Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC2283 - Diseño y Análisis de Algoritmos

Interrogación 1

21 de agosto de 2020

Profesor Marcelo Arenas

Bernardo Barías y Ricardo Rodríguez

Pregunta 1

Comenzamos realizando el paso inductivo. Nuestra hipótesis de inducción es que existe un $c \in \mathbb{R}^+$ y un n_0 tal que para todo $k < n$ se tiene que $T(k) \leq ck \log k$. A continuación mostramos que $T(n) \leq cn \log n$.

$$\begin{aligned}
T(n) &= T(\lfloor \frac{n}{3} \rfloor) + T(\lfloor \frac{2n}{3} \rfloor) + n \\
&\leq c\lfloor \frac{n}{3} \rfloor \log(\lfloor \frac{n}{3} \rfloor) + c\lfloor \frac{2n}{3} \rfloor \log(\lfloor \frac{2n}{3} \rfloor) + n \\
&\leq c\frac{n}{3} \log(\frac{n}{3}) + c\frac{2n}{3} \log(\frac{2n}{3}) + n \\
&= c\frac{n}{3} \log n - c\frac{n}{3} \log 3 + c\frac{2n}{3} \log n + c\frac{2n}{3} \log 2 - c\frac{2n}{3} \log 3 + n \\
&= cn \log n - cn \log 3 + c\frac{2n}{3} + n \\
&= cn \log n + n(-c \log 3 + \frac{2c}{3} + 1), \qquad \text{tomando } c > \frac{1}{\log 3 - 2/3} \\
&\leq cn \log n
\end{aligned}$$

La última desigualdad se obtiene del hecho que

$$\begin{aligned}
& -c \log 3 + \frac{2c}{3} + 1 < 0 \\
\Leftrightarrow \quad & 1 < c \log 3 - \frac{2c}{3} \\
\Leftrightarrow \quad & \frac{1}{\log 3 - \frac{2}{3}} < c
\end{aligned}$$

Para los casos base fijamos $c = 3 > \frac{1}{\log 3 - \frac{2}{3}}$. Podemos ver que

$$\begin{aligned}
T(0) &= 1 \not\leq 3 \cdot 0 \log 0 = 0 \\
T(1) &= T(0) + T(0) + 1 = 3 \not\leq 3 \cdot 1 \log 1 = 0 \\
T(2) &= T(0) + T(1) + 2 = 6 \leq 3 \cdot 2 \log 2 = 6 \\
T(3) &= T(1) + T(2) + 3 = 12 \leq 3 \cdot 3 \log 3 \approx 14{,}27 \\
T(4) &= T(1) + T(2) + 4 = 13 \leq 3 \cdot 4 \log 4 = 24 \\
T(5) &= T(1) + T(3) + 5 = 20 \leq 3 \cdot 5 \log 5 \approx 34{,}83
\end{aligned}$$

Como $T(6)$ depende de $T(2)$ y $T(4)$, y en ambos se cumple la propiedad, entonces $n = 6$ ya no es caso base.

Nota

La asignación de puntaje en esta pregunta fue de 0 a 6, pero como la pregunta tiene un total de 1.5 puntos de la prueba, la asignación final es la que se escribió en el pdf de la corrección dividido por 4.

Pregunta 2

a) Existen muchos ejemplos distintos, se asignaron 0,25 puntos por dar un ejemplo correcto y 0,25 puntos por demostrar correctamente que lo es.

b) Se dio 0,5 puntos por probar que $g \in O(f)$ y 0,5 puntos por probar que $f \notin O(g)$. Para probar que $g \in O(f)$, basta ver que la condición de la definición de o es para cualquier constante c, en cambio para O necesitamos que exista una constante que cumpla la condición, basta usar cualquiera, por ejemplo $c = 1$.

Para probar que $f \notin O(g)$ lo hacemos por contradicción, supongamos que $f \in O(g)$, luego por la definición de O existe $M \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $\forall n \geq n_0$ se tiene que

$$f(n) \leq Mg(n)$$

Esto es equivalente a

$$\frac{1}{M} f(n) \leq g(n)$$

Por otro lado, como $g \in o(f)$, para la constante $\frac{1}{M}$ existe un n_0' tal que para $n \geq n_0'$ se cumple

$$g(n) \leq \frac{1}{M} f(n)$$

Combinando ambas desigualdades se obtiene que $g(n) = \frac{1}{M} f(n)$, ahora bien, como $g \in o(f)$, tenemos que para toda constante C existe un natural n_0'' tal que si $n \geq n_0''$ se tiene que $g(n) \leq Cf(n)$, Sea k el máximo entre n_0'' y el punto desde donde f empieza a ser positiva, si $n \geq k$ tenemos que

$$g(n) \leq Cf(n) \quad \iff \quad \frac{1}{M} f(n) \leq Cf(n) \quad \iff \quad \frac{1}{M} \leq C$$

Recordemos que C es una constante arbitraria y la propiedad se cumple para cualquier valor de ella que elijamos, sea $C < \frac{1}{M}$, de acá $C < \frac{1}{M}$ por hipotesis y por lo probado arriba se tiene que al mismo tiempo $\frac{1}{M} \leq C$, lo que es un contradicción, luego $f \notin o(g)$

Pregunta 3

Se asignaron 0.75 puntos por dar un algoritmo correcto y 0.75 puntos por un análisis

correcto de su complejidad Un punto de partida es el algoritmo para edit distance que se vio en clases, $EditDistance(w_1, i, w_2, j)$, consideremos la tabla que se construye al implementar ese algoritmo via programación dinámica botton-up, la podemos recorrer desde la posicion inferior derecha e ir deduciendo la operación a realizar según los costos adyacentes.

En el análisis de complejidad, basta explicar el orden asíntotico de cada parte del algoritmo y acotar la cantidad de veces que se ejecuta cada parte.

Pregunta 4

a) El algoritmo retorna los coeficientes del polinomio representado por los coeficientes $\bar{a}$ (digamos $p(x)$) elevado a $k = 2^i$. A continuación lo demostraremos por inducción sobre i.

Para el caso base tomamos $i = 0$, y obtenemos que $k = 2^0 = 1$, y vemos que el algoritmo retorna $\bar{a}$, que son justamente los coeficientes de $p(x)^1$.

Supongamos que para $i-1$ el algoritmo con $k' = 2^{i-1}$ retorna los coeficientes del polinomio $p(x)^{k'}$. Mostraremos la propiedad para $k = 2^i$. Por la hipótesis de inducción, $(b_0, ..., b_{n-1})$ corresponden a los coeficientes de $p(x)^{k/2} = p(x)^{k'}$. Luego,

- $\bar{c}$ es $\bar{b}$ rellenado con ceros hasta tener largo potencia de 2.
- $\bar{y}$ es la representación punto valor de $\bar{c}$.
- $\bar{z}$ es la representación punto valor del producto de $\bar{y}$ con $\bar{y}$.
- $\bar{d}$ es la representación canónica de $\bar{z}$, que era multiplicar el polinomio representado por $\bar{c}$ consigo mismo, que a la vez es el polinomio $\bar{b}$ multiplicado consigo mismo. Pero $\bar{b}$ representaba $p(x)^{k/2}$. Luego, $\bar{d}$ representa $p(x)^{k/2} \cdot p(x)^{k/2} = p(x)^k$.

Entonces, concluimos que la propiedad se cumple para i.

b) La ecuación de recurrencia para el algoritmo es la siguiente:

$$\begin{aligned} T(\ell, k) &= T(\ell, \frac{k}{2}) + T(FFT(m)) + m + T(FFT(m)) + c \\ &= T(\ell, \frac{k}{2}) + 2m \log m + m + c \end{aligned}$$

Podemos acotar m:

$$\begin{aligned} m &= 2^{\lceil \log 2n \rceil} \\ &= 2^{\lceil \log 2 + \log n \rceil} \\ &= 2^{\lceil 1 + \log n \rceil} \\ &\leq 2^{1 + \log n + 1} \\ &= 4n \end{aligned}$$

También podemos acotar n. La intuición es que n está acotado por ℓk, ya que al multiplicar k veces un polinomio de grado $\ell - 1$ nos queda un polinomio de ese grado. Para demostrarlo formalmente, sea k fijo y n_k el largo de $\bar{b} = ALG(\bar{a}, k/2)$. Vamos a demostrar que $n_k \leq \ell k$ por inducción k.

Para el caso base, sea $k = 2$. Luego, $(b_0, ..., b_{n_k - 1}) = ALG(\bar{a}, 1) = \bar{a}$. Entonces, $n_k = |\bar{a}| = \ell$.

Para el caso inductivo suponemos que $n_r \leq \ell r$ para todo $r < k$, y debemos demostrar que $n_k \leq \ell k$.

Sabemos que

$$(b_0, ..., b_{n_k - 1}) = ALG(\bar{a}, k/2) = \underbrace{(d_0, ..., d_{2n_{k/2} - 2})}_{\text{esto retorna el algoritmo}}$$

Luego, como $n_k - 1 \leq 2n_{k/2} - 2$, entonces

$$\begin{aligned} n_k = 2n_{k/2} - 1 &\leq 2n_{k/2}, \qquad \text{aplicando la HI nos queda que} \\ &\leq 2\ell k/2 \\ &= \ell k \end{aligned}$$

Luego, hemos demostrado que $n_k \leq \ell k$. Esto quiere decir que $m \leq 4n_k \leq 4\ell k$.

Por último, como el algoritmo hace recursión un total de $\log k$ veces, y en cada llamada recursiva hace a lo más $O(m \log m)$ operaciones, entonces podemos acotar $T(\ell, k)$ por

$$\log k \cdot O(m \log m) = \log k \cdot O(4\ell k \log 4\ell k) = O(\log k \cdot \ell k \log \ell k)$$