 329)	Dynamic Programming Applied to Recursión, Fibonacci Sequence.
	10 Otros Problemas				
	Libro	**Capítulo / Apartado**	**Visualización**	**Implementación**	**Nomenclatura**
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Capítulo 8 Apartado 1.2 (pág. 293)	Llamadas recursivas (Fig. 8.2 pág. 294)	Pseudocódigo (pág. 295)	El Campeonato Mundial
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 5 (pág. 410)	NO	Pseudocódigo (pág. 411)	El Campeonato Mundial (con y sin empate)
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 4 (pág. 408)	Vectores en el calculo de recurrencia (pág. 409)	Pseudocódigo (págs. 409/10)	Reparto de botín
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 16 (pág. 442)	NO	Pseudocódigo (pág. 443)	Reparto Almacenes
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Capítulo 13 Apartado 17 (pág. 444)	Esquema de tabla (Fig. 13.13 pág. 445)	Pseudocódigo (págs. 445/7)	Vacas-pienso
P6	S. Skiena, **The *Algorithm Design Manual***	Capitulo 8 Apartado5 (pág. 294)	Matrices con ejemplos de entrada (Fig. 8.8 pág. 297)	Pseudocódigo (pág. 296)	The Partition Problem
P7	S. Skiena, ***The Algorithm Design Manual***	Capítulo 8 Apartado 6.1 (pág. 300)	Ejemplo triangulaciones (Fig. 8.10 pág. 300) Seleccionando vértice (Fig. 8.11 pág. 301)	Pseudocódigo (pág. 301)	Minimum weight triangulation
P8	William McAllister, ***Data Structures and Algorithms using Java***.	Capítulo 6 Apartado 4.1 (pág. 327)	Cambio contenido array n! (Fig. 6.14 pág. 328)	NO	Dynamic Programming Applied to recursión, n!

A continuación se presenta otra tabla donde se encuentran todos problemas presentados en los apartados anteriores, con un primer análisis de las características de las figuras.

	1. Cadenas de Caracteres			
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Fig. 7.1 pág. 207	Tabla bidimensional con valores (resultado) En cada dimensión, etiquetas de valores	*The longest common subsequence problem*

			Separación de etiquetas y celdas (doble borde)	
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to Algorithms***	Fig. 15.8 pág. 395	Tabla bidimensional con valores (resultado) En cada dimensión, identificador de índice y etiquetas de índices y valores Dependencias entre celdas (flechas) Resaltada la secuencia de decisiones óptimas (sombreado de etiquetas y celdas)	*Longest common subsequence*
P3	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Fig.7.13 pág. 265	Tabla bidimensional con valores (resultado) Etiqueta de tabla (celda genérica) En cada dimensión, identificador de índice y etiquetas de índices y valores	*El problema de la subsecuencia común más larga*
		Fig. 7.2 pág. 66	Tabla genérica (tamaños *m*, *n*) con etiquetas (explicación) Dependencias entre celdas (flechas) Sólo se muestra la parte izquierda, superior y la celda inferior derecha (resultado) Resto suprimido	
P4	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Fig. 12.10 pág. 504	Cadenas de caracteres (ejemplo), se emplean colores y líneas para identificar las similitudes entre ambas	*The longest common subsequence*
		Fig. 12.11 pág. 506	Tabla bidimensional (ejemplo) Etiqueta de los valores numéricas Cadenas de caracteres referentes a los valores de la tabla. Se emplean colores y líneas para para identificar las similitudes entre ambas	
P5	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***		NO	*Separating sequences of words into lines*
P6	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.9 pág. 433	Tabla genérica para tamaños *c*, *n* (explicación) Etiqueta de tabla En cada dimensión, etiquetas de valores Dependencia genérica entre celdas (flechas y sombreado de celda dependiente) Resto suprimido	*Transformar A en B*
P7	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Tabla 7.3 pág. 268	Tabla bidimensional rellena (resultado) Etiquetas de identificadores En cada dimensión, identificador de índice y etiquetas de índices y valores Dependencias entre celdas (flechas)	*El problema de alineación de dos secuencias*
		Tabla 7.4 pág. 268	Tabla bidimensional rellena (resultado) En cada dimensión, etiquetas de índices y valores Dependencias entre celdas (flechas)	
P8	S. Skiena, ***The Algorithm Design Manual***	Fig. 8.4 pág. 284	Tabla bidimensional rellena (resultado) En cada dimensión, identificador de índice y etiquetas de índices y valores Resaltada la secuencia de decisiones óptimas (negrita) Separación de etiquetas y celdas (borde)	*Approximate String Matching*
		Fig. 8.5 pág. 285	Tabla bidimensional rellena (resultado, ¿decisiones?) En cada dimensión, identificador de índice y etiquetas de índices y valores Resaltada la secuencia de decisiones óptimas (negrita)	

			Separación de etiquetas y celdas (borde)	
P9	S. Skiena, ***The Algorithm Design Manual***		NO	*Longest increasing sequence*
P10	S. Skiena, ***The Algorithm Design Manual***	Fig. 8.9 pág. 298	Representación dependiente del dominio (árbol de análisis sintáctico)	*Parsing Context-Free Grammars*
P11	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos***	Fig. 7.14 pág. 271	Figura séxtuple, con representaciones dependientes del dominio (estructura química: ejemplos)	*Problema de apareamiento del máximo par de bases de ARN*
		Fig. 7.15 pág. 273	Representación dependiente del dominio (estructura química: explicación) Etiquetas y flecha	
		Fig. 7.16 pág. 274	Representación dependiente del dominio (estructura química: explicación) Etiquetas y flechas	
		Fig. 7.17 pág. 274	Representación dependiente del dominio (estructura química: explicación) Etiquetas y flechas	
		Tabla 7.5 pág. 275	Tabla bidimensional rellena (resultado) En cada dimensión, identificador de índice y etiquetas de índices Parte sin rellenar (con guiones) Separación de etiquetas superiores y el resto (sombreado)	
P12	I. Parberry, *Problems on Algorithms*	Fig. 8.2 pág. 93	Secuencia de representaciones dependientes del dominio (cadenas de caracteres: resultado)	
		Fig. 8.2 pág. 94	Secuencia de representaciones dependientes del dominio (cadenas de caracteres: resultado)	
			2. Caminos Mínimos	
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	M. H. Alsuwaiyel, ***Algorithms Design Techniques and Analysis***	Fig. 7.4 pág. 216	Representación dependiente del dominio (grafo: ejemplo)	*The All-Pairs Shortest Path Problem*
		Fig. 7.5 pág. 216	Secuencia de tablas rellenas Formato de matriz matemática	
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.5 pág. 302	Figura doble: • Secuencia de tablas rellenas (formato de matriz matemática: resultado) • Representación dependiente del dominio (grafo: ejemplo)	*Caminos Mínimos*
		pág. 304	Representación dependiente del dominio (grafo en formato de matriz matemática: resultado)	
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Fig. 5.3 pág. 209	Figura doble: • Representación dependiente del dominio (grafo: ejemplo) • Matriz de adyacencia	*All Pairs Shortest Path*
		Fig. 5.4 pág. 210	Figura doble: • Representación dependiente del dominio (grafo: ejemplo) • Matriz de adyacencia	
		Fig. 5.5 pág. 211	Secuencia de tablas rellenas (ejemplo) Etiqueta de tabla En cada dimensión, etiquetas de índices	
P4	N. Martí Oliet, Y. Ortega y J. A.	Fig. 13.7 pág.	Dos tablas genéricas (explicación)	*Camino mínimo*

	Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	423	Etiqueta de tabla En cada dimensión, etiquetas genéricas de índices Dependencia genérica entre celdas (líneas y sombreado de celda dependiente) Líneas en filas y columnas genéricas Resto suprimido	
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Fig. 7.2 pág. 254	Figura doble: • Representación dependiente del dominio (grafo: ejemplo) • Representación dependiente del dominio (camino: resultado) Etiquetas en nodos (numeración alfabética)	*Ruta más corta*
		Figs. 7.3, 7.4,7.5 págs 254/5/6	Secuencia de representaciones dependientes del dominio (grafo) correspondientes a subproblemas Etiquetas genéricas en nodos (numeración alfabética) y arcos (formulación de subsolución)	
		Fig. 7.6 pág.258	Figura doble con representaciones dependientes del dominio (grafos: ejemplo y resultado) Etiquetas en nodos (numeración alfabética)	
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Fig. 3.2 pág. 95	Representación dependiente del dominio (grafo: ejemplo) Etiquetas en nodos (numeración con subíndices)	*Floyd's algorithm for shortest paths*
		Fig. 3.3 pág. 97	Figura doble de matrices (de adyacencia y de caminos mínimos) Etiqueta de tabla En cada dimensión, etiquetas de índice Separación de etiquetas y el resto (bordes)	
		Fig. 3.4 pág. 98	Grafo (explicativo) Etiquetas genéricas en interior de nodos Uso de puntos suspensivos Emplea llaves para dar información sobre diferentes partes del grafo	
		Fig. 3.5 pág. 102	Matriz de caminos mínimos En cada dimensión, etiquetas de índice Separación de etiquetas y el resto (bordes)	
P7	I. Parberry. ***Problems on Algorithms***		NO	*Floyd´s Algorithm*
P8	S. Sahni, ***Data Structures, Algorithms, and Applications in Java***	Fig. 20.4 pág. 819	Figura triple de dos arrays de costes y el tercero de resultado. Valores separados por espacio.	*All-Pairs Shortest Paths*
P9	R. Sedgewick, ***Algorithms in Java***	Fig. 21.14 pág. 310	Secuencia de imágenes que ilustran la construcción del camino mínimo. Se ilustra con grafos así como con tablas. Se sombrean valores de la tabla Se sombrean nodos y flechas del grafo	*Shortest path*
P10	S. Sahni, ***Data Structures, Algorithms, and Applications in Java***	NO		*Single-Source Shortest Paths with Negative Cost. Algoritmo Bellman-Ford*
P11	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Fig. 15 pág. 232	Figura doble (ejemplo): • Representación dependiente del dominio (grafo) • Matriz de adyacencia	*The traveling sales person problem*
P12	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones***	Fig. 3.16 pág. 124	Representación dependiente del dominio (grafo: ejemplo)	*The traveling sales person problem*

	and Bartlett		Etiquetas en nodos (numeración con subíndices)	
		Fig. 3.17 pág. 124	Matriz de adyacencia En cada dimensión, etiquetas de índice Separación de etiquetas y el resto (bordes)	
		3. Coeficientes Binomiales		
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.1 pág. 291	Tabla genérica para tamaños *k*, *n* (explicación) En ambas dimensiones, etiquetas de índices (con puntos suspensivos) Rellenas celdas iniciales (parte superior izquierda) y genéricas con dependencias entre celdas (flechas y operador) Resto vacío	*Cálculo del coeficiente binomial*
P2	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***.	Fig. 3.1 págs. 92/3	Tabla genérica para tamaños *k*, *n* (explicación) En ambas dimensiones, etiquetas de índices (con espacio en blanco) Rellenas celdas iniciales (parte superior izquierda) y genéricas con dependencias entre celdas (flechas) Resto vacío	*The Binomial Coefficient*
P3	S. Skiena. ***The Algorithm Design Manual***	Pág. 278	Representación dependiente del dominio (triángulo de Pascal: ejemplo)	*Binomail Coefficients*
		Fig. 8.3 pág. 279	Figura doble con tabla (ejemplo): • Tabla con orden de rellenado • Tabla rellena (sólo la mitad significativa: resultado) En ambas dimensiones, etiqueta de variable e índices Parte no significativa vacía Separación de etiquetas y el resto (bordes)	
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.1 pág. 398	Árbol de recursión (ejemplo) Nodos con representación dependiente del dominio (números combinatorios)	*Esquema general. Coeficientes binomiales*
		pág. 397	Representación dependiente del dominio (triángulo de Pascal: ejemplo genérico) Puntos suspensivos	
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Tabla 13.2 pág. 448	Tabla genérica para *n*, *r* (explicación) En cada dimensión, etiquetas de valores Separación de etiquetas y el resto (bordes) Partes simplificadas con puntos suspensivos Etiquetas genéricas (identificadores y números combinatorios)	*Número combinatorio*
		4. Problemas de Árboles		
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Fig. 10.3 pág. 466	Representación dependiente del dominio (árbol binario de búsqueda: resultado)	*Constructing Optimal Binary Search Trees*
		Fig. 10.4 pág. 469	Figura doble, genérica (explicación): • Representación dependiente del dominio, (árbol binario de búsqueda) • Tabla de tamaño *n* (etiquetas de índices	

			en ambas dimensiones; sólo casos base y una fila y columna genérica)	
		Fig. 10.5 pág. 470	Tabla de rendimiento	
P2	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Fig. 15.9 pág. 398	Figura triple, con tabla (datos de entrada: resultado) y dos representaciones genéricas del dominio (árbol binario de búsqueda: dos soluciones posibles) Etiquetas en nodos (numeración con subíndices) Tablas con valores	*Optimal Binary Search Trees*
		Fig. 15.10 pág. 403	Figura triple, con tablas rellenas (¿dos de valores y una de decisiones?: resultado) Etiqueta de tabla Por cada dimensión, etiqueta de variable e índices Media tabla sin dibujar Tablas rotadas, con celda de solución arriba	
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Fig. 5.6 pág. 212	Figura doble, con representaciones dependientes del dominio (árbol binario de búsqueda: dos resultados posibles)	*Optimal Binary Search Trees*
		Fig. 5.7 pág. 213	Figura doble, con representaciones dependientes del dominio (igual a fig. anterior, con nodos externos)	
		Example 5.15 págs. 214/5	Figura quíntuple, con representaciones dependientes del dominio (árbol binario de búsqueda: cinco resultados posibles)	
		Fig. 5.8 pág. 216	Representación dependiente del dominio (árbol binario de búsqueda: explicación) Subárboles representados como triángulos Etiquetas (identificadores)	
		Fig. 5.8 pág. 216	Representación dependiente del dominio (tabla con valores: resultado) Emplean etiquetas numéricas para columnas y filas, así como flechas La información se separa mediante espacios	
		Fig. 5.10 pág. 218	Representación dependiente del dominio (árbol binario de búsqueda: resultado)	
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***		NO	*Árbol de búsqueda binario óptimo*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Fig. 7.19 pág. 284	Figura cuádruple, con representaciones dependientes del dominio (árbol binario de búsqueda: cuatro resultados posibles)	*El problema del árbol binario óptimo*
		Figs. 7.21/2 pág. 285	Figura doble, con representaciones dependientes del dominio (árbol binario de búsqueda): • Solución • Solución con nodos externos	
		Fig. 7.23 pág. 287	Representación dependiente del dominio (árbol binario de búsqueda: explicación) Subárboles simplificados como triángulos, con datos dentro	
		Fig. 7.24 pág. 290	Representación dependiente del dominio (árbol binario de búsqueda: explicación) Dependencia entre subárboles	

			(subproblemas) como flechas Etiquetas (subproblemas)	
P6	R. Neapolitan y K. Naimipour. ***Foundations of Algorithms, Jones and Bartlett***	Fig. 3.10 pág. 114	Figura doble, con representaciones dependientes del dominio (árbol binario de búsqueda: dos resultados posibles)	*Optimal Binary Search Trees*
		Fig. 3.11/12 págs. 117/8	Figura quíntuple, con representaciones dependientes del dominio (árbol binario de búsqueda: cinco resultados genéricos para n=3) Etiquetas (numeración con subíndices)	
		Fig. 3.13 pág. 119	Representación dependiente del dominio (árbol binario de búsqueda: explicación) Etiquetas (numeración con subíndices)	
		Figs. 3.14/5 págs. 122/3	Figura triple: • Vector y texto (ejemplo) • Dos tablas rellenas (de valores y decisiones: resultado; etiqueta de tabla; rellena una mitad y el resto vacío; etiquetas de índices) • Representación dependiente del dominio (árbol binario de búsqueda: resultado)	
P7	Ian Parberry. ***Problems on Algorithms***		NO	*Optimal Binary Search Trees*
P8	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos***	Fig. 7.25 pág. 291	Representación dependiente del dominio (árbol: resultado) Etiquetas (numeración con subíndices)	*El problema ponderado de dominación perfecta en árboles*
		Fig. 7.26 pág. 292	Secuencia de dos conjuntos disjuntos (operación de unión: explicación) Etiquetas (numeración con subíndices)	
		Figs. 7.27/33 págs. 295/9	Secuencia de seis representaciones dependientes del dominio (subgrafos: resultados parciales) Etiquetas (numeración con subíndices)	
	5. Problemas de Grafos			
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Fig. 5.1 pág. 203	Representación dependiente del dominio (grafo: ejemplo y resultado superpuestos)	*Multistage Graphs*
		Fig. 5.2 pág. 204	Representación dependiente del dominio (grafo concreto: explicación) Distancias sustituidas por fórmulas con numeración de índices	
P2	S. Sahni. ***Data Structures, Algorithms, and Applications in Java.***	Fig. 20.8 pág. 825	Representación dependiente del dominio (red: ejemplo)	*Noncrossing Subset of Nets*
		Fig. 20.9 pág. 826	Representación dependiente del dominio (red: resultado)	
		Fig. 20.10 pág. 828	Tabla rellena En cada dimensión, etiqueta de variable e índices Separación de índices y celdas (borde)	
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***		NO	*Puentes - Islas*
P4	N. Martí Oliet, Y. Ortega y J. A.		NO	*Planificación viaje - canoas*

	Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***			
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai, ***Introducción al diseño y análisis de algoritmos.***	Figs. 7.34/6 págs. 301/2	Tres figuras, con representación dependiente del dominio (grafo: explicación) Etiquetas (numeración alfabética, subíndices)	*El problema ponderado de búsqueda de bordes en una gráfica en un solo paso*
		Fig. 7.37 pág. 303	Figura triple, con representación dependiente del dominio (árbol: explicación de 3 casos) Subárboles simplificados como triángulos Etiquetas (subíndices)	
		Figs. 7.38/9 pág. 304	Dos figuras, con representación dependiente del dominio (árbol: explicación de dos reglas) Etiquetas (numeración alfabética, subíndices)	
		Figs. 7.40/41 págs. 306/9	Secuencia de cinco figuras, con representación dependiente del dominio (árbol: resultado) Etiqueta interna en nodos (subíndices)	
P6	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Fig. 7.45 pág. 310	Representación dependiente del dominio (polígono: problema y solución superpuestos) Líneas auxiliares Etiquetas en vértices extremos	*El problema de rutas de m-Vigilantes para polígonos de 1 espiral resuelto para pd.*
		Fig. 7.46 pág. 311	Representación dependiente del dominio (polígono: explicación) Etiquetas en vértices y partes	
		Fig. 7.47 pág. 311	Representación dependiente del dominio (polígono: explicación) Líneas auxiliares Etiquetas en vértices internos Texto y flechas de explicación	
		Fig. 7.48 pág. 313	Representación dependiente del dominio (polígono: explicación, solución superpuesta) Líneas y ángulos rectos auxiliares Etiquetas en vértices y puntos destacados	
		Fig. 7.49 pág. 314	Figura triple, con tres representaciones dependientes del dominio (polígono: casos particulares) Líneas y ángulos rectos auxiliares Etiquetas en vértices y puntos destacados	
			6. Monedas	
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.3 pág. 296	Tabla rellena con valores (resultado) Para columnas, etiquetas de variable e índices Para filas, etiquetas y valores independientes	*Devolver Cambio*
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.2 pág. 400	Tabla genérica (explicación) Etiqueta de tabla En cada dimensión, etiquetas genéricas de índices Dependencia entre celdas (flechas y sombreado de celda dependiente) Resto suprimido	*Devolver Cambio*

P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13. pág. 431	Representación dependiente del dominio (línea de tiempo: explicación) Etiquetas con valores genéricos	*La cantidad máxima de dinero que se puede obtener haciendo inversiones adecuadas.*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***		NO	*Diferentes formas de franquear una carta.*
	7. Multiplicación de Matrices			
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Fig. 7.2 pág. 211	Tabla de tamaño 5 (explicación) con etiquetas de celdas En una dimensión, etiqueta variable e índices En otra, etiquetas conjuntas de variable y valor Diagonales y flechas (orden de cómputo) Celdas sombreadas ¿?	*Matrix Chain Multiplication*
		Fig. 7.3 pág. 213	Tabla (resultado) Etiqueta y valor en cada celda	
P2	S. Baase y A. Van Gelder. ***Computer Algorithms: Introduction to Design and Analysis***	Fig. 10.2 pág. 464	Figura doble (resultado y solución): • Dos tablas (costes y decisiones; formato de matriz; sólo la mitad; celdas vacías con un punto) • Árbol binario (nodos internos con operadores dentro; nodos externos con etiquetas dentro y valores fuera)	*Multiplying a Sequence of Matrices*
P3	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.6 pág. 308	Tabla (resultado) En ambas dimensiones, etiquetas de variable e índices Orden de relleno (diagonales) Resto de celdas vacías Valor óptimo en cada fila (etiqueta y valor)	*Multiplicación encadenada de matrices*
P4	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein. ***Introduction to Algorithms***	Fig. 15.5 pág. 376	Figura doble, con dos tablas (de valores y de decisiones) Identificador de tabla arriba En ambas dimensiones, etiqueta y valores de índices Inclinada 45º con celda principal arriba Celdas sombreadas	*Matrix chain multiplication*
		Fig.15.7 pág. 385	Árbol de recursión Subárboles sombreados con silueta	
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.4 pág. 412	Tabla genérica (explicación) Etiqueta de tabla arriba izquierda Índices de valores genéricos Diagonal principal (separa la parte a rellenar) Dependencia entre celdas (celdas cuadradas, puntos suspensivos y sombreado de celda dependiente) Resto de celdas vacías	*Producto de matrices*
		Fig. 13.5 pág. 413	Tabla (explicación) Valores en ambas dimensiones Varias diagonales y resto vacías o puntos suspensivos Junto a cada diagonal, fuera de la tabla, valor óptimo	
P6	R. Neapolitan y K. Naimipour.	Fig. 3.8 pág.	Tabla de valores rellena	*Chained Matrix Multiplication*

	Foundations of Algorithms, Jones and Bartlett	108	En ambas dimensiones, etiquetas de valores Etiquetas de diagonales y de celda principal Dependencias parciales entre celdas (flechas para emparejar y sombreado para celda dependiente) Celdas vacías	
		Fig. 3.9 pág. 112	Tabla de decisiones rellena Etiqueta de tabla arriba e izquierda En ambas dimensiones, etiquetas de valores Celdas vacías	
P7	Ian Parberry. ***Problems on Algorithms***		NO	*Iterated Matrix Product*
P8	S. Sahni. ***Data Structures, Algorithms, and Applications in Java***		NO	*Matrix Multiplication Chains*
P9	M. T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Fig. 12.9 pág. 502	Tabla bidimensional (explicación) Etiqueta de la tabla y en cada dimensión Sólo se pone un valor (simplificar) Se emplean colores y flechas junto con las fórmulas que se aplican	*Matrix-chain-product*

8. Planificación

	Libro	Fig. y pág.	Visualización	Nomenclatura
P1	M. H. Alsuwaiyel. ***Algorithms Design Techniques and Analysis***	Fig. 7.5 pág. 219	Tabla rellena (resultado) Etiquetas de valores en ambas dimensiones Borde entre etiquetas y celdas	*The Knapsack Problem*
P2	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.4 pág. 300	Tabla rellena (resultado) En una dimensión, etiqueta variable e índices En otra, etiquetas conjuntas de variable y valor	*El problema de la mochila*
P3	E. Horowitz y S. Sahni. ***Fundamentals of Computer Algorithms***	Fig. 5.11 pág. 221	Secuencia de diagramas dobles y dependientes del dominio (cómputo) Etiquetas y valores por eje Líneas discontinuas	*0/1 knapsack*
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig.13.3 pág.404	Tabla rellena (explicación) Etiqueta de tabla En ambas dimensiones, etiquetas de valores Dependencia entre celdas (flechas y sombreado de celda dependiente) Resto de celdas vacías	*El problema de la mochila(Ali Babá y los cuarenta ladrones)*
P5	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Fig. 7.18 pág. 283	Árbol de recursión (resultado) Horizontal hacia la derecha, celdas de hoja unidas en un nodo final Dentro de cada nodo, etiquetas (inicial y final) o valores (decisiones) Etiqueta y valor de decisión en cada arco, más valor acumulado	*Problema 0/1 de la mochila*
P6	R. Sedgewick. ***Algorithms in Java.***	Fig. 5.16 pág. 223	Figura múltiple con representaciones dependientes del dominio (problema y 4 soluciones posibles)	*Knapsack Problem*
		Fig. 5.17 pág. 224	Árbol de recursión (resultado) Nodos internos circulares y externos, cuadrados Nodos con valor interno	
		Fig. 5.18 pág. 226	Árbol de recursión para memorización (resultado) Nodos internos circulares y externos, cuadrados	

			Nodos con valor interno	
P7	S. Sahni. ***Data Structures, Algorithms, and Applications in Java.***	Fig. 20.1 pág. 803	Árbol de recursión (resultado) Nodos circulares Nodos con valor interno (capacidad libre)	*Knapsack Problem*
		Fig. 20.2 pág. 804	Tabla (resultado) En ambas dimensiones, etiqueta de variable y de valores Bordes entre etiquetas y celdas	
P8	R. Neapolitan y K. Naimipou. ***Foundations of Algorithms***	Fig. 4.10 pág. 167	Se muestran los elementos como rectángulos sombreados. El tamaño de los elementos según su peso. La mochila se muestra como un rectángulo, cuyo tamaño es proporcional a su capacidad. Etiquetas para el peso como para el beneficio. Etiquetas identificativas de los elementos.	*A Dynamic Programming Approach to the 0-1 Knapsack Problem*
P9	I. Parberry**.** ***Problems on Algorithms***		NO	*The knapsack problem*
P10	M.T. Goodrich y R. Tamassia, ***Data Structures and Algorithms in Java***	Fig. 12.12 pág. 508	Figura múltiple en el que se representa el llenado de la mochila (ejemplo). Se emplean etiquetas numéricas para identificar el beneficio-peso. Se emplean colores, cuya intensidad referencia su beneficio.	*The 0-1 knapsack problem*
P11	T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, ***Introduction to Algorithms***	Fig. 15.2 pág. 361	Figura múltiple: seis representaciones dependientes del dominios (seis soluciones posibles)	*Rod cutting*
		Fig. 15.3 pág. 364	Árbol de recursión Nodos circulares y con valores	
			Grafo de dependencia Nodos circulares y con valores	
P12	*E. Horowitz y S. Sahni ,* ***Fundamentals of Computer Algorithms***	Fig.5.15 pág. 234	Figura doble, con representación dependiente del dominio (2 soluciones posibles) Etiquetas con subíndices e índices	*Flow Shop Schedulin (jobs-processors)*
		Fig.5.16 pág. 235	Representación dependiente del dominio (solución) Etiquetas con subíndices	
P13	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.11 pág. 440	Figura doble, con representación dependiente del dominio (duración de tareas: explicación) Etiquetas de variables y numeración alfabética	*Tareas - Procesadores*
		Fig. 13.12 pág. 441	Tabla genérica (explicación) En ambas dimensiones, etiquetas de valores (fórmulas) Líneas discontinuas para alinear Líneas discontinuas para representar partes suprimidas	
P14	N. Martí Oliet, Y. Ortega y J. A. Verdejo. ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.10 pág. 436	Tabla genérica (explicación) En ambas dimensiones, etiquetas de valores (fórmulas) Líneas discontinuas para alinear Líneas discontinuas para representar partes suprimidas	*Programas - Cintas*
P15	R.C.T. Lee, S.S. Tseng, R.C. Chang e Y.T. Tsai. ***Introducción al diseño y análisis de algoritmos.***	Figs. 7.7/12 págs. 260/2	Seis figuras, con grafos parciales de dependencia/ decisiones (cómputo) Etiquetas de valores y numeración alfabética	*Asignación - Recursos*

9. Sucesión de Fibonacci				
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	S. Baase y A. Van Gelder, ***Computer Algorithms: Introduction to Design and Analysis***	Fig. 10.1 pág. 454	Grafo de dependencia Nodos circulares con valor interno	*Fibonacci function (Introdución)*
P2	R. Sedgewick, ***Algorithms in Java***	Fig. 5.14 pág. 221	Traza de llamadas recursivas	*Fibonacci numbers*
		Fig. 5.14 pág. 221	Árbol de recursión Nodos internos circulares y externos, cuadrados Nodos con resultado interno	
		Fig. 5.17 pág. 224	Árbol de recursión para memorización Nodos internos circulares y externos, cuadrados Nodos con resultado interno	
P3	S. Skiena, ***The Algorithm Design Manual***	Fig. 8.1 pág. 275	Árbol de recursión Nodos sin marco, con identificador y valor Resultado en nodo raíz	*Fibonacci numbers*
		Fig. 8.2 pág. 276	Árbol de recursión para memorización Nodos sin marco, con identificador y valor Resultado en nodo raíz	
P4	William McAllister, ***Data Structures and Algorithms using Java.***		NO	Dynamic Programming Applied to Recursión, Fibonacci Sequence.
10.Otros Problemas				
	Libro	**Fig. y pág.**	**Visualización**	**Nomenclatura**
P1	G. Brassard y P. Bratley. ***Fundamentos de algoritmia***	Fig. 8.2 pág. 294	Árbol de recursión de 3 niveles (explicación) Nodos sin marco, con valores genéricos Parte suprimida (con "etc.")	El Campeonato Mundial
P2	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***		NO	El Campeonato Mundial (con y sin empate)
P3	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	pág. 409	Vectores que ilustran el cálculo de los valores de recurrencia. Se emplean flechas para indicar los elementos que intervienen Explicación Parte suprimida con espacio	Reparto de botín
P4	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***		NO	Reparto Almacenes
P5	N. Martí Oliet, Y. Ortega y J. A. Verdejo, ***Estructuras de datos y métodos algorítmicos: ejercicios resueltos***	Fig. 13.13 pág. 445	Tabla genérica (explicación) Etiqueta de tabla en esquina superior izq En ambas dimensiones, etiquetas de valores con puntos suspensivos Diagonal de celdas para representar orden de cómputo Dependencia entre celdas (flechas y sombreado de celda dependiente) Resto de celdas vacías	Vacas-pienso
P6	S. Skiena**, *The Algorithm Design Manual***	Fig. 8.8 pág. 297	Figura doble, en donde se muestran dos matrices para dos ejemplos de entrada. Se emplean etiquetas para columnas y filas.	The Partition Problem
P7	S. Skiena, ***The Algorithm Design***	Fig. 8.10 pág.	Figura doble, con representación dependiente	Minimum weight triangulation

	Manual	300	del dominio (polígonos: 2 soluciones)	
		Fig. 8.11 pág. 301	Representación dependiente del dominio (polígono: explicación) Etiquetas alfabéticas y líneas	
P8	William McAllister, ***Data Structures and Algorithms using Java***.	Fig. 6.14 pág. 328	Secuencia de pilas de control Para cada celda, etiqueta de índice fuera y valor dentro	Dynamic Programming Applied to recursión, n!

4 Conclusiones

En este informe se presenta una completa revisión bibliográfica en el ámbito de diseño y análisis de algoritmos sobre problemas resueltos mediante la técnica de programación dinámica. Estos problemas se han clasificado según los tipos de ejercicios encontrados entre la bibliografía consultada dando lugar a nueve subapartados definidos (cadenas de caracteres, camino mínimos, coeficientes binomiales, árboles, grafos, monedas, multiplicación de matrices, planificación, sucesión de Fibonacci) y un décimo subapartado con otros problemas de interés no clasificados (otros problemas).

En esta revisión bibliográfica se han tenido en cuenta de forma especial todas las ilustraciones que acompañan a cada problema, incluyendo aquellas de mayor relevancia en este informe así como pseudocódigos de las funciones de mayor interés. Todo ello hace que la clasificación realizada junto con la inclusión de ilustraciones pueda resultar de gran utilidad a la hora del diseño de ayudantes interactivos en los que se emplee la técnica de programación dinámica.

La elaboración de este informe deja abierto el camino de posibles trabajos futuros, bien con la ampliación de este mismo informe, aumentando la bibliografía consultada o incluyendo nuevos problemas tipo, así como utilizando los resultados obtenidos para ampliar la línea de trabajo sobre ayudantes interactivos, en la resolución de problemas empleando la técnica de programación dinámica.

Agradecimientos. Este trabajo se ha financiado con el proyecto TIN2011-29542-C02-01 del Ministerio de Innovación y Ciencia.

Referencias

1. M. H. Alsuwaiyel, 1999. Algorithms Design Techniques and Analysis, World Scientific
2. S. Baase y A. van Gelder, 2000. Computer Algorithms: Introduction to Design and Analysis
3. G. Brassard y P. Bratley, 1996. Fundamentos de algoritmia, Prentice-Hall
4. T. H. Cormen, C. E. Leiserson, R. L. Rivest y C. Stein, 2001. Introduction to Algorithms,The MIT Press, 2ª ed.
5. M. T. Goodrich y R. Tamassia, 2001. Data Structures and Algorithms in Java, John Wiley & Sons, 2ª ed.
6. E. Horowitz y S. Sahni, 1978. Fundamentals of Computer Algorithms, Pitman

7. R. C. T. Lee, S. S. Tseng, R. C. Chang e Y. T. Tsai, 2005. Introducción al diseño y análisis de algoritmos, McGraw-Hill
8. N. Martí Oliet, Y. Ortega y J. A. Verdejo, 2004. Estructuras de datos y métodos algorítmicos: ejercicios resueltos, Pearson
9. R. Neapolitan y K. Naimipour, 1997. Foundations of Algorithms, Jones and Bartlett
10. I.Parberry, 1995. Problems on Algorithms, Prentice Hall
11. S. Sahni, 2005. Data Structures, Algorithms, and Applications in Java, Silicon Press
12. R. Sedgewick, 2002. Algorithms in Java, Addison-Wesley
13. S. Skiena, 1998. The Algorithm Design Manual, Springer-Verlag
14. W. McAllister, Data Structures and Algorithms using Java.