Febrero-2019-Solucion.pdf

mike_

Algorítmica y Modelos de Computación

3º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería
Universidad de Huelva

Solución Febrero 2019

Ejercicio 1 (2 pts)

a) Para realizar la traza, es necesario hacer también la traza de Partition cada vez que se llame. Sin embargo, el código de Partition tiene un fallo que hace no pueda funcionar el algoritmo. Si se realiza su traza para la primera llamada:

A = {31, 23, 90, 0, 77, 52, 49, 87, 60, 15}, k = 7

1	2	3	4	5	6	7	8	9	10
31	23	90	0	77	52	49	87	60	15

Inicialmente, j no podrá avanzar puesto que 15 no es mayor que el pivote, que es 31. El iterador i tampoco podrá avanzar dado que, al encontrarse en la posición del pivote, 31 no es mayor que 31. Como no se han cruzado (el if), se intercambian:

i									j
31									15
15									31

Tras intercambiarse, se vuelve al comienzo del bucle (tanto i como j no se han modificado). J no podrá avanzar dado que ahora tiene como valor 31, que es el pivote, y, por tanto, no es mayor. La variable i sí podrá incrementar su valor mientras que el valor que direccione en el vector sea menor que 31, que sería hasta 90:

		i							j
15	23	90							31
15	23	31							90

J ya sí puede avanzar mientras direcciones a un elemento mayor que 31. Ahora es i la que no puede avanzar:

		i	j						
15	23	31	0	77	52	49	87	60	90
15	23	0	31	77	52	49	87	60	90

Una vez intercambiados, j vuelve a detenerse por contener el valor 31 e i avanza hasta la siguiente casilla, donde se detiene por el valor 31:

		i, j							
15	23	0	31	77	52	49	87	60	90

Este debería ser el punto de partición que devolviera el algoritmo, pero, dado que el código contiene "`mientras (j ≥ i)`", **el bucle no finaliza** ya que ninguno de los dos iteradores puede avanzar dado que sus condiciones de avance son que sean mayores o menores que el pivote (31), permaneciendo siempre en la misma casilla, cumpliéndose siempre la condición del bucle más externo.

Por tanto, hay que realizar la siguiente modificación en el algoritmo Partition para que pueda funcionar correctamente:

KEEP CALM AND ESTUDIA UN POQUITO

WUOLAH + #QuédateEnCasa

```
funcion Partition(A : vector, primero, ultimo : entero)
  piv = A[primero]
  i = primero
  j = ultimo
  mientras (j > i) hacer
    mientras (A[j] > piv) hacer
      j = j - 1
    fmientras
    mientras (A[i] < piv) hacer
      i = i + 1
    fmientras
    si (i < j) entonces
      tmp = A[j]
      A[j] = A[i]
      A[i] = tmp
    fsi
  fmientras
  devuelve j
ffuncion
```

Ahora ya se pueden realizar correctamente las trazas de los algoritmos:

Traza algoritmo iterativo:

```
SelectIT(A, 1, 10, 7)
  mientras (1 < 10)
  q = Partition(A, 1, 10)
```

1	2	3	4	5	6	7	8	9	10
31	23	90	0	77	52	49	87	60	15
15	23	0	31	77	52	49	87	60	90

```
  q ← 4
  7 ≤ 4 → NO
  primero = q + 1 (5)
  mientras (5 < 10)
  q = Partition(A, 5, 10)
```

5	6	7	8	9	10
77	52	49	87	60	90
60	52	49	77	87	90

```
  q ← 8
  7 ≤ 8 → SI
  ultimo = q
  mientras (5 < 8)
  q = Partition(A, 5, 8)
```

5	6	7	8
60	52	49	77
49	52	60	77

```
  q ← 7
  7 ≤ 7 → SI
  ultimo = q
  mientras (5 < 7)
  q = Partition(A, 5, 7)
```

5	6	7
49	52	60
49	52	60

```
  q ← 5
  7 ≤ 5 → NO
  primero = q + 1 (6)
  mientras (6 < 7)
  q = Partition(A, 6, 7)
```

6	7
52	60
52	60

```
  q ← 6
  7 ≤ 6 → NO
  primero = q + 1 (7)
  mientras (7 < 7) → FALSO
  60 ← devolver A[primero]
```

7
60

Traza algoritmo recursivo:

SelectRc(A, 1, 10, 7)
  1 == 10 → **NO**
  q = **Partition**(A, 1, 10)

	1	2	3	4	5	6	7	8	9	10
	31	23	90	0	77	52	49	87	60	15
	15	23	0	31	77	52	49	87	60	90

  q ← 4
  i = q - primero + 1 (4 - 1 + 1)
  7 ≤ 4 → **NO**
  SelectRc(A, 4+1, 10, 7-4)
    5 == 10 → **NO**
    q = **Partition**(A, 5, 10)

	5	6	7	8	9	10
	77	52	49	87	60	90
	60	52	49	77	87	90

    q ← 8
    i = q - primero + 1 (8 - 5 + 1)
    3 ≤ 4 → **SI**
    SelectRc(A, 5, 8, 3)
      5 == 7 → **NO**
      q = **Partition**(A, 5, 8)

	5	6	7	8
	60	52	49	77
	49	52	60	77

      q ← 7
      i = 7 - 5 + 1
      3 ≤ 3 → SI
      SelectRc(A, 5, 7, 3)
        5 == 7 → **NO**
        q = **Partition**(A, 5, 7)

	5	6	7
	49	52	60
	49	52	60

        q ← 5
        i = 5 - 5 + 1
        3 ≤ 1 → NO
        SelectRc(A, 5+1, 7, 3-1)
          6 == 7 → **NO**
          q = **Partition**(A, 6, 7)

	6	7
	52	60
	52	60

          q ← 6
          i = 6 - 6 + 1
          2 ≤ 1 → NO
          SelectRc(A, 6+1, 7, 2-1)
            7 == 7 → **SI**
            60 ← **devolver** A[primero]

	7
	60

Y los sucesivos return hasta la primera llamada.

b) Análisis complejidad algoritmo iterativo

Como no se especifica qué caso del algoritmo hay que analizar, es necesario entonces analizar los tres casos del algoritmo:

Caso peor:

El caso peor sería que, en cada iteración, Partition divide el vector en dos partes de tamaño 1 y n-1, siendo el elemento para buscar el último, lo que haría que haya que seguir realizando iteraciones.

Teniendo en cuenta que Partition dividirá siempre el vector en 1 y n-1, y que se trata del caso peor (se hace el mayor número de operaciones posible):

$T_{Part}(n) = 6 + \sum(3 + \sum(5) + 3 + \sum(5) + T_{Si}) + 1$

Para que sea el mayor número de operaciones posible y que el vector quede dividido en 1 y n-1, uno de los bucles internos nunca se cumplirá la condición (será para la partición de tamaño 1), el otro se cumplirá sólo una vez por iteración del bucle principal, y el 'si' final se cumplirá siempre. Sabiendo esto, podemos adaptar la función a:

$T_{Part} = 7 + \sum_{i=1}^{n}(2 + 3 + 5 + 8 + 1) \rightarrow T_{Part}(n) = 7 + \sum_{i=1}^{n}(19) \rightarrow$ **$T_{Part}(n) = 19n + 7 \in O(n)$.**

Es como si fuera un único bucle en el que, internamente, sólo se hace la comprobación de la i (primer bucle mientras), después se hace una única iteración del bucle de j, seguido del 'si'.

Entonces, el algoritmo de K-ésimo Menor Iterativo, en cada iteración reduciría en 1 el tamaño del vector que le pasa a Partition, finalizando cuando el vector sea de tamaño 1, por tanto:

$T(n) = 1 + 1 + \sum_{i=1}^{n}(1 + 1 + T_{Part}(i) + 6) + 1 \rightarrow T(n) = 3 + \sum_{i=1}^{n}(19i + 7 + 8) \rightarrow$
$T(n) = 3 + \sum_{i=1}^{n}(19i) + \sum_{i=1}^{n}(15) \rightarrow T(n) = 3 + 19 \cdot \sum_{i=1}^{n}(i) + 15(n - 1 + 1) \rightarrow$
$T(n) = 19 \cdot \frac{n(n+1)}{2} + 15n + 3 \rightarrow$ **$T(n) = \frac{19}{2}n^2 + \frac{49}{2}n + 3 \in O(n^2)$.**

Caso mejor:

Este sería que tras una primera llamada a Partition, se particione el vector de tal manera que, al igual que en el caso peor, sea de 1 y n-1, pero al revés. En lugar de encontrarse el elemento buscado en la partición de tamaño n-1, se encuentra en la de tamaño 1, que rompería la ejecución del bucle y finalizaría. Por tanto, sólo se compondrá de una única iteración con una llamada a Partition:

Hay que obtener una nueva función para Partition, ya que, aunque sean del mismo tamaño las particiones, esta vez no se cumplirá nunca el if, reduciendo el número de operaciones:

$T_{Part}(n) = 7 + \sum_{i=1}^{1}(2 + 3 + 2 + \sum_{j=1}^{n}(5)) \rightarrow T_{Part}(n) = 7 + \sum_{i=1}^{1}(7 + 5n) \rightarrow$ **$T_{Part}(n) = 5n + 14 \in O(n)$.**

Y el análisis para el K-ésimo Menor Elemento sabiendo que sólo es una única iteración:

$T(n) = 1 + 1 + \sum_{i=1}^{1}(1 + 1 + T_{Part}(n) + 5) + 1 \rightarrow T(n) = 3 + \sum_{i=1}^{1}(5n + 21) \rightarrow T(n) = 3 + 5 \cdot \sum_{i=1}^{1}(n) + 21 \rightarrow$
$T(n) = 5n + 24 \in O(n)$.

Caso medio:
Este sería para cualquier otro caso, en el que es igual de probable que las particiones sean de tamaño 1 y n-1, o de tamaño n/2, o cualquier otro.

Partition funcionaría de la siguiente manera: dividirá un vector v (de tamaño n) en dos partes de tamaño p y n-p, que sumarán siempre n (obvio). Podemos suponer como equiprobable que haya que realizar para cada elemento un intercambio o no, por tanto, se harán la mitad de los intercambios posibles (la mitad de los que se harían en el caso peor).

Quedaría entonces que, la suma del número de iteraciones totales de los bucles internos será n (o lo que es lo mismo, sería como un único bucle, al fin y al cabo, desde el punto de vista del número de operaciones que realizará), y que el 'si' se cumplirá la mitad de las veces, por tanto:

Para verse algo más claro, en cada iteración se harían $\frac{p}{n}$ por el número de operaciones de un bucle más $\frac{n-p}{n}$ por el número de operaciones del segundo bucle, seguido de las operaciones del if:

$T_{Part}(n) = 7 + \sum_{i=1}^{n} \left(\frac{p}{n}(2+5) + \frac{n-p}{n}(2+5) + \frac{1}{2}8 + 1\right) \rightarrow T_{Part}(n) = 7 + \sum_{i=1}^{n} \left(\frac{p+n-p}{n}(2+5) + \frac{1}{2}8 + 1\right) \rightarrow$
$T_{Part}(n) = 7 + \sum_{i=1}^{n} (2 + 5 + \frac{1}{2}8 + 1) \rightarrow T_{Part}(n) = 7 + \sum_{i=1}^{n} (12) \rightarrow$ **$T_{Part}(n) = 12n + 7 \in O(n)$.**

Nota. En caso de tener dudas o falta de tiempo, dado que tanto en el caso peor Partition tiene O(n), se puede suponer el caso medio como k·n, o bien como el mismo caso peor, dado que el caso medio estaría siempre contenido en él (siempre sería inferior).

Para K-ésimo Menor Elemento, su análisis será similar a los caso Mejor y Peor, pero se desconoce el número de iteraciones que podría realizar el algoritmo. Es igual de probable que sean 2 iteraciones o n iteraciones, dado que puede particionarse el vector en cualquier tamaño. Por tanto, se hará la media para la suma de los n posibles tamaños (1, 2, ..., n-1, n):

$T(n) = 2 + \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{i}(5 + T_{Part}(j))) + 2 \rightarrow T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{i}(5 + 12j + 7)) \rightarrow$
$T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}(\sum_{j=1}^{i}(12j + 12)) \rightarrow T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}(12 \cdot \sum_{j=1}^{i}(j) + 12(i - 1 + 1)) \rightarrow$
$T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}(12 \cdot \sum_{j=1}^{i}(j) + 12i) \rightarrow T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}\left(12\left(\frac{i(i+1)}{2}\right) + 12i\right) \rightarrow$
$T(n) = 4 + \frac{1}{n}\sum_{i=1}^{n}(6i^2 + 18i) \rightarrow T(n) = 4 + \frac{6}{n}\sum_{i=1}^{n}(i^2) + \frac{18}{n}\sum_{i=1}^{n}(i) \rightarrow$
$T(n) = 4 + \frac{6}{n} \cdot \frac{n(n+1)(2n+1)}{6} + \frac{18}{n} \cdot \frac{n(n+1)}{2} \rightarrow T(n) = 4 + (n + 1)(2n + 1) + 9 \cdot (n+1) \rightarrow$
$T(n) = 2n^2 + 12n + 14 \in O(n^2)$.

Es el valor que sale con esa suposición.

Fórmulas utilizadas:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \approx n^3/3$$

c) Análisis complejidad algoritmo recursivo

Aquí ya se utilizarán las funciones de complejidad de Partition ya calculadas.

Caso peor:

$T(n) = \begin{cases} 3 & si\ n = 1 \\ T(n-1) + 12 + 19n + 7 & si\ n > 1 \end{cases}$ →

$T(n) = \begin{cases} 3 & si\ n = 1 \\ T(n-1) + 19n + 19 & si\ n > 1 \end{cases}$

T(n) – T(n – 1) = 19n + 19 → No homogénea

T(n) – T(n – 1) → (x – 1)

Para pasar a la izquierda 19n + 19, convertir primero a: $b^n \cdot p(n)^d$ →

$19n + 19 = 1^n \cdot (19n^1 + 19)$ → b = 1, d = 0

Pasa a la izquierda como: $(x - b)^{d+1}$ → $(x - 1)^{1+1}$

$(x - 1)(x - 1)^2 = 0$ → r = 1 triple

$T(n) = c_0 \cdot 1^n \cdot n^0 + c_1 \cdot 1^n \cdot n^1 + c_2 \cdot 1^n \cdot n^2$ → $T(n) = c_0 + c_1 n + c_2 n^2$

Calcular valores de las constantes: tres constantes, tres ecuaciones

A partir del caso base: T(1) = 3 →

T(2) = T(1) + 19·2 + 19 = 3 + 57 = 60

T(3) = T(2) + 19·3 + 19 = 60 + 76 = 136

T(4) = T(3) + 19·4 + 19 = 136 + 95 = 231

$\begin{cases} c_0 + 2c_1 + 4c_2 = 60 \\ c_0 + 3c_1 + 9c_2 = 136 \\ c_0 + 4c_1 + 16c_2 = 231 \end{cases}$ → $c_0 = -35$, $c_1 = \frac{57}{2}$, $c_2 = \frac{19}{2}$

$T(n) = \frac{19}{2}n^2 + \frac{57}{2}n - 35 \in O(n^2)$.

Caso mejor:

Sólo una llamada recursiva:

$T(n) = \begin{cases} 3 & si\ n = 1 \\ T(1) + 9 + 5n + 14 & si\ n > 1 \end{cases}$ →

$T(n) = \begin{cases} 3 & si\ n = 1 \\ T(1) + 5n + 23 & si\ n > 1 \end{cases}$

T(n) = T(1) + 5n + 23 → T(n) = 3 + 5n + 23 → **T(n) = 5n + 26 ∈ O(n).**

Caso medio:

Son equiprobables todos los posibles tamaños de partición, por tanto:

$T(n) = \begin{cases} 3 & si\ n = 1 \\ \frac{1}{n}\sum_{i=1}^{n-1}(T(i)) + 11 + 12n + 7 & si\ n > 1 \end{cases}$ →

$T(n) = \begin{cases} 3 & si\ n = 1 \\ \frac{1}{n}\sum_{i=1}^{n-1}(T(i)) + 12n + 18 & si\ n > 1 \end{cases}$

$T(n) = \frac{1}{n-1}\sum_{i=1}^{n-1}(T(i)) + 12n + 18$ → Sacamos el caso base del sumatorio →

$T(n) = \frac{1}{n-1}\sum_{i=2}^{n-1}(T(i)) + \frac{T(1)}{n-1} + 12n + 18$

Si tomamos como n igual al primer valor que no fuera caso base, en este caso, n = 2:

n = 2 → T(2) = $\frac{1}{2-1}\sum_{i=2}^{2-1}(T(i)) + \frac{T(1)}{2-1}$ + 12·2 + 18 → T(2) = $\frac{T(1)}{2-1}$ + 42 → T(2) = T(1) + 42 = 45

Si definimos k = 45, podremos ver como para todo n ≥ 2, k·n ≥ T(n):

n = 2, 45 ≤ 45·2 = 90 → Diferencia de 45

n = 3 → T(3) = $\frac{1}{3-1}\sum_{i=2}^{3-1}(T(i)) + \frac{T(1)}{3-1}$ + 12·3 + 18 → T(3) = $\frac{T(2)}{3-1} + \frac{T(1)}{3-1}$ + 54 = 22.5 + 1.5 + 54 = 78

78 ≤ 45·3 = 135 → Diferencia de 57

n = 4 → T(4) = $\frac{1}{4-1}\sum_{i=2}^{4-1}(T(i)) + \frac{T(1)}{4-1}$ + 12·4 + 18 → T(4) = $\frac{T(2)+T(3)}{4-1} + \frac{T(1)}{4-1}$ + 66 = 30.75 + 1 + 66 = 97.75

98 ≤ 45·4 = 180 → Diferencia de 82

Como se puede ver, cuanto más aumenta n su valor, más se separan T(n) de 49n, quedando 49n siempre por encima, por tanto, podemos decir que T(n) ∈ 49n, o lo que es lo mismo, **T(n) ∈ O(n).**

Ejercicio 2 (1.5 pts)

a) Procedimiento directo:

Este queda reservado.

Divide y vencerás:

```
funcion FibonacciDyV(n : entero)
  si n == 1 OR n == 2 // en pseudocódigo empieza en 1
    devuelve 1
  fsi
  devuelve FibonacciDyV(n-1) + FibonacciDyV(n-2)
ffuncion
```

Programación dinámica:

```
funcion Fibonacci(n : entero)
  F = (0₁, ... 0ₙ) // Vector de ceros de tamaño n
  F(1) = 1
  F(2) = 1
  i = 2
  mientras i ≤ n
    F(i) = F(i-1) + F(i-2)
    i = i + 1
  fmientras
  devuelve F(n)
ffuncion
```

b) Estimación de los órdenes de complejidad:

Procedimiento directo: queda reservado.

Divide y vencerás:

$T(n) = \begin{cases} 2 & si\ n \leq 2 \\ T(n-1) + T(n-2) + 9 & si\ n > 2 \end{cases}$

T(n) = T(n – 1) + T(n – 2) + 9 → T(n) – T(n – 1) – T(n – 2) = 9 → No homogénea

T(n) – T(n – 1) – T(n – 2) → $x^2 - x - 1$

$9 = b^n \cdot p(n)^d$ → b = 1, d = 0 → $(x - b)^{d+1}$ → $(x - 1)^{0+1}$

$(x^2 - x - 1)(x - 1) = 0$ → $r_1 = 1$ simple, $r_2 = \frac{1+\sqrt{5}}{2}$ simple, $r_3 = \frac{1-\sqrt{5}}{2}$ simple

$T(n) = c_0 \cdot 1^n \cdot n^0 + c_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n \cdot n^0 + c_2 \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n \cdot n^0$ → $T(n) = c_0 + c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$

T(3) = T(2) + T(1) + 9 = 2 + 2 + 9 = 13

T(4) = T(3) + T(2) + 9 = 13 + 2 + 9 = 24

T(5) = T(4) + T(3) + 9 = 24 + 13 + 9 = 46

$$\begin{cases} c_0 + \left(\frac{1+\sqrt{5}}{2}\right)^3 c_1 + \left(\frac{1-\sqrt{5}}{2}\right)^3 c_2 = 13 \\ c_0 + \left(\frac{1+\sqrt{5}}{2}\right)^4 c_1 + \left(\frac{1-\sqrt{5}}{2}\right)^4 c_2 = 24 \\ c_0 + \left(\frac{1+\sqrt{5}}{2}\right)^5 c_1 + \left(\frac{1-\sqrt{5}}{2}\right)^5 c_2 = 46 \end{cases} \rightarrow \begin{cases} c_0 + 4.24c_1 - 0.24c_2 = 13 \\ c_0 + 6.85c_1 + 0.15c_2 = 24 \\ c_0 + 11.09c_1 - 0.09c_2 = 46 \end{cases}$$ → $c_0 = -9$, $c_1 = \frac{187}{38}$, $c_2 = -\frac{539}{114}$

Como c_1 es un valor positivo, entonces $\mathbf{T(n) \in O\left(\frac{1+\sqrt{5}}{2}\right)^n}$

Programación dinámica:

T(n) = 5 + 1 + 1 + $\sum_{i=2}^{n}(2+9)$ + 1 → T(n) = 8 + $\sum_{i=2}^{n}(11)$ → T(n) = 11(n-2+1) + 8 →

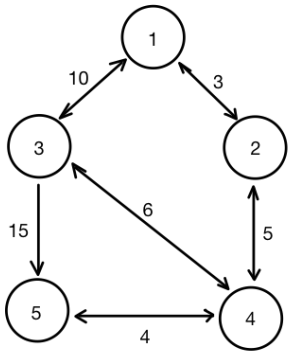
T(n) = 11n -3 ∈ O(n).

Claramente, la versión en programación dinámica es más rápida que la versión basada en DyV:

O(n) ∈ O$\left(\frac{1+\sqrt{5}}{2}\right)^{n}$

Ejercicio 3 (3 pts)

a)



C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	∞	∞	1	-	1	1		
2	3	-	∞	5	∞	2	2	-		2	
3	10	∞	-	6	15	3	3		-	3	3
4	∞	5	6	-	4	4		4	4	-	4
5	∞	∞	∞	4	-	5				5	-

k = 1, i = 2, j = 3 → C(2,3) < C(2,1) + C(1,3) – SI → C(2,3) = 13, D(2,3) = 1
k = 1, i = 2, j = 4→ C(2,4) < C(2,1) + C(1,4) – NO
k = 1, i = 2, j = 5→ C(2,5) < C(2,1) + C(2,5) – NO

k = 1, i = 3, j = 2 → C(3,2) < C(3,1) + C(1,2) – SI → C(3,2) = 13, D(3,2) = 1
k = 1, i = 3, j = 4 → C(3,4) < C(3,1) + C(1,4) – NO
k = 1, i = 3, j = 5 → C(3,5) < C(3,1) + C(1,5) – NO

k = 1, i = 4, j = 2 → C(4,2) < C(4,1) + C(1,2) – NO y ya que C(4,1) = ∞, se salta a la siguiente i

k = 1, i = 5, j = 2 → C(5,2) < C(5,1) + C(1,2) – NO y ya que C(5,1) = ∞, se finaliza k = 1, y se pasa a k = 2

C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	∞	∞	1	-	1	1		
2	3	-	13	5	∞	2	2	-	1	2	
3	10	13	-	6	15	3	3	1	-	3	3
4	∞	5	6	-	4	4		4	4	-	4
5	∞	∞	∞	4	-	5				5	-

k = 2, i = 1, j = 3 → C(1,3) < C(1,2) + C(2,3) – NO
k = 2, i = 1, j = 4 → C(1,4) < C(1,2) + C(2,4) – SI → C(1,4) = 8, D(1,4) = 2
k = 2, i = 1, j = 5 → C(1,5) < C(1,2) + C(2,5) – NO

k = 2, i = 3, j = 1 → C(3,1) < C(3,2) + C(2,1) – NO
k = 2, i = 3, j = 4 → C(3,4) < C(3,2) + C(2,4) – NO
k = 2, i = 3, j = 5 → C(3,5) < C(3,2) + C(2,5) – NO

k = 2, i = 4, j = 1 → C(4,1) < C(4,2) + C(2,1) – SI → C(4,1) = 8, D(4,1) = 2
k = 2, i = 4, j = 3 → C(4,3) < C(4,2) + C(2,3) – NO
k = 2, i = 4, j = 5 → C(4,5) < C(4,2) + C(2,5) – NO

Algorítmica y Modelos de Computación Solución Febrero 2019

k = 2, i = 5, j = 1 → C(5,1) < C(5,2) + C(2,1) – NO y ya que C(5,2) = ∞, se finaliza k = 2 y se salta a k = 3

C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	8	∞	1	-	1	1	2	
2	3	-	13	5	∞	2	2	-	1	2	
3	10	13	-	6	15	3	3	1	-	3	3
4	8	5	6	-	4	4	2	4	4	-	4
5	∞	∞	∞	4	-	5				5	-

k = 3, i = 1, j = 2 → C(1,2) < C(1,3) + C(3,2) – NO
k = 3, i = 1, j = 4 → C(1,4) < C(1,3) + C(3,4) – NO
k = 3, i = 1, j = 5 → C(1,5) < C(1,3) + C(3,5) – SI → C(1,5) = 25, D(1,5) = 3

k = 3, i = 2, j = 1 → C(2,1) < C(2,3) + C(3,1) – NO
k = 3, i = 2, j = 4 → C(2,4) < C(2,3) + C(3,4) – NO
k = 3, i = 2, j = 5 → C(2,5) < C(2,3) + C(3,5) – SI → C(2,5) = 28, D(2,5) = 3

k = 3, i = 4, j = 1 → C(4,1) < C(4,3) + C(3,1) – NO
k = 3, i = 4, j = 2 → C(4,2) < C(4,3) + C(3,2) – NO
k = 3, i = 4, j = 5 → C(4,5) < C(4,3) + C(3,5) – NO

k = 3, i = 5, j = 1 → C(5,1) < C(5,3) + C(3,1) – NO, y ya que C(5,3) = ∞, se finaliza k = 3 y se salta a k = 4

C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	8	25	1	-	1	1	2	3
2	3	-	13	5	28	2	2	-	1	2	3
3	10	13	-	6	15	3	3	1	-	3	3
4	8	5	6	-	4	4	2	4	4	-	4
5	∞	∞	∞	4	-	5				5	-

k = 4, i = 1, j = 2 → C(1,2) < C(1,4) + C(4,2) – NO
k = 4, i = 1, j = 3 → C(1,3) < C(1,4) + C(4,3) – NO
k = 4, i = 1, j = 5 → C(1,5) < C(1,4) + C(4,5) – SI → C(1,5) = 12, D(1,5) = 4

k = 4, i = 2, j = 1 → C(2,1) < C(2,4) + C(4,1) – NO
k = 4, i = 2, j = 3 → C(2,3) < C(2,4) + C(4,3) – SI → C(2,3) = 11, D(2,3) = 4
k = 4, i = 2, j = 5 → C(2,5) < C(2,4) + C(4,5) – SI → C(2,5) = 9, D(2,5) = 4

k = 4, i = 3, j = 1 → C(3,1) < C(3,4) + C(4,1) – NO
k = 4, i = 3, j = 2 → C(3,2) < C(3,4) + C(4,2) – SI → C(3,2) = 11, D(3,2) = 4
k = 4, i = 3, j = 5 → C(3,5) < C(3,4) + C(4,5) – SI → C(3,5) = 10, D(3,5) = 4

k = 4, i = 5, j = 1 → C(5,1) < C(5,4) + C(4,1) – SI → C(5,1) = 12, D(5,1) = 4
k = 4, i = 5, j = 2 → C(5,2) < C(5,4) + C(4,2) – SI → C(5,2) = 9, D(5,2) = 4
k = 4, i = 3, j = 3 → C(5,3) < C(5,4) + C(4,3) – SI → C(5,3) = 10, D(5,3) = 4

C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	8	12	1	-	1	1	2	4
2	3	-	11	5	9	2	2	-	4	2	4
3	10	11	-	6	10	3	3	4	-	3	4
4	8	5	6	-	4	4	2	4	4	-	4
5	12	9	10	4	-	5	4	4	4	5	-

k = 5, i = 1, j = 2 → C(1,2) < C(1,5) + C(5,2) – NO
k = 5, i = 1, j = 3 → C(1,3) < C(1,5) + C(5,3) – NO
k = 5, i = 1, j = 4 → C(1,4) < C(1,5) + C(5,4) – NO
k = 5, i = 2, j = 1 → C(2,1) < C(2,5) + C(5,1) – NO
k = 5, i = 2, j = 3 → C(2,3) < C(2,5) + C(5,3) – NO
k = 5, i = 2, j = 4 → C(2,4) < C(2,5) + C(5,4) – NO

k = 5, i = 3, j = 1 → C(3,1) < C(3,5) + C(5,1) – NO
k = 5, i = 3, j = 2 → C(3,2) < C(3,5) + C(5,2) – NO
k = 5, i = 3, j = 4 → C(3,4) < C(3,5) + C(5,4) – NO

k = 5, i = 4, j = 1 → C(4,1) < C(4,5) + C(5,1) – NO
k = 5, i = 4, j = 2 → C(4,2) < C(4,5) + C(5,2) – NO
k = 5, i = 4, j = 3 → C(4,3) < C(4,5) + C(5,3) – NO

C	1	2	3	4	5	D	1	2	3	4	5
1	-	3	10	8	12	1	-	1	1	2	4
2	3	-	11	5	9	2	2	-	4	2	4
3	10	11	-	6	10	3	3	4	-	3	4
4	8	5	6	-	4	4	2	4	4	-	4
5	12	9	10	4	-	5	4	4	4	5	-

b)

Algoritmo para distancia más corta entre dos vértices:

```
funcion distancia(i, j : enteros)
  devuelve C(i,j)
ffuncion
```

Distancia entre los vértices 1 y 5: `distancia(1, 5)` → 12

Algoritmo para ruta asociada al camino mínimo entre dos vértices:

```
funcion caminoMinimo(i, j : enteros)
  S ← () // lista vacía
  x = j
  S.insertarAlPrincipio(x)
  mientras x ≠ i
    x = D(i,x)
    S.insertarAlPrincipio(x)
  fmientras
  devuelve S
ffuncion
```

Camino mínimo entre los vértices 1y 5: `caminoMinimo(1, 5)` → 1 - 2 - 4 - 5

Ejercicio 4 (1.5 pts)

a) Es igual que el problema de la mochila0-1, excepto en que se puede fraccionar por la mitad los objetos si no caben.

Dado que se permite este grado de flexibilidad a la hora de meter objetos, el criterio utilizado será introducir aquel objeto que tenga una mayor relación beneficio/peso, puesto que, al poder partir los objetos, el nivel de aprovechamiento de la capacidad de la mochila podría ser mayor:

Candidatos

```
funcion mochila(P(1...N), B(1...N) : enteros, M : entero)
  S(1...N) = (0, ... 0) // Conjunto solución: admite valores 0, 1 o 1/2
  pesoAct = 0
  ordenados(1...N) = ordenar(P, B) // Ordena los artículos en función del
                                   // beneficio/peso, devolviendo un
                                   // array con los índices de las
                                   // posiciones, de mayor a menor
  i = 1
  mientras (i ≤ N Y pesoAct ≤ M)
    pos = ordenados(i) // articulo con mayor b/p en cada iteración
    peso = P(pos)
    si (pesoAct + peso ≤ M) // probamos primero si cabe entero
      S(pos) = 1
      pesoAct = pesoAct + peso
    sino
      si (pesoAct + peso/2 ≤ M) // probamos si cabe la mitad del objeto
        S(pos) = 1/2
        pesoAct = pesoAct + peso/2
      fsi
    fsi
    i = i + 1
  fmientras
  devuelve S
ffuncion
```

Criterio →

Bucle voraz

Una estimación del tiempo de ejecución del algoritmo sería:

- Inicializar las variables sería coste O(1).
- Ordenar, podría estar basado en algún método de ordenación rápida como QuickSort o MergeSort. La peor implementación posible sería que hubiera un segundo vector con los índices, y se hagan el doble de intercambios (en el de valores y en el de índices), lo que dejaría que su coste fuera 2·n·log(n), que sigue perteneciendo a O(n·log(n)).
- El bucle mientras, en el peor caso, recorrería todos los objetos, y para cada objeto, intentaría introducir la mitad (significa que entero no cabía). El cuerpo del bucle es, en el peor caso O(1), y si se repite N veces (N objetos), el coste del bucle sería de N·O(1) que es igual a O(n).

El coste final del algoritmo sería la suma de todas sus partes, que son secuenciales, haciendo que sea O(n·log(n)).

b) Aplicación del algoritmo para: n = 2, M = 5, p = (8, 5), b = (10, 6)

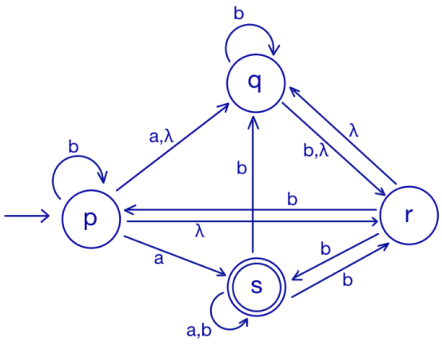
S = (0, 0)
pesoAct = 0
ordenados = ordenar((8, 5), (10, 6)) → (10/8, 6/5)
ordenados = (1, 2) // Hay que recordar que se ordenan los índices y no los objetos en sí
pos = 1, peso = 8, pesoAct + 8 ≤ M? – NO
pos = 1, peso = 8/2, pesoAct + 4 ≤ M? – SI → S = (1/2, 0)
pos = 2, peso = 5, pesoAct + 5 ≤ M? – NO
pos = 2, peso = 5/2, pesoAct + 2.5 ≤ M? – NO

La solución obtenida es: S = (1/2, 0), con beneficio (1/2)·10 = 5.

Sin embargo, no es la solución óptima, la cual sería: S = (0, 1), con beneficio = 6.

Ejercicio 5 (2 pts)

	a	b	λ
→ p	{q, s}	{p}	{q, r}
q		{q, r}	{r}
r		{p, s}	{q}
* s	{s}	{q, r, s}	



a) AFD equivalente:

Q_0 = λ-Clausura(p) = {p, q, r}

f'(Q_0, a) = {q, s} → λ-Clausura({q, s}) = {q, s, r} → Nuevo estado: Q_1 *

f'(Q_0, b) = {p, q, r, s} → λ-Clausura({p, q, r, s}) = {p, q, r, s} → Nuevo estado: Q_2 *

f'(Q_1, a) = {s} → λ-Clausura({s}) = {s} → Nuevo estado: Q_3 *

f'(Q_1, b) = {q, r, s, p} → λ-Clausura({q, r, s, p}) = {q, r, s, p} = Q_2

f'(Q_2, a) = {q, s} → λ-Clausura({q, s}) = {q, s, r} = Q_1

f'(Q_2, b) = {p, q, r, s} → λ-Clausura({p, q, r, s}) = Q_2

f'(Q_3, a) = {s} → λ-Clausura({s}) = {s} = Q_3

f'(Q_3, b) = {q, r, s} → λ-Clausura({q, r, s}) = {q, r, s} = Q_1

	a	b
→ Q_0	Q_1	Q_2
* Q_1	Q_3	Q_2
* Q_2	Q_1	Q_2
* Q_3	Q_3	Q_1

b) AFD mínimo:

Agrupamos inicialmente en finales y no finales:

Q/E_0 = (c_0 = {Q_0}, c_1 = {Q_1, Q_2, Q_3})

f'(Q_0, a) = Q_1 ∈ c_1, f'(Q_0, b) = Q_2 ∈ c_1

f'(Q_1, a) = Q_3 ∈ c_1 f'(Q_1, b) = Q_2 ∈ c_1

f'(Q_2, a) = Q_1 ∈ c_1 f'(Q_2, b) = Q_2 ∈ c_1

f'(Q_3, a) = Q_3 ∈ c_1 f'(Q_2, b) = Q_1 ∈ c_1

No es necesario seguir dividiendo en nuevos conjuntos, el AFD mínimo es:

	a	b
→ c_0	c_1	c_1
* c_1	c_1	c_1

c) Gramática regular (tipo 3) equivalente al AFD del apartado anterior:
G = ({c_0, c_1}, {a, b}, c_0, P), siendo P:
- c_0 ::= ac_1 | bc_1
- c_1 ::= ac_1 | bc_1

d) Expresión regular equivalente al AFD del apartado b:
Ecuaciones características del AFD: $\begin{cases} x_0 = ax_1 + bx_1 + a + b \\ x_1 = ax_1 + bx_1 + a + b \end{cases}$

Para obtener el valor de la expresión regular, hay que sacar el valor de x0. Para sacar el valor de x0, es necesario obtener primero el valor de x1:

x_1 = ax_1 + bx_1 + a + b → Regla de Inferencia → x_1 = (a + b)*(a + b)

x_0 = ax_1 + bx_1 + a + b → x_0 = a[(a + b)*(a + b)] + b[(a + b)*(a + b)] + a + b →
x_0 = a(a + b)*(a + b) + b(a + b)*(a + b) + a + b