Pontificia Universidad Católica de Chile
Facultad de Ingenieria
Departamento de Ciencias de la Computación
IIC2283 - Diseño y Análisis de Algoritmos

Ayudantía 1

24 de agosto de 2018

Ayudante: Andrés Ríos

Definiciones

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ | (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq c \cdot f(n))\}$$
$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ | (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c \cdot f(n) \leq g(n))\}$$
$$\Theta(f) = O(f) \cap \Omega(f)$$

Problema 1

Demuestre la veracidad o falsedad de las siguientes afirmaciones:

1. $2^{2n} \in O(2^n)$

   Solución:

   Por definición, para que $2^{2n} \in O(2^n)$ tendría que darse que

   $$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(2^{2n} \leq c \cdot 2^n)$$

   Simplificando la desigualdad, necesitamos que

   $$2^n \cdot 2^n \leq c \cdot 2^n$$

   $$2^n \leq c$$

   Esto es una contradicción: por muy grande que sea la constante c, siempre vamos a poder elegir un valor de n lo suficientemente grande tal que $2^n \not\leq c$ (basta $N = \lceil log_2(c) \rceil + 1$). Por lo tanto, $2^{2n} \notin O(2^n)$

2. $2^{n+1} \in \Theta(2^n)$

   Solución:

   Tenemos que $2^{n+1} = 2 \cdot 2^n$. Por lo tanto, eligiendo $c = 2$ se tiene que $2^{n+1} \in \Omega(2^n)$. Similarmente, con $c = \frac{1}{2}$ se tiene que $2^{n+1} \in O(2^n)$. Por lo tanto, $2^{n+1} \in \Theta(2^n)$.

3. $n! \in O(n^n)$

   Solución:

   Tenemos que $n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1 \leq n \cdot n \cdot n \cdot ... \cdot n = n^n$. Por lo tanto, con $c = 1$ se tiene que $n! \in O(n^n)$.

4. $n! \in \Omega(2^n)$

   Solución:

   Tenemos que $n! = n \cdot (n-1) \cdot (n-2) \cdot ... \cdot 1 \geq 2 \cdot 2 \cdot 2 \cdot ... \cdot 2 = 2^n$ para $n \geq 4$. Por lo tanto, con $c = 1, n_0 = 4$ se tiene que $n! \in \Omega(2^n)$

5. $n! \in \Omega(n^n)$

   Solución:

   Tenemos que $\lim_{n\to\infty} \frac{n!}{n^n} = \lim_{n\to\infty} \frac{n(n-1)...1}{n \cdot n \cdot ... \cdot n} \leq \lim_{n\to\infty} \frac{1}{n} = 0$

   Por lo tanto, $n^n \notin O(n!)$ (ver problema 3). Esto implica que $n! \notin \Omega(n^n)$.

6. $\log(n^n) \in O(\log(n!))$

   Solución:

   Se van a probar las siguientes dos afirmaciones:

   a) $\log((\frac{n}{2})^{\frac{n}{2}}) \in O(\log(n!))$

   b) $\log(n^n) \in O(\log((\frac{n}{2})^{\frac{n}{2}}))$

   Luego estas dos afirmaciones en conjunto implican que $log(n^n) \in O(log(n!))$.

   a) $\log((\frac{n}{2})^{\frac{n}{2}}) \in O(\log(n!))$

   Tenemos dos casos: cuando n es par y cuando n es impar.

   Caso par:

$$\begin{aligned} n! &= (2i)! \\ &= (2i)(2i-1)(2i-2)...(i+1)(i)(i-1)...1 \\ &> (2i)(2i-1)(2i-2)...(i+1) \\ &> i \cdot i \cdot i \cdot ... \cdot i \\ &= i^i \\ &= \left(\frac{n}{2}\right)^{\frac{n}{2}} \end{aligned}$$

   Caso impar:

$$\begin{aligned} n! &= (2i+1)! \\ &= (2i+1)(2i)(2i-1)...(i+1)(i)(i-1)...1 \\ &> (2i+1)(2i)(2i-1)(2i-2)...(i+1) \\ &> (i+1) \cdot (i+1) \cdot (i+1) \cdot ... \cdot (i+1) \\ &= (i+1)^{(i+1)} \\ &> \left(\frac{n}{2}\right)^{\frac{n}{2}} \end{aligned}$$

   Por lo tanto, $\left(\frac{n}{2}\right)^{\frac{n}{2}} \in O(n!)$ lo que implica $\log(\left(\frac{n}{2}\right)^{\frac{n}{2}}) \in O(\log(n!))$

   b) $\log(n^n) \in O(\log((\frac{n}{2})^{\frac{n}{2}}))$

   Asumamos que $\log(n^n) \leq c \cdot \log((\frac{n}{2})^{\frac{n}{2}})$ e intentemos obtener los valores correctos para n_0 y c.

$$\begin{aligned}\log(n^n) = n\log(n) &\leq c \cdot \log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)\\ &= \frac{c}{2}n\log\left(\frac{n}{2}\right)\\ &= \frac{c}{2}n(\log(n) - \log(2))\\ &= \frac{c}{2}n\log(n) - \frac{c}{2}n\log(2)\\ &= \frac{c}{2}n\log(n) - \frac{c}{2}n\end{aligned}$$

$$\begin{aligned}n\log(n) &\leq \frac{c}{2}n\log(n) - \frac{c}{2}n\\ \frac{c}{2}n &\leq \left(\frac{c}{2} - 1\right)n\log(n)\\ \frac{c}{2} &\leq \left(\frac{c}{2} - 1\right)\log(n)\end{aligned}$$

Si tomamos $c = 3$, es fácil ver que la inecuación anterior se cumple para todo $n \geq 8$. Por lo tanto, eligiendo $c = 3, n_0 = 8$ tenemos que $\forall n \geq n_0 \;\; \log(n^n) \leq 3 \cdot \log((\frac{n}{2})^{\frac{n}{2}})$.

Esto implica que $\log(n^n) \in O(\log((\frac{n}{2})^{\frac{n}{2}}))$.

Como $\log(n^n) \in O(\log((\frac{n}{2})^{\frac{n}{2}}))$ y $\log((\frac{n}{2})^{\frac{n}{2}}) \in O(\log(n!))$, se concluye que $\log(n^n) \in O(\log(n!))$.

7. Si $f(n) \in O(g(n))$ y $h(n) : \mathbb{N} \to \mathbb{N}, h(n) \geq n \; \forall n > n_1$, entonces $f(h(n)) \in O(g(h(n)))$

   Solución:

   Es verdadero.

   Se tiene que $\forall n \geq n_0, f(n) \leq c \cdot g(n)$. En particular, tomando $n = h(x)$ donde $x \geq max(n_0, n_1)$ tenemos lo pedido, pues $h(x) \geq n_0$ en este caso.

8. Si $f(n) \in \Theta(g(n))$, entonces $h(f(n)) \in \Theta(h(g(n)))$

   Solución:

   Falso. Contraejemplo:

$$f(n) = 2n$$
$$g(n) = n$$
$$h(n) = 2^n$$

   En la parte 1 se demostró que $h(f(n)) \notin O(h(g(n)))$ en este caso.

Problema 2

Determine la complejidad de las siguientes recurrencias en notación Θ. Puede asumir que n es de la forma b^i para algún b. (Tip: $\Sigma_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$)

1.
$$T(n) = \begin{cases}1 & x = 1\\ T(n-1) + n^2 & x > 1\end{cases}$$

   Solución:

   Expandiendo la recurrencia tenemos que

$$
\begin{aligned}
T(n) &= T(n-1) + n^2 \\
&= T(n-2) + (n-1)^2 + n^2 \\
&= ... \\
&= \sum_{i=1}^{n} i^2 \\
&= \frac{n(n+1)(2n+1)}{6} \in \Theta(n^3)
\end{aligned}
$$

2.

$$
T(n) = \begin{cases} 1 & x = 1 \\ 2 \cdot T(\lfloor n/2 \rfloor) + n^2 & x > 1 \end{cases}
$$

Solución:

Asumiendo que n es de la forma 2^i y expandiendo la recurrencia, tenemos que

$$
\begin{aligned}
T(n) &= 2T(2^{i-1}) + 2^i \\
&= 2(2T(2^{i-2}) + 2^{2(i-1)}) + 2^{2i} \\
&= 2^2 T(2^{i-2}) + 2^{2i-1} + 2^i \\
&= ... \\
&= 2^i T(1) + \sum_{j=0}^{i} 2^{2i-j} \\
&= 2^i T(1) + \sum_{j=0}^{i} 2^{i+j} \\
&= 2^i T(1) + 2^i \sum_{j=0}^{i} 2^j \\
&= 2^i T(1) + 2^i \frac{2^{i+1} - 1}{2 - 1} \\
&= n + n(2n-1) \\
&= 2n^2 \in \Theta(n^2)
\end{aligned}
$$

Problema 3

1. Dado $f(n), g(n)$ y asumiendo que $\lim_{n\to\infty} \frac{f(n)}{g(n)} = l$, demuestre que

   a) $l = 0 \implies f(n) \in O(g(n))$ y $g(n) \notin O(f(n))$

   Solución:

   Recordar definición de límite al infinito:

   $$\lim_{n\to\infty} h(n) = l \iff (\forall \varepsilon > 0)(\exists n_0)(\forall n \geq n_0) \quad |h(n) - l| < \varepsilon$$

   Con esto, tenemos que

   $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \implies (\forall \varepsilon > 0)(\exists n_0)(\forall n \geq n_0) \frac{f(n)}{g(n)} < \varepsilon$$

   Tomamos $\varepsilon = 1$ para obtener

$$(\exists n_0)(\forall n \geq n_0)\frac{f(n)}{g(n)} < 1$$
$$(\exists n_0)(\forall n \geq n_0)f(n) < g(n)$$

Lo que implica que $f(n) \in O(g(n))$. Falta demostrar que $g(n) \notin O(f(n))$.

Supongamos que $g(n) \in O(f(n))$ e intentemos llegar a una contradicción. Tenemos que

$$(\exists c \in \mathbb{R}^+)(\exists n_0)(\forall n \geq n_0)g(n) \leq c \cdot f(n)$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0)(\forall n \geq n_0)\frac{1}{c} \leq \frac{f(n)}{g(n)}$$

Pero esto es una contradicción pues $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (ver definicion de límite). Por lo tanto, nuestra suposición era falsa y $g(n) \notin O(f(n))$.

b) $l = c$ con $c \in \mathbb{R}^+ \implies f(n) \in \Theta(g(n))$

Solución:

Por definición de límite,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \implies (\forall \varepsilon > 0)(\exists n_0)(\forall n \geq n_0)\left|\frac{f(n)}{g(n)} - c\right| < \varepsilon$$

De la desigualdad se obtiene que

$$\frac{f(n)}{g(n)} - c < \varepsilon$$
$$c - \frac{f(n)}{g(n)} < \varepsilon$$

Lo que implica que

$$f(n) < (\varepsilon + c)g(n)$$
$$(c - \varepsilon)g(n) < f(n)$$

Tomando $\varepsilon = \frac{c}{2}$:

$$f(n) < \frac{3}{2}cg(n)$$
$$\frac{1}{2}cg(n) < f(n)$$

Esto implica que $f(n) \in \Theta(g(n))$.

c) $l = \infty \implies g(n) \in O(f(n))$ y $f(n) \notin O(f(n))$

Es análogo a la parte a).

2. De un ejemplo de funciones f y g tales que $f(n) \in \Theta(g(n))$ pero $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ no existe.

Solución:

Un ejemplo sería $f(n) = sin(n) + 2$, $g(n) = cos(n) + 2$.

Problema 4

1. Determine la complejidad de Merge Sort (puede asumir que n es de la forma 2^i)

   Solución:

   El tiempo de Merge Sort puede ser definido por la recurrencia:

$$T(n) = \begin{cases} 1 & x = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n & x > 1 \end{cases}$$

   Asumiendo que n es de la forma 2^i, resulta

$$T(n) = \begin{cases} 1 & x = 1 \\ 2T(n/2) + n & x > 1 \end{cases}$$

   Expandimos la recurrencia:

$$\begin{aligned} T(2^i) &= 2T(2^{i-1}) + 2^i \\ &= 2(2T(2^{i-2}) + 2^{i-1}) + 2^i \\ &= 2^2 T(2^{i-2}) + 2 \cdot 2^i \\ &= 2^3 T(2^{i-3}) + 3 \cdot 2^i \\ &= ... \\ &= 2^i \cdot T(1) + i \cdot 2^i \\ &= n + \log(n) \cdot n \in \Theta(n \log(n)) \end{aligned}$$

2. Un indice mágico en un arreglo $A[0...n-1]$ se define como un indice i tal que $A[i] = i$. Dado un arreglo ordenado sin elementos repetidos, entregue un algoritmo que encuentre un indice mágico (si existe) y analice la complejidad del algoritmo. El algoritmo debe ser más eficiente que simplemente revisar el arreglo elemento por elemento.

   Solución:

```
function IM(arr, offset)
    if arr.length = 1 then
        if arr[0] = offset then return offset
        else   return null
    idx ← ⌊arr.length/2⌋
    n ← arr[idx]
    if n = idx then return idx + offset
    if n < idx then return IM(arr[idx + 1 :], offset + idx + 1)
    return IM(arr[: idx], offset)
```

   El tiempo de ejecución de este algoritmo está dado por

$$T(n) = \begin{cases} 1 & x = 1 \\ T(\lceil n/2 \rceil) + 1 & x > 1 \end{cases}$$

   Notar que es la misma recurrencia que caracteriza a la búsqueda binaria. Asumiendo $n = 2^i$, esta recurrencia se puede expandir de manera análoga al problema 4.1 para obtener $T(n) \in O(\log(n))$.

3. Dado un conjunto de números $Q = \{q_1, ..., q_m\}$ en los naturales se busca un polinomio p tal que $p(q) = 0 \ \ \forall q \in Q$.

- Entregue el polinomio buscado como multiplicación de polinomios de grado 1.

  Solución:

  El polinomio buscado es $\Pi_{i=1}^{m}(x - q_i)$.

- ¿Cual es la complejidad de un algoritmo que simplemente multiplica los polinomios de grado 1 de manera ingenua?

  Solución:

  Lo que el algoritmo haría es multiplicar un polinomio de grado 1 con otro de grado 1, luego uno de grado 2 con uno de grado 1, uno de grado 3 con uno de grado 1, etc. El costo de multiplicar un polinomio de grado 1 por otro de grado n es proporcional a n. Por lo tanto, el costo es

$$T(n) = \sum_{i=1}^{m} c \cdot i = c\frac{n(n+1)}{2} \in \Theta(n^2)$$

- Asumiendo que tiene una funcion $M(\cdot, \cdot)$ que toma dos polinomios de grado n en su forma canónica y retorna un polinomio de grado $2n$ (también en forma canónica) en tiempo $O(n \cdot log(n))$, entregue un algoritmo que resuelva el problema y funcione en tiempo menor que cuadrático. Puede asumir que m es de la forma 2^i. Entregue la complejidad del algoritmo en notación O.

  Solución:

  Un algoritmo sería

  function EncontrarPolinomio($q_1, ..., q_m$)
  if $m = 1$ **then**
  return $(x - q_1)$
  $l \leftarrow$ EncontrarPolinomio($q_1, ..., q_{\lfloor m/2 \rfloor}$)
  $r \leftarrow$ EncontrarPolinomio($q_{\lfloor m/2 \rfloor + 1}, ..., q_m$)
  return M(l, r)

  El tiempo de ejecución de este algoritmo puede ser acotado por la recurrencia

$$T(m) \leq \begin{cases} 1 & m = 1 \\ 2T(\frac{m}{2}) + k \cdot \frac{m}{2}\log(\frac{m}{2}) & m > 1 \end{cases}$$

  Asumiendo que m es de la forma 2^i, podemos desarrollar esta recurrencia:

$$\begin{aligned} T(2^i) &\leq 2T(2^{l-1}) + k2^{l-1}(l-1) \\ &\leq 2(2T(2^{l-2}) + k2^{l-2}(l-2)) + k2^{l-1}(l-1) \\ &= 2^2T(2^{l-2}) + k2^{l-1}(l-2) + k2^{l-1}(l-1) \\ &\leq ... \\ &\leq 2^iT(1) + \sum_{i=1}^{l-1} k2^{l-1}i \\ &= m + k2^{l-1}\sum_{i=1}^{l-1} i \\ &= m + k\frac{m}{2}\frac{i(i-1)}{2} \\ &\leq m + kmi^2 \\ &= m + km\log^2(n) \in O(m\log^2(m)) \end{aligned}$$