

Algoritmos y Estructuras de Datos III, TP2

Nicolás Chehebar, Matías Duran, Lucas Somacal

Índice

1. Problema 1	2
1.1. El Problema	2
1.1.1. Descripción	2
1.1.2. Ejemplos	2
1.2. El Algoritmo	2
1.2.1. La función de dinámica	2
1.2.2. El Pseudocódigo	3
1.3. Complejidad	4
1.4. Experimentación	5
1.4.1. Contexto	5
1.5. Conclusiones	5
2. Problema 2.1	5
3. Problema 2.2	5
4. Problema 3	5

1. Problema 1

1.1. El Problema

1.1.1. Descripción

Planteado de otra forma, el problema a resolver consiste en una situación en la que tenemos n trabajos t_1, t_2, \dots, t_n y dada cualquier división de los trabajos en dos secuencias $A = (t_{a_1}, t_{a_2}, \dots, t_{a_{|A|}})$ y $B = (t_{b_1}, t_{b_2}, \dots, t_{b_{|B|}})$ con $a_i < a_j \wedge b_i < b_j$ si $i < j$ (cada secuencia representa los trabajos que realiza una máquina) tiene asociado un costo; donde este viene dado por la suma del costo de A y el de B . El costo de A es $\sum_{i=1}^{|A|} costo(t_{a_i}, t_{a_{i-1}})$ donde $costo$ es una función que toma valores en \mathbb{N}_0 y $costo(t_i, t_j)$ está definido si $i > j$ con $i \in [1, 2, \dots, n] \wedge j \in [0, 1, \dots, n-1]$ y representa el costo de poner el trabajo i sobre el j (el costo de poner sobre el trabajo t_0 es el de ponerlo sobre la máquina vacía y $a_0 = 0$). El costo de B se calcula análogamente.

El problema pide dados los trabajos y la función de costo, dar A o B que minimice el costo y decir cuánto es este costo (basta dar uno de los dos ya que el otro se deduce por ser el complemento -en el conjunto de trabajos-).

1.1.2. Ejemplos

- $\begin{matrix} 4 \\ 2 \\ 300 \end{matrix}$

$\begin{matrix} 3 \\ 3 \\ 300 \end{matrix}$

tenemos 4 trabajos que sacando el primero son excesivamente caros de poner por primera vez en una máquina, luego si ponemos todos en la misma, el costo será $2 + 3 + 3 + 3 = 11$ y una máquina tendrá todos los trabajos (si todos no están en la misma, en algún momento pagamos 300 y el costo ya sería mayor a 11).

- $\begin{matrix} 4 \\ 2 \\ 300 \end{matrix}$

$\begin{matrix} 1 \\ 3 \\ 300 \end{matrix}$

tenemos 4 trabajos, ponemos el primero en una máquina y nos da costo 2, si bien en el próximo paso lo mejor es poner el nuevo trabajo encima (si hicieramos un algoritmo goloso), en ese caso el siguiente trabajo costará 300 haciendo el total > 299 , y si no hubieramos puesto el segundo encima, si bien costaba más en ese paso, reducía el costo del próximo, dando un costo total de 12 estando los trabajos 1, 3 y 4 en una máquina.

1.2. El Algoritmo

1.2.1. La función de dinámica

Para resolver el problema, utilizaremos programación dinámica. La idea de esto se basa en que la solución de nuestro problema es calculable en base a la solución de subproblemas (utilizamos optimalidad de subproblemas). Definimos así $f(q, h)$ como la función que asigna el mínimo costo posible para llegar al trabajo q -ésimo hecho (habiendo hecho de la 1 hasta la q inclusive) con el trabajo h como el último que se hizo en algunas de las máquinas. Tomamos como dominio de la f a los $q \in [1, 2, \dots, t] \wedge h \in [0, 1, \dots, q-1]$. donde $h = 0$ significa que hay una máquina vacía. Es clave notar que siempre que luego de que realizamos el trabajo q en una de las máquinas estará en el tope dicho trabajo, por ende basta definir que hay en la otra. Definimos a continuación la función para los valores en el dominio ya mencionado:

$$f(q, h) = \begin{cases} costo(q, 0) & \text{si } q = 1 \wedge h = 0 \\ f(q-1, h) + costo(q, q-1) & \text{si } h < q-1 \\ \min_{0 \leq h \leq q-2} f(q-1, h) + precios[q][h] & \text{caso contrario (h=q-1)} \end{cases} \quad (1)$$

Esta función hace efectivamente lo que queremos:

- En el primer caso lo hace pues si $q = 1$ esto implica $h = 0$ por restricciones de dominio y es la mínima cantidad dado que coloqué solo el primer trabajo, pues si o si el costo será el de colocar la primera sobre la máquina vacía, por ende será el mínimo.

- En el segundo caso también lo hace pues si esta un trabajo $h < q - 1$ en una máquina es porque el ultimo trabajo colocado (el q) se colocó en la otra, por ende previo a finalizar el trabajo q , estaba en una máquina el $q - 1$ y en otra el h . Más aún sabemos que el q lo colocamos sobre el $q - 1$. Supongamos $f(q, h)$ el costo mínimo dado el trabajo q hecho y el trabajo h en alguna impresora (análogo para $f(q - 1, h)$), si es $f(q, h) < f(q - 1, h) + \text{costo}(q, q - 1)$ luego es absurdo pues $f(q - 1, h)$ no es el mínimo, ya que hago la secuencia que da el mínimo en q trabajos hechos con h en una máquina sin el ultimo paso (resto su costo, o sea el de poner a q sobre $q - 1$) y me queda que tengo una forma de tener $q - 1$ trabajos hechos con h en una maquina con costo $f(q, h) - \text{costo}(q, q - 1) < f(q - 1, h)$ lo que es absurdo pues $f(q - 1, h)$ era el mínimo.
- En el tercer caso también sucede pues si está el trabajo $q - 1$ en una máquina con la impresión q ya hecha, es porque la impresión q se colocó sobre alguna impresión h con $0 \leq h \leq q - 2$. Dada dicho trabajo, de forma totalmente análoga al caso de arriba, debe ser el mínimo buscado con q trabajos hechos y el $q - 1$ en una máquina $f(q, q - 1) = f(q - 1, h) + \text{costo}(q, h)$. Luego como no se cual trabajo de todos pudo haber sido, me quedo con el mínimo moviendo los h en el rango dado.

Así, definimos una función que resuelve el problema pedido si hayamos el $\min_{0 \leq h \leq t-1} f(t, h)$ pues es el mínimo costo de realizar hasta el trabajo t (o sea todos) con el trabajo h en alguna máquina (me fijo todos los escenarios posibles como puede terminar la otra máquina, o sea todos los posibles h).

Así, podemos implementar la f dada, donde podemos ir recordando los valores que toma f y evitar calcularlos varias veces. Más aún podríamos mantener una lista (ordenada) de cuales son los elementos que hay en alguna máquina y cada vez que agregamos un trabajo, chequeamos si lo agregamos sobre el ultimo de la lista y en ese caso lo incluimos al final de esta (sino es porque fue a la otra máquina).

Lo que sucede es que tenemos varios subproblemas y en este caso siempre resolvemos todos, por lo que no parece tener una clara ventaja hacerlo top-down. Más aún, hacerlo bottom-up nos permitirá solo guardarnos los subproblemas relativos a tener hecho exactamente hasta el anterior trabajo (con $q - 1$ y para todos los h , los menores no los utilizo en el calculo de $f(q, h)$), o sea nos reduce la complejidad espacial! Ya que inicialmente debíamos guardar el valor de f para todo $q \in [1, 2, \dots, t] \wedge h \in [0, 1, \dots, q - 1]$ lo que era $\mathcal{O}(n^2)$, y de esta forma solamente guardamos los valores para $q - 1$ lo que es $\mathcal{O}(n)$.

Veamos todo esto en un pseudocódigo:

1.2.2. El Pseudocódigo

Cabe aclarar que en el pseudocódigo (como también en la implementación) numeramos los trabajos desde 0 a excei3n de en *costos* (que es una matriz) en el segundo indice, los trabajos estan numerados desde 1 (ya que el 0

se reserva para el costo de poner sobre la máquina vacía).

Algorithm 1: Devuelve el mínimo costo de entre todas las formas posibles de realizar todas las impresiones

```

1 Backtracking (confiables, actual, matriz, maximo);
   Input : trabajos  $\in \mathbb{N}_0$ ; costo  $\in \mathbb{N}_0^{\text{trabajos} \times \text{trabajos}}$ 
   Output: costo  $\in \mathbb{N}_0$ , lista vector enteros
2 Inicializo en 0 actualcosto y anteriorcosto vectores de enteros (de tamaño trabajos);
3 Inicializo en vectores vacíos actuallista y anteriorlista vectores de vectores de enteros (de tamaño
   trabajos);
4 for  $q \in [0, 1, \dots, \text{trabajos})$  do
5   for  $h \in [0, 1, \dots, q)$  do
6     actualcosto[ $h$ ] = anteriorcosto[ $h$ ] + costo[ $q$ ][ $h$ ];
7     actuallista[ $h$ ] = anteriorlista[ $h$ ];
8     if Estaba  $q - 1$  en anteriorlista[ $h$ ] then
9       | Agrego  $q$  a anteriorlista[ $h$ ];
10    end
11    actualcosto[ $q$ ] =  $\min(\text{actualcosto}[q], \text{anteriorcosto}[h] + \text{costo}[q][h])$ ;
12    Recuerdo en elegido el  $h$  que consiguió el minimo;
13  end
14  actuallista[ $q$ ] = anteriorlista[elegido];
15  if NO Estaba  $q - 1$  en anteriorlista[elegido] then
16    | Agrego  $q$  a anteriorlista[elegido];
17  end
18  anteriorcosto = actualcosto;
19  anteriorlista = actuallista;
20 end
21 costo = minimo de actualcosto (se alcanza en actualcosto[posicion]);
22 lista = actuallista[posicion];
23 return costo, lista;

```

En el pseudocódigo básicamente lo que hacemos es aplicar la f pero en orden, es importante esto ya que hay que tener cuidado en el orden en que resolvemos las dependencias (es porque estamos haciendo bottom-up). Es claro que cada fila, usa la fila anterior, o sea para calcular $f(q, h)$ para todo h uso todos los valores de $f(q - 1, h)$ para todo h . Por esto es que ambos for se anidan de dicha manera. Al principio del for actualizamos el costo segun nos dice la f y la lista de los trabajos que hay en alguna máquina se modifica solo si era el de la máquina que tenía a $q - 1$ ya que es a la que le agrego el trabajo q . Además, como voy a recorrer todos los h , voy actualizando el *actualcosto*[q] si tengo un menor *anteriorcosto*[h] + *costo*[q][h]; una vez que iteré en todos los h calculé el minimo que es $f(q, q - 1)$. Ahí salimos del primer for y como hacía con *actuallista*[h] actualizo si corresponde la *actuallista*[q]. Finalmente, antes de pasar a la siguiente iteración pongo en *anteriorcosto* el *actualcosto* y en *anteriorlista* la *actuallista*, ya que en la proxima iteración los actuales serán anteriores y sobre actual pisaré y guardaré nuevos resultados. Finalmente, se devuelve el mínimo buscado y su lista asociada (lo que nos pedían era $\min_{0 \leq h \leq t-1} f(t, h)$ que en nuestro caso es el minimo de *actualcosto*).

1.3. Complejidad

Cabe aclarar que para el análisis de complejidad tomaremos n como la cantidad de trabajos. Como pudimos ver en la explicación de la función de dinámica, tenemos n^2 subproblemas y cada uno se resuelve en $\mathcal{O}(1)$ salvo los subproblemas donde $h = q - 1$ que toman $\mathcal{O}(n)$. Luego tengo $\mathcal{O}(n^2)$ subproblemas (son n^2 en total y le saco los que no se resuelven en $\mathcal{O}(1)$ que son n) resueltos en $\mathcal{O}(1)$ cada uno y $\mathcal{O}(n)$ subproblemas (hay uno por cada q) resueltos en $\mathcal{O}(n)$ cada uno. Luego se deduce que la complejidad sera $\mathcal{O}(n * n) + \mathcal{O}(n^2 * 1) = \mathcal{O}(n^2)$

Más aún esto se ratifica si miramos el pseudocódigo ya que realizamos todas operaciones que son $\mathcal{O}(n)$ u $\mathcal{O}(n)$ fuera del ciclo (inicialización o recorrido de vectores de tamaño a lo sumo n). Veamos que sucede dentro del ciclo: tenemos dos ciclos anidados que se ejecuta cada uno a lo sumo n veces, por ende todo se ejecuta a lo sumo n^2 veces y todo lo de adentro son operaciones $\mathcal{O}(1)$ (checkear si $q - 1$ esta en *anteriorlista*[h] es $\mathcal{O}(1)$ porque inserto siempre ordenado y si es un elemento, es el ultimo; lo mismo vale para checkear si $q - 1$ esta en *anteriorlista*[*elegido*]). Luego salimos del segundo for (el anidado) y cabe aclarar que copiar el vector *actualcosto* y *actuallista* no es $\mathcal{O}(1)$ sino $\mathcal{O}(n)$, pero esta solo en uno de los ciclos, por lo que se repite n veces y aporta una complejidad de $n * \mathcal{O}(n) = \mathcal{O}(n^2)$. Por ende en el ciclo tenemos $n^2 * \mathcal{O}(1) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$ y sumado a lo que

esta fuera del ciclo nos da $\mathcal{O}(n) + \mathcal{O}(n^2) = \mathcal{O}(n^2)$

1.4. Experimentación

1.4.1. Contexto

La experimentación se realizó toda en la misma computadora, cuyo procesador era Intel(R) Atom(TM) CPU N2600 @ 1.60GHz, de 36 bits physical, 48 bits virtual, con una memoria RAM de 2048 MB. Para experimentar, se calculó el tiempo que tardaba el algoritmo sin considerar el tiempo de lectura y escritura ni el tiempo que llevaba armar la matriz (ya que se leía un dato, se escribía la matriz y luego se leía el siguiente). El tiempo se medía no como tiempo global sino como tiempo de proceso, calculando la cantidad de ticks del reloj (con el tipo `clock_t` de C++) y luego se dividía el delta de ticks sobre `CLOCKS_PER_SEC`. En todos los experimentos el tiempo se mide en segundos.

1.5. Conclusiones

2. Problema 2.1

3. Problema 2.2

4. Problema 3