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

Auxiliar 9 - "Algoritmos Online y Probabilísticos"

Profesores: Pablo Barceló
Gonzalo Navarro
Auxiliares: Matilde Rivas
Bernardo Subercaseaux

3 de Diciembre del 2018

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. La feria Online

Tiene usted k\$ y quiere gastarse la mayor cantidad posible en una feria a lo largo de una calle de un solo sentido (es decir, usted solo puede hacer una pasada por ella), de modo que cuando compra algo ya no puede devolverlo (además hay un solo ejemplar de cada producto).
Los productos tienen distintos precios entre 1 y k. Pero por ejemplo si se gasta 1\$ en el primer producto, y todos los demás cuestan k\$, no podrá comprar nada más.

a) Muestre que no se puede ser mejor que k-competitivo para este problema.

b) Considere ahora una variante en la que usted puede devolver uno o más productos ya comprados y recuperar el dinero (pero no puede comprar algo por lo que ya pasó). Muestre un algoritmo 2-competitivo para este caso.

P3. Polinomios

Dados 3 polinomios en una variable p, q de grado n y r de grado $2n$. Muestre un algoritmo probabilístico que tome tiempo $\mathcal{O}(n)$ en determinar si $pq = r$.

P4. Las Vegas vs Monte Carlo

a) Muestre que si se tiene un algoritmo de tipo Las Vegas de tiempo esperado T y una constante $c > 0$. Se puede construir, para el problema, un algoritmo Monte Carlo de tiempo cT y probabilidad de error a lo más $1/c$.

b) ¿Siempre se puede transformar un algoritmo Monte Carlo en un algoritmo de tipo Las Vegas? ¿Que restricción adicional sobre el problema podemos colocar para que sea cierto?

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 ALG un algoritmo que resuelve este problema. Como adversario creamos el siguiente input que depende del comportamiento de ALG.
Ofrecemos productos a precio 1\$ a lo más k veces, si ALG compra alguno de estos productos, el siguiente que le ofrecemos es de precio k\$ (que ya no puede comprar) y dejamos de ofrecer productos. En cualquiera de los dos casos (si ALG compra o no alguno de los productos) lo hará al menos k veces peor que el óptimo (comprar uno de 1\$ o nada cuando pudo haber comprado el de k\$ o los k de 1\$ respectivamente).
Por lo tanto, cualquier algoritmo tiene un input donde lo hace al menos k veces peor que el óptimo, por lo que no se puede ser mejor que k-competitivo para el problema.

b) Nuestro algoritmo hará lo siguiente:

- Si le ofrecen un producto de precio $\lceil k/2 \rceil$, vende todo lo que ha comprado y compra este producto.
- Sino, solo compra el producto.
- Si en algún momento lo comprado alcanza un monto $\geq \lceil k/2 \rceil$ deja de comprar.

De este modo me aseguro de gastar siempre $\geq \$\frac{k}{2}$

P3. Elegimos uniformemente $x \in \{0, \ldots, 4n+1\}$ y luego respondemos `true` si $p(x) \cdot q(x) = r(x)$ y `false` en otro caso.

- En el caso de que la igualdad sea correcta, el algoritmo responde `true` de manera correcta.
- Sin embargo, cuando la igualdad es falsa ($pq \neq r$) el algoritmo podría responder erróneamente `true` en caso que $p(x) \cdot q(x) = r(x)$ o equivalentemente cuando el polinomio $g = pq - r$ se hace cero, i.e. cuando x es cero de g.
  Veamos que g es de grado a lo más $2n+1$ por lo que tiene a lo más $2n+1$ raíces, y luego la probabilidad que el algoritmo se equivoque es:

$$\mathbb{P}(x \; es \; raiz \; de \; g) \leq \frac{2n+1}{4n+2} = \frac{1}{2}$$

P4. Preliminares :

- Desigualdad de Markov: Si X es una variable aleatoria no negativa con valor esperado positivo entonces

$$\forall c > 0, \; \mathbb{P}(X \geq c\mathbb{E}(X)) \leq \frac{1}{c}$$

- Algoritmo Montecarlo: Algoritmo probabilístico que termina en un tiempo determinado, pero tiene una probabilidad de error distinta de 0.
- Algoritmo Las Vegas: Algoritmo probabilístico con probabilidad de error 0, pero que no asegura el término de su ejecución, solo asegura un valor esperado finito del tiempo que demora.

a) Si tenemos un algoritmo A "Las Vegas" que resuelve el problema en tiempo esperado $T > 0$, creamos el siguiente algoritmo "Montecarlo":

- Correr A por cT tiempo, si ha terminado, respondemos lo mismo que A, si no respondemos "`cualquier cosa`".

Definamos la variable aleatoria X como el tiempo que demora el algoritmo A. Por Desigualdad de Markov sabemos que $\mathbb{P}(X \geq cT) \leq \frac{1}{c}$, pero $\mathbb{P}(X \geq cT)$ es la probabilidad de que A se demore más de cT y por lo tanto una cota superior a la probabilidad de que nuestro algoritmo "Montecarlo" se equivoque.

b) La condición para hacer la transformación es contar con un procedimiento eficiente que verifique el `output` dado por el algoritmo "Montecarlo" sea correcto, pues si lo tenemos podemos hacer un algoritmo "Las Vegas" que corra el "Montecarlo" hasta que este último entregue una respuesta correcta.

Agradecimientos a ~~Manuel~~ Ariel Cáceres :D