Programación 3
Año 2023

Práctico 8

Ejercicio 1 (Kleinberg & Tardos, Ex. 6.2)**.** Estamos gestionando el trabajo de un equipo de hackers y necesitamos planificar las tareas para las próximas semanas. Tenemos dos tipos de trabajos: los poco estresantes y los muy estresantes. Para cada semana tenemos que decidir el tipo de trabajo que realizará el equipo. Si realizamos un trabajo poco estresante en la semana i obtenemos una ganancia de l_i pesos, mientras que si realizamos un trabajo muy estresante obtenemos una ganancia de h_i pesos, donde l_i y h_i son mayores que cero. Si el equipo va a realizar un trabajo muy estresante en la semana i (lo cual da mayor ganancia), deberá no hacer nada durante la semana $i-1$ para prepararse.

Un plan de trabajo para n semanas se especifica diciendo si el equipo realizará un trabajo de mucho estrés, poco estrés o nada durante cada semana. Dados los valores $l_1, l_2, \ldots, l_n$ y $h_1, h_2, \ldots, h_n$, nuestro objetivo es hallar un plan óptimo en el sentido de maximizar la ganancia.

```
1  Hacer i = 1
2  while i < n do
3      if h_{i+1} > l_i + l_{i+1} then
4          BadSol[i] = nada
5          BadSol[i + 1] = muy estresante
6          i = i + 2
7      else
8          BadSol[i] = poco estresante
9          i = i + 1
10     end
11 end
12 if i = n then BadSol[i] = poco estresante
13 Devolver BadSol
```

Figura 1.1: Algoritmo incorrecto para encontrar un plan de trabajo óptimo.

(a) Dé un ejemplo de instancia para la cual el algoritmo de la figura 1.1 produce un plan de trabajo que no es óptimo.

(b) Dada una instancia del problema, sea $OPT(i)$ la ganancia máxima que puede obtenerse en las primeras i semanas, $0 \leq i \leq n$. ¿Cuánto valen $OPT(0)$ y $OPT(1)$?

(c) Si en la semana n, $n > 0$, decidimos realizar un trabajo poco estresante, ¿cuál es la mayor ganancia que podemos obtener a lo largo de n semanas en función de $OPT(n-1)$?

(d) Si en la semana n, $n > 1$, decidimos realizar un trabajo muy estresante, ¿cuál es la mayor ganancia que podemos obtener a lo largo de n semanas en función de $OPT(n-2)$?

(e) Muestre que OPT satisface una relación de recurrencia y especifique cuál es esa relación.

(f) Dé un algoritmo eficiente para calcular $OPT(n)$.

(g) Dé un algoritmo eficiente para hallar un plan de trabajo óptimo.

Ejercicio 2 (Kleinberg & Tardos, Ex. 6.6)**.** Un algoritmo de "pretty printing" tiene el objetivo de organizar las palabras de un texto en líneas consecutivas, de forma que al imprimir estas líneas el espacio en blanco que queda sobre el margen derecho se distribuya armoniosamente. Por ejemplo, el siguiente texto

```
A cada uno
de los muros de cada hexágono corresponden
cinco anaqueles;
cada anaquel
encierra treinta y dos
libros de formato uniforme; cada libro es de
cuatrocientas diez páginas;
cada página,
de cuarenta renglones;
cada renglón,
de unas ochenta letras
de color negro.
```

resulta mucho más agradable cuando lo organizamos de modo que el margen derecho sea más "parejo", como se muestra continuación

```
A cada uno de los muros de cada hexágono
corresponden cinco anaqueles; cada
anaquel encierra treinta y dos libros
de formato uniforme; cada libro es de
cuatrocientas diez páginas; cada página,
de cuarenta renglones; cada renglón, de
unas ochenta letras de color negro.
```

Supongamos que el texto de entrada consiste en una secuencia de palabras $W = w_1, w_2, \cdots w_n$, donde w_i tiene c_i caracteres de largo. El texto se imprime usando una fuente monoespaciada (todos los caracteres ocupan el mismo espacio) y el ancho de página permite imprimir a lo sumo L caracteres por línea. Ignoramos aspectos de puntuación (por ejemplo podemos suponer que un símbolos de puntuación que sigue a una palabra w_i está contado en el largo c_i) y no contemplamos la posibilidad de separar una palabra en dos líneas consecutivas. Todas las palabras, salvo la última de cada línea, están seguidas de un espacio en blanco. Por lo tanto, si $w_j, w_{j+1}, \ldots, w_k$ se imprimen en una misma línea, ℓ, la cantidad de caracteres ocupados en la línea ℓ es

$$L_\ell = \left(\sum_{i=j}^{k-1} (c_i + 1) \right) + c_k \tag{2.1}$$

y la cantidad de caracteres que "sobran" en la línea ℓ con respecto al límite de L caracteres por línea es

$$d_\ell = L - L_\ell \,. \tag{2.2}$$

Decimos que una partición P de un texto en líneas es *válida* si $d_\ell \geq 0$ para toda línea ℓ de P. El problema del "pretty printing" puede formalizarse como una minimización entre todas las particiones válidas de alguna función de costo definida sobre los valores d_ℓ.

Dé un algoritmo eficiente para encontrar una partición válida de un texto que minimiza la función de costo

$$C(P) = \sum_{\ell \in P} d_\ell^2 \,. \tag{2.3}$$

Sugerencia: Para todo par de índices i, j, $1 \leq i \leq j \leq n$, sea $e_{i,j} = d_\ell^2$ si $d_\ell \geq 0$, donde ℓ es la línea de texto formada por las palabras $w_i, w_{i+1}, \ldots, w_j$, y $e_{i,j} = +\infty$ si $d_\ell < 0$. Calcule los valores $e_{i,j}$ y aplique una estrategia similar a la estudiada en la sección 6.3 del libro de referencia del curso.

Ejercicio 3 (Kleinberg & Tardos, Ex. 6.9)**.** Estamos ayudando a dirigir un sistema de cómputo de alto desempeño, capaz de procesar varios terabytes de datos por día, y queremos planificar el trabajo para los próximos n días. En cada día i, $1 \leq i \leq n$, nuestros clientes proporcionan x_i terabytes de datos para procesar. Por cada terabyte procesado recibimos una ganancia fija. Sin embargo, como nuestro sistema tiene una capacidad de cómputo limitada, no siempre es posible procesar todos los datos que se nos presentan; si en un determinado día nos quedan datos sin procesar, esos datos caducan y se inhabilitan al finalizar el día (no es posible trabajar sobre estos datos en el futuro porque los resultados dejan de ser vigentes).

El procesamiento de los datos se realiza con un programa que no es del todo confiable. En particular, debido a un incremento sostenido del consumo de memoria del programa a medida que transcurre el tiempo, la cantidad de datos que se pueden procesar disminuye con cada día que pasa desde la última vez que el sistema fue reiniciado. En el primer día luego de que el sistema ha sido reiniciado se pueden procesar s_1 terabytes, en el segundo día s_2 terabytes y así hasta el n-ésimo día, donde se cumple $s_1 > s_2 > ... > s_n > 0$. Si reiniciamos el sistema en un determinado día este recobra su capacidad máxima, pero durante ese día no es posible procesar datos.

El problema consiste en, dadas las cantidades de datos disponibles por día, $x_1, x_2, ..., x_n$, y el perfil de rendimiento del sistema, $s_1, s_2, ..., s_n$, establecer los días en los que se reiniciará el sistema de modo de maximizar la cantidad total de datos procesados. Notar que en el día i solo hay disponibles para procesar hasta x_i terabytes de datos, que podría ser menor que la capacidad del sistema en ese día. Asumimos que el sistema comienza el primer día en su máxima capacidad, lo cual equivale a considerar que siempre se reinicia el sistema en un día cero que antecede al período que estamos planificando.

(a) Dé un ejemplo de una instancia con las siguientes propiedades:

- Se cumple que $x_i > s_i$ para todo día i, $1 \leq i \leq n$.
- La solución óptima reinicia el sistema al menos 2 veces.

Además del ejemplo, especifique una solución óptima (no es necesario demostrar que es óptima).

(b) Dada una instancia del problema, para cada par de índices i, j, $1 \leq j \leq i \leq n$, definimos $OPT(i, j)$ como la cantidad máxima de datos que es posible procesar en el lapso comprendido entre el día i inclusive y el día

n inclusive, habiendo reiniciado el sistema por última vez j días antes que i, es decir en el día $i - j$ (recordar que el sistema se reinicia en el día cero, que corresponde a $i = j$). Indique cuánto vale $OPT(n, j)$ para cada j, $1 \leq j \leq n$.

(c) Para $i < n$, ¿cuánto vale $OPT(i, j)$ si decidimos reiniciar el sistema en el día i?

(d) Para $i < n$, ¿cuánto vale $OPT(i, j)$ si decidimos no reiniciar el sistema en el día i?

(e) Muestre que OPT satisface una relación de recurrencia y especifique cuál es esa relación.

(f) Dé un algoritmo eficiente para calcular la cantidad total de terabytes procesados en una solución óptima.

(g) Dé un algoritmo eficiente para determinar un conjunto de días en los cuales reiniciar el sistema para maximizar la cantidad total de terabytes procesados.