

# Algoritmos y Estructuras de Datos III

Departamento de Computación  
Facultad de Ciencias Exactas y Naturales  
Universidad de Buenos Aires

## Trabajo Práctico 2

Integrante	LU	Correo electrónico
Laura Muiño	399/11	mmuino@dc.uba.ar
Martín Santos	413/11	martin.n.santos@gmail.com
Luis Toffoletti	827/11	luis.toffoletti@gmail.com
Florencia Zanollo	934/11	florenciazanollo@hotmail.com

**Reservado para la cátedra**

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

## Índice

<b>1. Ejercicio 1</b>	<b>3</b>
1.1. Descripción: . . . . .	3
1.2. Pseudocódigo: . . . . .	3
1.3. Complejidad: . . . . .	4
<b>2. Ejercicio 2.1</b>	<b>6</b>
<b>3. Ejercicio 2.2</b>	<b>6</b>
<b>4. Ejercicio 3</b>	<b>6</b>

## 1. Ejercicio 1

### 1.1. Descripción:

En este ejercicio debemos encontrar el costo mínimo de un conjunto de impresiones, dado un grupo de trabajos.

Para realizar la impresión contamos con dos máquinas iguales donde podemos repartir los trabajos, estos tienen un costo particular según quién fué el trabajo previo o si son el primer trabajo de la máquina.

Además se debe cumplir un orden en las impresiones, el trabajo  $i$  no puede ir antes que el  $j$  si  $i > j$ .

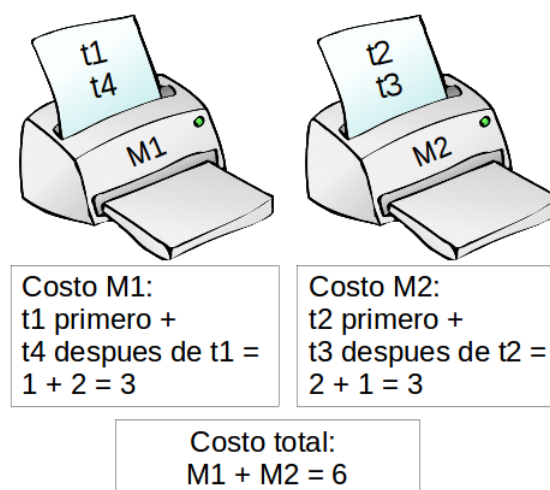
Es decir, los trabajos se pueden repartir entre las máquinas de muchas maneras distintas, pero para calcular el costo debe tenerse en cuenta el orden mencionado anteriormente.

Es nuestra meta encontrar la forma de repartirlos que abarate el costo total de las impresiones.

Por ejemplo si contamos con estos trabajos  $i$ , con sus respectivos precios respecto al trabajo anterior  $j$ :

$t_i \backslash t_j$	0	1	2	3
1	1	x	x	x
2	2	4	x	x
3	6	3	1	x
4	3	2	5	4

Entonces la solución con costo mínimo sería:



### 1.2. Pseudocódigo:

*Nota:*  $agregar\_a\_maquina(j)$  modifica un entero que depende del  $agregar\_a\_maquina(j - 1)$  pero sigue siendo  $O(1)$  ya que accede una vez a la matriz, realiza una comparación de enteros y modifica una variable entera.

**Algorithm 1** Impresiones ordenadas

---

```

1: procedure DIVIDIR TRABAJOS( $n, Costos$ )  ▷  $n$ = cant trabajos,  $Costos$ = costo de cada trabajo respecto a cada
   posible anterior
2:   Inicializo una matriz  $dp$ , tamaño  $n \times n$ 
3:    $dp[0][1] \leftarrow Costos(1, 0)$   ▷ en la posición (0,1) de la matriz guardo el valor de hacer el trabajo uno primero
4:    $agregar\_a\_maquina(1)$   ▷ guardo en qué máquina agregué el trabajo
5:   for  $j \leftarrow 2, n$  do  ▷ el trabajo 1 ya lo agregué
6:     for  $i \leftarrow 0, j - 1$  do
7:       if  $i = j - 1$  then
8:          $dp[i][j] \leftarrow minimo(dp[i][k] + Costos(j, k), \forall 0 \leq k \leq j - 2)$ 
9:          $agregar\_a\_maquina(j)$ 
10:      else
11:         $dp[i][j] \leftarrow (dp[i][j - 1] + Costos(j, j - 1))$ 
12:         $agregar\_a\_maquina(j)$ 
13:      end if
14:    end for
15:  end for
16:  return  $minimo(dp[i][n], \forall 0 \leq i \leq n - 1)$ 
17:  ▷ retorno el mínimo de la última columna, además de la combinación para lograr ese mínimo
18: end procedure

```

---

**1.3. Complejidad:**

Para analizar la complejidad voy a intentar representar cómo 'funciona' el algoritmo con una entrada de  $n$  trabajos. La idea de esto es ver qué cosas calcula y de qué manera, para poder ver cuántas operaciones realiza en total.

Me pareció mejor esta forma en vez de un análisis línea por línea ya que se ve más detalladamente cómo actúa el algoritmo. Además para no mezclar tantas cosas y que sea más legible voy a olvidarme de las partes de  $agregar\_a\_maquina(j)$  (Líneas 4, 9 y 12 del pseudocódigo), esto obviamente no influye en la complejidad total ya que cada operación de ese tipo es  $O(1)$ .

En principio se crea una matriz  $n \times n$  y se modifica la posición (0,1) con el costo del trabajo 1. Esto es  $O(n^2) + O(1) = O(n^2)$ <sup>1</sup> (Líneas 2 y 3).

La posición (0,1) se calcula en  $O(1)$ :

$i \backslash j$	1	2	3	4	...	n
0	$O(1)$					
1	x					
2	x	x				
3	x	x	x			
...	x	x	x	x		
n-1	x	x	x	x	x	

<sup>2</sup>

Luego, ingresamos en los For (Líneas 5 y 6). El algoritmo recorre las posiciones en blanco indicadas en el gráfico (la 'mitad' de la matriz si trazamos una línea en diagonal) y para cada posición hace lo siguiente:

<sup>1</sup>La matriz está implementada como vector de vectores

<sup>2</sup>Los casilleros en blanco son los que faltan calcular y los que contienen una x nunca se calculan porque el For de la Línea 6 va desde  $i=0$  hasta  $j-1$ .

Si  $i = j - 1$  :

El algoritmo recorre las posiciones válidas de la columna anterior a la que se encuentra procesando, y busca el mínimo entre ellas (sumándole el costo de hacer  $j$ , Línea 8).

Por lo tanto se debe encontrar el mínimo de  $j-1$  posiciones, esto es  $O(j-1)$ .

Si  $i \neq j - 1$  :

La posición  $(i,j)$  de la matriz necesita sólo la posición  $(i,j-1)$  para ser calculada, esto es  $O(1)$  ya que se accede a la matriz y se suman enteros (Línea 11).

En resumen si  $i = j - 1$  es  $O(j-1)$  sino es  $O(1)$ :

$i \backslash j$	1	2	3	4	...	n
0	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
1	x	$O(j-1)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
2	x	x	$O(j-1)$	$O(1)$	$O(1)$	$O(1)$
3	x	x	x	$O(j-1)$	$O(1)$	$O(1)$
...	x	x	x	x	$O(j-1)$	$O(1)$
n-1	x	x	x	x	x	$O(j-1)$

El algoritmo entonces recorre  $\sum_{j=2}^n (j-1) + n$  posiciones donde  $\sum_{j=2}^n (j-1)$  de ellas las resuelve en  $O(1)$  y el resto en  $O(j-1)$ , que como  $j$  es a lo sumo  $n$   $O(j-1)=O(n)$ .

Puedo decir entonces que la complejidad final es hacer  $\sum_{j=2}^n (j-1)$  veces  $O(1)$  +  $n$  veces  $O(n)$ .<sup>3</sup>

Ahora  $\sum_{j=2}^n (j-1) = \frac{n^2-n}{2}$  por lo tanto es  $O(n^2)$

Complejidad del ciclo:  $O(n^2) * O(1) + O(n) * O(n) = O(n^2) * O(n^2) = O(2 * n^2) = O(n^2)$

La línea 16 es  $O(n)$  (recorre la última columna de la matriz).

**Entonces, complejidad total:**  $O(n^2) + O(n^2) + O(n) = O(n^2)$

<sup>3</sup>Disculpen el abuso de notación

2. Ejercicio 2.1

3. Ejercicio 2.2

4. Ejercicio 3