Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC1253 — Matemáticas Discretas — 1' 2019

Ayudantía 9

Notación asintótica

Problema 1

Demuestre usando la definición de notación $\mathcal{O}$ que

$$\log(a_k n^k + \ldots + a_1 n + a_0) \in \mathcal{O}(\log(n))$$

con $k \geq 0$ y $a_i \geq 0$ para todo $i \leq k$.

Solucion propuesta.

Debemos obtener una constante real $c > 0$ y un natural n_0 tales que

$$\log(a_k n^k + \ldots + a_1 n + a_0) \leq c \log(n), \ \forall n \geq n_0$$

para probar lo pedido. Para ello, primero notamos que

$$a_k n^k + \ldots + a_1 n + a_0 \leq a_k n^k + \ldots + a_1 n^k + a_0 n^k$$

para todo $n \geq 1$. Luego,

$$\begin{aligned}
\log(a_k n^k + \ldots + a_1 n + a_0) &\leq \log(a_k n^k + \ldots + a_1 n^k + a_0 n^k) && \text{logaritmo creciente} \\
&= \log\left(\textstyle\sum_{i=0}^{k} a_i n^k\right) && \text{agrupando} \\
&= \log\left(\textstyle\sum_{i=0}^{k} a_i\right) + \log(n^k) && \text{logaritmo de un producto} \\
&= \log\left(\textstyle\sum_{i=0}^{k} a_i\right) + k \log(n) && \text{logaritmo de una potencia}
\end{aligned}$$

Notemos que $\log(n) \geq 1$ para todo $n \geq b$, donde b es la base del logaritmo. Entonces, para todo $n \geq b$ se tiene

$$\begin{aligned}
\log\left(\textstyle\sum_{i=0}^{k} a_i\right) + k \log(n) &\leq \log\left(\textstyle\sum_{i=0}^{k} a_i\right) \log(n) + k \log(n) && \text{pues } \log(n) \geq 1 \\
&= \left(\log\left(\textstyle\sum_{i=0}^{k} a_i\right) + k\right) \log(n) && \text{factorizando}
\end{aligned}$$

Como lo que acompaña a $\log(n)$ no depende de n, es la constante c que estábamos buscando, y dado que esta inecuación es válida para $n \geq b$, tenemos $n_0 = b$. La existencia de c y n_0 demuestran que la función original es $\mathcal{O}(\log(n))$.

Problema 2

Sea $f : \mathbb{N} \to \mathbb{R}^+$ y $g : \mathbb{N} \to \mathbb{R}^+$ donde $\mathbb{R}^+$ son todos los reales positivos mayores que 0. Demuestre que si $f \in \mathcal{O}(g)$ entonces existe $c^* > 0$ tal que para todo $n \in \mathbb{N}$ se cumple que $f(n) \leq c^* \cdot g(n)$.

Solución propuesta.

A diferencia de la definición de $\mathcal{O}$-grande, la inecuación $f(n) \leq c^* \cdot g(n)$ debe cumplirse para cualquier natural n y no solo para aquellos a partir de un cierto n_0. Para demostrar que esto es posible, usaremos la definición de $\mathcal{O}$-grande.

Por enunciado, sabemos que $f \in \mathcal{O}(g)$ y por lo tanto, existen c, n_0 tales que para todo $n \geq n_0$, se tiene que $f(n) \leq c \cdot g(n)$. Si $n_0 = 0$, basta tomar $c^* = c$ y se demuestra lo pedido. Ahora nos concentramos en el caso $n_0 > 0$.

Dado que ninguna de las funciones es cero, en particular g, el cociente $f(n)/g(n)$ está bien definido y nos indica qué tanto más grande es f que g para cada n. Luego, consideremos la constante

$$c' = \max_{n \in \{0,\ldots,n_0-1\}} \left\{ \frac{f(n)}{g(n)} \right\}$$

Esta constante nos indica el máximo de las razones entre ambas funciones para el tramo donde no conocíamos una constante adecuada. En efecto, definiendo

$$c^* = \max\{c, c'\}$$

obtenemos la constante pedida:

- Si $n < n_0$, entonces $c^* \cdot g(n) \geq c' \cdot g(n) \geq \frac{f(n)}{g(n)} g(n) = f(n)$.
- Si $n \geq n_0$, entonces $c^* \cdot g(n) \geq c \cdot g(n) \geq f(n)$.

Por lo tanto, para todo $n \in \mathbb{N}$, se tiene que $f(n) \leq c^* \cdot g(n)$.

Problema 3

Demuestre que $\log_2(n!) \in \Theta(n \log_2(n))$ usando la definición de notación Θ (no puede usar límites). Para esta demostración, puede asumir la fórmula de Stirling:

$$n! \in \Theta\left(\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right)$$

Solución propuesta

Debemos probar que $\log_2(n!) \in \mathcal{O}(n \log_2(n))$ y que $n \log_2(n) \in \mathcal{O}(\log_2(n!))$.

- Desarrollando el factorial, notamos que

  $$n! = 1 \cdot 2 \cdot \cdots \cdot n \leq n \cdot n \cdot \cdots \cdot n = n^n$$

  por lo cual, y dado que el logaritmo es creciente,

  $$\log_2(n!) \leq \log_2(n^n) = n \log_2(n)$$

  Luego, las constantes que atestiguan la definición de $\mathcal{O}$-grande pedida son $c = 1$ (pues obtuvimos exactamente la función $n \log_2(n)$ como cota superior de $\log_2(n!)$) y $n_0 = 1$ (pues en cero no está definido el logaritmo).

- Omitiremos la base del logaritmo por simplicidad. De la fórmula de Stirling sabemos que existen c', n_0' tales que para todo $n \geq n_0$,

$$\sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \leq c \cdot n! \quad \Leftrightarrow \quad \frac{1}{c} \cdot \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n \leq n!$$

Por simplicidad definimos la constante $d = 1/c$. Aplicando logaritmo y separando los productos, obtenemos

$$\log\left(d \cdot \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n\right) \leq \log(n!)$$

$$\Leftrightarrow \quad \log(d) + \frac{1}{2}\log(\pi) + \frac{1}{2}\log(n) + \log(n^n) - \log(e^n) \leq \log(n!)$$

Reagrupando, y usando que $\log(n) \geq 1$ para todo $n \geq 2$, obtenemos

$$\log(n^n) - \log(e^n) \leq \log(n!) - \log(d) - \frac{1}{2}\log(\pi) - \frac{1}{2}\log(n) \leq \log(n!)$$

Como $e^n \in \mathcal{O}(n!)$, hay constantes c'', n_0'' tales que $e^n \leq c''n!$ para todo $n \geq n_0''$. Con esto,

$$\log(n^n) \leq \log(n!) + \log(e^n) \leq (c'' + 1)\log(n!)$$

Con esto, podemos definir $c = (c'' + 1)$ y tomar $n_0 = \text{máx}\{n_0', 2, n_0''\}$ de forma que

$$n\log(n) \leq c\log(n!), \quad \forall n \geq n_0$$

lo que prueba que $n\log(n) \in \mathcal{O}(\log(n!))$.

Problema 4

Considere el siguiente algoritmo:

```
Function theavengersaredead (n)
    k := 1
    for i = 1 to n do
        k := k * n
    i := 1
    while i ≤ k do
        i := i * 2
    return i
```

Encuentre una función f y demuestre (usando la definición formal de la notación Θ) que el tiempo de `theavengersaredead` en términos de n es $\Theta(f(n))$.

Solución propuesta.

Sea $T(n)$ el tiempo (en número de pasos) del algoritmo con input n. El **for** del algoritmo se ejecuta n veces, luego de las cuales la variable k tiene valor $k = n^n$.

Luego, en cada iteración del bloque **while** la variable i es de la forma $i = 2^\ell$, donde ℓ es el número de iteraciones del **while** en ese momento. El número de iteraciones se deduce de la condición del **while** apenas deja de satisfacerse, i.e.

$$2^\ell > n^n \quad \Leftrightarrow \quad \ell > n\log_2(n)$$

por lo que deducimos que el bloque **while** ejecuta $n\log(n)$ iteraciones.

Con lo anterior, el número de iteraciones del algoritmo completo es

$$T(n) = n + n\log(n) + \mathcal{O}(1)$$

donde el término $\mathcal{O}(1)$ alude a las operaciones que se ejecutan una cantidad fija de veces, no dependiente del input n (por ejemplo, la instrucción $k := 1$ al comienzo del algoritmo).

Tomando $f(n) = n\log(n)$, tenemos que $T(n) \in \Theta(f(n))$. En efecto,

- $T(n) \in \mathcal{O}(f(n))$ pues para $n \geq 1$, $\log(n) \geq 1$ y por lo tanto $n \leq n\log(n)$. Luego, como además cualquier constante m cumple $m \in \mathcal{O}(n\log(n))$,

$$n + n\log(n) + \mathcal{O}(1) \leq c \cdot n\log(n)$$

- $n\log(n) \in \mathcal{O}(T(n))$ pues basta tomar $c = 1$ y $n_0 = 1$,

$$n\log(n) \leq n + n\log(n) + \mathcal{O}(1) = T(n)$$