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

Auxiliar 7 - "Algoritmos Online, Dominios Discretos y Análisis Amortizado"

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

16 de Octubre del 2017

P1. k servidores, online (continuación)

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.
Consideremos ahora el problema de 2 servidores y sus peticiones en una línea y el siguiente algoritmo:

- Si ambos servidores están al mismo lado de la petición, se envía el servidor más cercano.
- Si la petición está entre los dos servidores, se envían los dos a la misma velocidad constante, deteniéndose cuando uno de ellos llega al objetivo.

Muestre que este algoritmo es 2-competitivo.

P2. Números en rango

Describa un algoritmo que, dados n enteros en $[0, \ldots, k]$, preprocese su entrada y responda cuántos de estos enteros se encuentran en el rango $[a, \ldots, b]$ en tiempo constante. Su algoritmo debería tomar tiempo $\mathcal{O}(n + k)$ en el preprocesamiento.

P3. Árbol cartesiano

Dado un arreglo de enteros distintos $A[1, n]$, un árbol cartesiano es un árbol binario de n nodos. Si el mínimo de A está en $A[i]$, entonces la raíz del árbol cartesiano corresponde a $A[i]$, el hijo izquierdo al árbol cartesiano de $A[1, i-1]$ y el hijo derecho al árbol cartesiano de $A[i+1, n]$. Cuando el rango de A se hace vacío el árbol cartesiano es vacío.

a) Dibuje el árbol cartesiano de $A[1, 10] = [2, 9, 4, 6, 7, 5, 3, 1, 8, 10]$.

b) Muestre que el árbol cartesiano se puede construir en tiempo $\mathcal{O}(n)$ a partir de $A[1, n]$. Para esto, considere usar inducción, procesando los elementos de A de izquierda a derecha y análisis amortizado para obtener la cota.

"Those who can imagine anything, can create the impossible."
Alan Turing

Soluciones

P1. Consideremos un conjunto de consultas en la línea. Para realizar este análisis compararemos el costo amortizado del ALG con el costo real de cada operación del OPT llegando así a la competitividad.
Para esto definamos lo siguiente:

- $(s_1)_i, (s_2)_i$ como las posiciones de los servidores de ALG después de haber respondido a la i-ésima consulta.
- $(s_1^*)_i, (s_2^*)_i$ como las posiciones de los servidores de OPT después de haber respondido a la i-ésima consulta.
- c_i es costo incurrido por ALG para responder la i-ésima consulta.
- c_i^* es costo incurrido por OPT para responder la i-ésima consulta.
- $(m_1)_i$, valor de uno de los emparejamientos entre los servidores de OPT y ALG luego de la i-ésima consulta. Corresponde a $d((s_1)_i, (s_1^*)_i) + d((s_2)_i, (s_2^*)_i)$.
- $(m_2)_i$, valor del otro emparejamiento entre los servidores de OPT y ALG luego de la i-ésima consulta. Corresponde a $d((s_1)_i, (s_2^*)_i) + d((s_2)_i, (s_1^*)_i)$.
- $m_i = \min((m_1)_i, (m_2)_i)$, valor del emparejamiento mínimo entre los servidores de OPT y ALG luego de la i-ésima consulta.
- $\phi_i = 2m_i + d((s_1)_i, (s_2)_i) = 2m_i + d_i$, es decir, el doble de emparejamiento mínimo más la distancia de los servidores de ALG.

Como los servidores comienzan en la mismas posiciones $\phi_0 = 0$, y dado que el potencial es una suma de distancias $\phi_i \geq 0$, por lo que es un potencial válido.

Analicemos ahora el cambio de potencial de una consulta cualquiera.
Cuando OPT mueve sus servidores c_i^*, d_i no cambia, y m_i puede cambiar a lo más c_i^*, pues puedo encontrar el mismo emparejamiento mínimo que tenía antes (el cual puede ser o no mínimo ahora), el que vale a lo sumo c_i^* unidades más. Con esto $\Delta\phi_i \leq 2c_i^*$.
Cuando ALG mueve sus servidores c_i:

- Si la consulta no cayó entre los servidores, entonces solo uno de ellos se mueve c_i alejándose del otro. Por lo tanto d_i aumenta en c_i.
  Sin perdida de mucha generalidad, podemos asumir que el servidor de la derecha va a atender la consulta, en donde se encuentra uno de los servidores de OPT, moviéndose c_i hacia este. Dicho esto el otro servidor de ALG se encuentra a la izquierda del que responde la consulta y lo único que nos queda analizar son las posibles posiciones del otro servidor de OPT.
  - A la derecha del servidor de OPT que respondió la consulta.

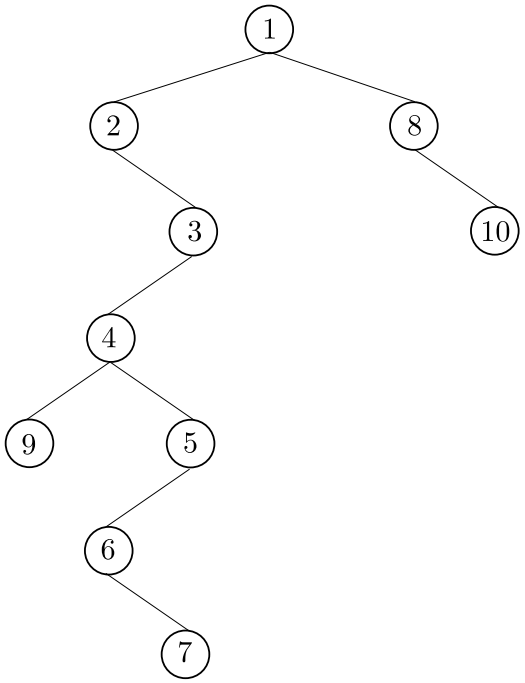
  - Entre el servidor que respondió la consulta en OPT y el que lo hará en ALG.
  - Entre los servidores de ALG.
  - A la izquierda del servidor izquierdo de ALG.

  En los 4 casos se puede apreciar que m_i disminuye c_i. Y por lo tanto ALG produce $\Delta\phi \leq -c_i$.
- Si la consulta cayó entre los dos servidores de ALG, entonces d_i disminuye en c_i. Por otro lado, como ambos se acercan $c_i/2$ a un servidor de OPT, uno de ellos se acerca esa distancia en el emparejamiento mínimo, mientras que el otro a lo más se aleja esa distancia por lo que el valor de m_i no aumenta (aunque puede que se reduzca). Y también $\Delta\phi \leq -c_i$.

Por lo tanto, $\Delta\phi_i \leq 2c_i^* - c_i$ y $\hat{c}_i \leq 2c_i^*$. Y de este modo, $C_{ALG} = \sum c_i \leq \sum \hat{c}_i \leq \sum 2c_i^* = 2C_{OPT}$.

P2. Si hacemos *CountingSort* sobre los elementos en tiempo $\mathcal{O}(n+k)$ podemos obtener el arreglo C que en $C[i]$ indica la posición del último i del conjunto de n enteros y por lo tanto, la cantidad de enteros $\leq i$, además si hay algún $i \in \{1, \dots k\}$ que no está en el conjunto de enteros entonces $C[i] = C[pred(i)]$ donde $pred(i)$ es el mayor entero menor a i que si está en el conjunto. Con esto, el número de enteros en el rango $[a, \dots, b]$ será $C[b] - C[a-1]$ 😁.

P3. a) Este es el árbol cartesiano correspondiente:



"Those who can imagine anything, can create the impossible."
Alan Turing

b) Construiremos un algoritmo que vaya generando el árbol cartesiano de un prefijo del arreglo que se nos da como input, es decir, si el arreglo de input es $A[1, n]$ entonces haremos una iteración de n pasos y al final del paso i tendremos el árbol cartesiano correspondiente a $A[1, i]$.

Además mantendremos el invariante de tener un puntero p al hijo más derecho del árbol, esto, pues en el árbol cartesiano de $A[1, i]$ el hijo más derecho es precisamente $A[i]$ (pues es el último elemento).

La construcción entonces es como sigue, empezamos con el árbol de un solo nodo (que seteamos como p) $A[1]$. Luego, asumiendo que tenemos el árbol cartesiano para $A[1, i]$ y p apuntando al nodo $A[i]$, si $A[i + 1] > A[i]$ entonces colgamos a $A[i + 1]$ como hijo derecho de $A[i]$ y seteamos p al nodo $A[i + 1]$. En caso que $A[i + 1] < A[i]$ subimos por el árbol desde p hasta encontrar un nodo f que contenga un $A[j] < A[i + 1]$, si s es su hijo derecho entonces ponemos a $A[i+1]$ como hijo derecho de f, ponemos p en ese nodo y como su hijo izquierdo a s. Finalmente con esto tenemos el árbol cartesiano de $A[i+1]$ con p apuntando a $A[i + 1]$ que es el nodo más derecho.

Para mostrar que este algoritmo toma tiempo $\mathcal{O}(n)$ veremos que las operaciones toman tiempo amortizado $\mathcal{O}(1)$.

Si definimos el potencial ϕ_i como la profundidad de p luego de la iteración i entonces:

- Si estamos en el caso $A[i+1] > A[i]$, entonces el potencial aumenta en 1, pero el costo real es constante por lo que el costo amortizado también lo es.
- Si es el caso que $A[i + 1] < A[i]$, y $A[j]$ esta a distancia k de p entonces el cambio de potencial es $-k + \Theta(1)$, pero por otro lado el costo es $k + \Theta(1)$, por lo que el costo amortizado resulta ser $\Theta(1)$.