Pauta de Corrección

Segundo Certamen
Algoritmos y Complejidad

3 de julio de 2017

1. Un algoritmo voraz obvio es ordenar los puntos en orden de x creciente, y luego iniciar el primer intervalo en el primer x, cubriendo todos los que están en $[x, x+1]$, y repetir con los no cubiertos.

   Veamos nuestras tres propiedades, suponiendo que los puntos están ordenados tal que $x_1 < x_2 < \cdots < x_n$.

   Elección voraz: Toda solución debe cubrir x_1, no tiene sentido que el intervalo que lo cubre no comience en x_1 (¿Para qué cubrir puntos anteriores a x_1?). O sea, hay una solución óptima que incluye el intervalo $[x_1, x_1+1]$.

   Estructura inductiva: Elegir el intervalo $[x_1, x_1+1]$ cubre algunos x, que quedan excluidos; debemos cubrir a los restantes, y el intervalo elegido no interfiere en ello. No hay restricciones externas.

   Subestructura óptima: La solución óptima de cubrir a los que no caen en $[x_1, x_1+1]$ se combina con ese intervalo para dar solución óptima al problema.

   Vimos que si se dan las tres propiedades, el algoritmo voraz da una solución óptima.

Puntajes

Total			30
Plantear algoritmo voraz		5	
Propiedades		20	
• Elección voraz	6		
• Estructura inductiva	7		
• Subestructura óptima	7		
Se cumplen las propiedades, da una solución óptima		5	

2. Analizamos cada algoritmo por turno para describir sus tiempos de ejecución.

   Algoritmo A: La recurrencia para el tiempo de ejecución es:

   $$A(n) = 2A(n/3) + O(n \log n)$$

   Este es el caso del teorema maestro con $a = 2, b = 3$, el valor crítico es $\log_3 2 < 1$, con lo que estamos en el último caso, ya que $n \log n = \Omega(n^1)$. La condición del último caso se cumple con $k = 2/3 < 1$, así que:

   $$A(n) = \Theta(n \log n)$$

   Algoritmo B: Acá la recurrencia es:

   $$B(n) = 2B(n-1) + c$$

   Podemos resolver esto mediante funciones generatices. Definimos:

   $$b(z) = \sum_{n \geq 0} B(n) z^n$$

   aplicando las propiedades:

   $$\frac{b(z) - B(0)}{z} = 2b(z) + \frac{c}{1-z}$$

   Resolviendo para $b(z)$, como fracciones parciales:

   $$b(z) = \frac{B(0) + c}{1 - 2z} - \frac{c}{1-z}$$

   de donde:

   $$\begin{aligned} B(n) &= [z^n]\, b(z) \\ &= (B(0) + c) \cdot 2^n - c \\ &= \Theta(2^n) \end{aligned}$$

   Más simple es hallar una cota, suficiente para nuestros requerimientos actuales. De la recurrencia:

   $$B(n) \geq 2B(n-1)$$

   de donde concluimos:

   $$B(n) = \Omega(2^n)$$

   Algoritmo C: La recurrencia es:

   $$C(n) = 3C(2n/3) + O(n)$$

   Acá el valor crítico para el teorema maestro es $\log_{3/2} 3 > 1$, es el primer caso y:

   $$C(n) = \Theta\left(n^{\log_{3/2} 3}\right)$$

   En realidad, $\log_{3/2} 3 = 2{,}7095$.

Se ve claramente que el algoritmo ganador es **A**.

Puntajes

Total			35
A		10	
Recurrencia	3		
Aplicar teorema maestro	7		
B		10	
Recurrencia	3		
Solución o cota de la recurrencia	7		
C		10	
Recurrencia	3		
Aplicar teorema maestro	7		
Conclusiones		5	

3. Para análisis amortizado, la alternativa que se ve más simple es el método potencial. Definimos la función potencial, donde n es el largo actual de la lista:

$$\Phi(n) = n$$

Es claro que $\Phi(n) \geq 0$, y el potencial inicial es $\Phi(0) = 0$.

Consideremos la secuencia de operaciones σ_i, de costos respectivos c_i y costos amortizados a_i, con estados s_i luego de la i-ésima operación:

$$a_i = c_i + \Phi(s_i) - \Phi(s_{i-1})$$

Si σ_i es insert(k), y en ese momento el largo de la lista era n, es:

$$\begin{aligned} a_i &= 1 + \Phi(n+1) - \Phi(n) \\ &= 1 + n + 1 - n \\ &= 2 \end{aligned}$$

Si σ_i es add(), y en ese momento el largo de la lista era n, es:

$$\begin{aligned} a_i &= n + \Phi(1) - \Phi(n) \\ &= n + 1 - n \\ &= 1 \end{aligned}$$

O sea, el costo amortizado de insert(k) es 2, el de add() es 1, ambos $O(1)$ como se indica.

Puntajes

Total		30
Planteo por método potencial	5	
Función potencial	10	
Cálculo de los costos amortizados	10	
Conclusión	5	

4. Vamos por turno. Suponemos funciones de hash ideales independientes h_i para simplificar el análisis.

   a) Esto corresponde a nk intentos fallidos de apuntarle a ese bit, con probabilidad de éxito $1/m$ en cada intento, o sea:

   $$\begin{aligned}\Pr[\text{bit } i \text{ en cero luego de agregar } n] &= \left(1 - \frac{1}{m}\right)^{nk} \\ &= \left(\left(1 - \frac{1}{m}\right)^{m}\right)^{nk/m} \\ &\approx \mathrm{e}^{-nk/m}\end{aligned}$$

   b) Esto corresponde a tener k éxitos (encontrar bit en 1) en k intentos independientes. La probabilidad de éxito en cada intento viene del punto anterior:

   $$\begin{aligned}\Pr[\text{falso positivo}] &= \prod_{1 \le i \le k} \Pr[\text{bit } h_i(x) \text{ es } 1] \\ &= \left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k} \\ &\approx \left(1 - \mathrm{e}^{-nk/m}\right)^{k}\end{aligned}$$

   Los filtros de Bloom balancean varios efectos contrarios: queremos usar poca memoria (m pequeño, pero eso hace más probables las colisiones, falsos positivos), poco tiempo de cómputo (minimizar k, usa menos memoria pero hace más probables las colisiones). Un análisis aproximado es como sigue. Sea $f(k)$ la función que nos interesa, queremos minimizar la probabilidad de falso positivo para m y n fijos, donde suponemos m lo suficientemente grande para poder aplicar la aproximación por la exponencial. Minimizar $f(k$ es minimizar $\ln f(k)$:

   $$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}k} \ln f(k) &= \frac{\mathrm{d}}{\mathrm{d}k} k \ln\left(1 - \mathrm{e}^{-nk/m}\right) \\ &= \ln\left(1 - \mathrm{e}^{-nk/m}\right) + \frac{nk}{m} \cdot \frac{\mathrm{e}^{-nk/m}}{1 - \mathrm{e}^{-nk/m}}\end{aligned}$$

   Igualando a cero la derivada, hallamos que el mínimo se da con $k = \frac{m}{n} \ln 2$, y este mínimo es global. En el mínimo, la probabilidad de falso positivo es:

   $$\left(\frac{1}{2}\right)^{k} = 0{,}6185^{m/n}$$

Conforme m crece respecto de n, disminuye la tasa de falsos positivos.

Puntajes

Total			30
a) Probabilidad de bit cero		15	
nk intentos independientes fallidos de probabilidad de éxito $1/m$	10		
Fórmula	5		
b) Probabilidad de falso positivo		15	
k intentos independientes exitosos	10		
Fórmula	5		

HvB/LaTeX 2_ε