Universidad de Chile
Departamento de Ciencias de la Computación
CC4102 - Diseño y Análisis de Algoritmos

Auxiliar 6 - "Algoritmos Online"

Profesor: Gonzalo Navarro
Auxiliar: ~~Manuel~~ Ariel Cáceres Reyes

16 de Octubre del 2017

P1. Tom y Jerry

Jerry lanza platos desde lo alto de un armario y Tom intenta recogerlos antes de que se estrellen en el suelo del pasillo. El pasillo tiene $2k+1$ baldosas y Tom recorre una baldosa por segundo. Un plato tarda k segundos en llegar al suelo desde que Jerry lo lanza. Tom sabe en qué baldosa caerá el plato en el instante en que Jerry lo lanza, de manera que puede recorrer hasta k baldosas para salvarlo. Jerry lanza un plato por segundo, y lo puede lanzar hacia la baldosa que quiera. Nos interesa diseñar una estrategia competitiva para Tom (en términos de la cantidad de platos salvados). Tom comienza parado en la baldosa central, y Jerry lanzará n platos.

a) Muestre que la estrategia de ir a buscar el siguiente plato lanzado en caso de que sea posible alcanzarlo (e ignorar todo hasta recogerlo), y sino seguir en el mismo lugar esperando el próximo plato, no es c-competitiva para ninguna constante c.

b) Muestre que la estrategia de ir a buscar siempre el plato más cercano al suelo, tampoco es c-competitiva.

c) Diseñe una estrategia $2k$-competitiva para Tom. Demuestre su competitividad y encuentre un caso donde se salve sólo uno de cada $2k$ platos que podría salvar en el algoritmo óptimo (que leyera la mente de Jerry).

P2. k servidores, online

Considere el escenario donde tiene k puntos (*servidores*) en un espacio *métrico* y una secuencia de puntos (*peticiones*) que debe atender. Cada vez que llega una petición, un servidor debe moverse hacia esa posición para atenderla.
El problema *online* consiste en minimizar la distancia recorrida por todos los servidores luego de n peticiones, sin saber la secuencia de puntos a atender.

a) Muestre que para cualquier algoritmo su radio competitivo es al menos k.

b) Consideremos ahora el problema de 2 servidores y sus peticiones en una línea. Muestre que el algoritmo de enviar al servidor más cercano no es c-competitivo para ninguna constante c.

"I didn't come here to be the average."
Michael Jordan

Soluciones

P1. Para desarrollar este problema nombraremos las baldosas $-k, -k+1, \ldots, -1, 0, 1, \ldots, k-1, k$ de un extremo a otro, de modo que Tom empieza en la baldosa 0. Además, dado que nos enfrentamos a un problema de maximización diremos que un algoritmo online es $\rho(n)$-competitivo si se cumple que existe una constante k para la cual se cumpla que en todo input de tamaño n

$$\rho(n) \cdot ALG \geq OPT + k$$

a) Si Jerry 😈, lanza el primer plato en $-k$, y los $n-1$ restantes en k, Tom irá a buscar el primer plato a $-k$, pero luego no irá por el resto de los platos, pues

- O bien, está esperando el plato de $-k$.
- O bien, ya atrapo el plato de $-k$, pero no alcanza los que están en k.

Por lo tanto, mientras este algoritmo solo salva 1 plato, el óptimo salva los $n-1$ de k. De modo que $OPT = n - 1 \cdot ALG$, y entonces, $> c \cdot ALG$ para cualquier constante c eligiendo n suficientemente grande.

b) Jerry podría lanzar el primer plato en -1 y los siguientes en $1, 2, \ldots, k-1, k, k-1, \ldots$. En este caso, Tom espera el plato de -1 y luego de obtenerlo va a buscar los platos en el mismo orden en que Jerry los lanzó, pero no alcanza ninguno de ellos, en cambio el óptimo hubiese dejado el plato en -1 y recogido los demás de la secuencia de lanzamientos. Se cumple entonces que $OPT = n - 1 \cdot ALG$, por lo que se procede del mismo modo que en la parte anterior para mostrar que este algoritmo no es c-competitivo.

c) La estrategia es la siguiente; si Tom está al medio, su objetivo será el plato que Jerry está lanzando en ese momento, lo va a buscar y luego vuelve al centro repitiendo así esta estrategia.
Notemos primero que Tom siempre recoge su objetivo (pues este está a distancia a lo más k). Además, en ir y volver, Tom se demora a lo más $2k$ segundos, y por lo tanto el óptimo offline a lo más pudo salvar $2k$ platos en ese transcurso. Con esto se cumple que en cada *ida y vuelta* $2k \cdot ALG \geq 2k \geq OPT$ y como esto se cumple para todas las "ida y vuelta"s realizadas, este algoritmo es $2k$-competitivo.

P2. a) Sea k y A un algoritmo que resuelve el problema de los servidores. Considere el siguiente input: un espacio métrico con $k+1$ puntos (uno más que el número de servidores) y los servidores parten ubicados en los primeros k. Las peticiones $\sigma = \sigma_1 \ldots \sigma_r$ serán aquellos puntos que A no tiene cubierto en ese momento 😈(por palomar siempre existe uno de estos puntos) empezando por $k+1$.

Con esto construiremos k algoritmos que lo hacen "mejor" que A, $B_1, \ldots, B_k$. De modo que :

"I didn't come here to be the average."
Michael Jordan

- Antes de la primera petición B_i manda al servidor que está en i a $k+1$.
- Queremos mantener el invariante que en cada petición todos los B_i tienen cubierto a σ_i.
- Para lograr lo anterior, cuando se pida σ_{i-1} y A lo cubra con un servidor que se encontraba en σ_i (esto es así pues las peticiones fueron construidas de esta forma), existirá un único[1] B_j que no tiene cubierto σ_i y este (luego de responder la petición $i-1$) lo cubrirá con el servidor que tiene en σ_{i-1}.
- Con esto tenemos que la suma de los costos de los B_i corresponden a los movimientos que se hacen antes de la primera petición, más los movimientos que hace exactamente uno de los B_i luego de cada petición. Sin embargo, este último costo es el mismo que realiza A (cuando A mueve uno de sus servidores de σ_i a σ_{i-1} alguno de los B_i ya lo hizo desde σ_{i-1} a σ_i). Es decir,

$$\begin{aligned}\sum_{i=1}^{k} C_{B_i}(\sigma) &= C_A + \sum_{i=1}^{k} dist(i, k+1) \\ &\leq C_A + k \max_i(dist(i, k+1))\end{aligned}$$

  Finalmente debe existir uno de estos algoritmos que cumpla:

$$OPT \leq C_{B_j}(\sigma) \leq \frac{1}{k} C_A + \max_i dist(i, k+1)$$

  ,por lo que A es al menos k- competitivo.

b) Consideremos el input con los puntos de la recta numérica, los servidores inicialmente ubicados en 2 y 4 y la secuencia de peticiones $1, 2, 1, 2, 1, 2, \ldots, 1, 2$ (repitiendo $1, 2$ n veces). El algoritmo estará moviendo el servidor de la izquierda entre 1 y 2 teniendo un costo total de $2n$. Sin embargo, el óptimo consiste en responder la primera petición con el servidor de la derecha y luego no mover más los servidores, a costo total 3. Finalmente, como n es un valor arbitrario, el algoritmo puede ser tan malo (respecto al óptimo) como queramos, por lo que no puede ser c-competitivo para ninguna constante c.

[1] Notar que este es otro invariante que mantienen los algoritmos