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

1. Ejercicio 1

1.1. Descripción:

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

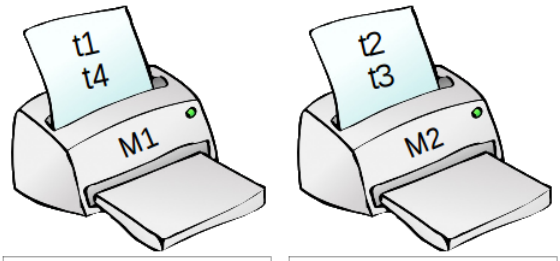
Para realizar la impresión contamos con dos máquinas iguales donde podemos repartir los trabajos, estos tienen un costo particular según quién fué el trabajo previo o si son el primer trabajo de la máquina.
Además se debe cumplir un orden en las impresiones, el trabajo i no puede ir antes que el j si 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:

ti \ tj	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:



Costo M1:
t1 primero +
t4 despues de t1 =
1 + 2 = 3

Costo M2:
t2 primero +
t3 despues de t2 =
2 + 1 = 3

Costo total:
M1 + M2 = 6

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*n
3:     dp[0][1]← 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 ← 2, n do   ▷ el trabajo 1 ya lo agregué
6:         for i ← 0, j − 1 do
7:             if i = j − 1 then
8:                 dp[i][j] ← minimo(dp[i][k] + Costos(j, k), ∀ 0 ≤ k ≤ j − 2)
9:                 agregar_a_maquina(j)   ▷ Costos(j, k) es el costo de hacer el trabajo j después del k
10:            else
11:                dp[i][j] ← (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], ∀ 0 ≤ i ≤ 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 linea por linea ya que se ve más detalladamente cómo actua 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 matríz $n * n$ y se modifica la posición (0,1) con el costo del trabajo 1. Esto es $O(n^2)$ + O(1)= $O(n^2)$[1] (Líneas 2 y 3).

La posición (0,1) se calcula en O(1):

i \ 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	

[2]

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 trazaramos una linea en diagonal) y para cada posición hace lo siguiente:

[1] La matríz está implementada como vector de vectores

[2] 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 (sumandole 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 \ 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 $O(n)$.[3]

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: $O(n^2) \ + \ O(n^2) \ + \ O(n) = O(n^2)$

1.4. Demostración:

La idea del algoritmo es sacar el costo según quién es el último trabajo de cada máquina (en vez de realizar todas las combinaciones) e ir guardando estos datos en una matriz para reutilizarlos más adelante.

Idea: dp(i,j)= mínimo costo para que una máquina termine en i y otra en j.

El algoritmo está definido por la siguiente función, con i<j [4]:

$$dp(i,j) = \begin{cases} dp(i, j-1) + Costos(j, j-1) & si \quad i \neq j-1 \\ \\ minimo(\ dp(k,i) \ + Costos(j,k) \ \forall 0 \leq k \leq j-2\ ) & si \quad i = j-1 \end{cases}$$

Como i<j e i≠j-1 ⇒ i<j-1, entonces para que una máquina termine en i y otra en j, j debe venir después de j-1 ya que j-1 no puede ir antes que i ni de ningún otro que no sea j (j-1 es el más grande distinto de j).

[3] Disculpen el abuso de notación
[4] para no sacar dos veces los valores, las máquinas son iguales por lo tanto hacer (i,j)=(j,i)

Si i=j-1 puedo combinar los trabajos de muchas maneras y j puede venir después de 0,1,... j-2. Además el costo de hacer j después de cada uno de ellos puede variar mucho, entonces debo ver todas esas combinaciones y luego puedo quedarme con la mínima que es la que me interesa.

Para demostrar la correctitud voy a probar que dada una posición (i,j) de la matriz, tengo en ella el mínimo costo de todas las combinaciones de forma que en una máquina el último trabajo sea i y en la otra sea j.

Hipótesis inductiva:
Dada una columna j de la matriz, en cada posición (i,j) con $0 \leq i < j$, tengo el mínimo costo de todas las combinaciones de forma que en una máquina el último trabajo sea i y en la otra sea j. De ahora en más lo voy a llamar mínimo(i,j).

Caso base:
El caso de j=1 es trivial y se encuentra fuera de los For (dónde implemento la función explicada más arriba), es un caso muy particular así que mi caso base será j=2.

Quiero ver que $\forall\ 0 \leq i < j$, dp(i,2) es mínimo(i,2).
Lo divido en casos:

Si i=0 ($i \neq j-1$): dp(0,2) = una máquina no tiene ningún trabajo y la otra termina en t2, es decir, tiene a t1 y t2. Como existe una única forma de hacer esta combinación el resultado es mínimo(0,2).

Si i=1 (i=j-1): dp(1,2) = una máquina tiene al t1 y otra al t2, luego existe una única forma de hacer esto, entonces es mínimo(1,2).

Paso inductivo:
Supongo que $\forall\ 0 \leq k < j$, vale la H.I., quiero ver que vale para j.
Lo divido en casos:

Si $i \neq j-1$: dp(i,j) =[5] dp(i,j-1) + Costos(j,j-1)
Como j-1<j entonces dp(i,j-1) es mínimo(i,j-1) por H.I. y Costos(j,j-1) es un valor único $\Rightarrow$ dp(i,j-1) + Costos(j,j-1) es mínimo(i,j).

Si i=j-1: dp(i,j) =[6] minimo(dp(k,i) + Costos(j, k) $\forall 0 \leq k \leq j-2$)
Como k≤j-2, k<j entonces dp(k,i) es mínimo(i,k)[7] por H.I.
Como los Costos(j, k) pueden variar mucho según el k no puedo asegurar que dp(k,i) + Costos(j, k) es mínimo(i,j). Pero dentro de todos esos valores me quedo con el mínimo entonces sé que vale.

1.5. Casos de prueba:

Dado que el algoritmo debe encontrar la división de trabajos que dá el mínimo, las instancias que permiten verificar su correctitud son las siguientes:

- Hay sólo un trabajo. Ej:

ti \ tj	0
1	3

  Solución: t1 en una máquina, la otra libre, costo: 3.
  El algoritmo dá la solución esperada.

El mínimo está dado por la siguiente distribución:

- Todos los trabajos en una máquina y ninguno en la otra.

[5] por definición de la función
[6] por definición de la función
[7] cómo las máquinas son iguales mínimo(i,k) = mínimo(k,i), lo escribo así para que se entienda mejor

Ej:

ti \ tj	0	1	2	3
1	1	-	-	-
2	3	2	-	-
3	6	3	2	-
4	3	2	1	3

Solución: t1, t2, t3, t4 en una máquina, la otra libre, costo:8.
El algoritmo dá la solución esperada.

- División (en orden) de los trabajos.

Ej:

ti \ tj	0	1	2	3
1	1	-	-	-
2	1	3	-	-
3	2	3	1	-
4	2	6	5	3

Solución: t1 en una máquina, t2, t3, t4 en la otra, costo: 6.
El algoritmo dá la solución esperada.

- División (sin orden) de los trabajos.

Ej:

ti \ tj	0	1	2	3
1	1	-	-	-
2	1	4	-	-
3	2	2	3	-
4	2	3	2	4

Solución: t1, t3 en una máquina t2, t4 en la otra, costo: 6.
El algoritmo dá la solución esperada.

Elegimos estos casos porque son distintas formas de distribuir los trabajos, lo que intentamos mostrar es que el algoritmo no deja distribuciones sín mirar.

2. Ejercicio 2.1

3. Ejercicio 2.2

4. Ejercicio 3