Complejidad algorítmica

Clase de repaso

Algoritmos y Estructuras de Datos II

Departamento de Computación

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

2do cuatrimestre 2020

Menú del día

Definiciones

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\mathcal{O}(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R} \mid (\exists\ n_0 \in \mathbb{N},\ c \in \mathbb{R}_{>0})\ \ f(n) \leq c \cdot g(n)\ \ \forall n \geq n_0\}$$

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\Omega(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R} \mid (\exists\ n_0 \in \mathbb{N},\ c \in \mathbb{R}_{>0})\ \ f(n) \geq c \cdot g(n)\ \ \forall n \geq n_0\}$$

Definición

Sea $g : \mathbb{N} \to \mathbb{R}$. Entonces:

$$\Theta(g) \stackrel{\text{def}}{=} \{f : \mathbb{N} \to \mathbb{R}\ \mid\ f \in O(g) \land f \in \Omega(g)\}$$

Propiedades – $\diamondsuit$ es "comodín" de $\mathcal{O}, \Omega, \Theta$

1. Toda f cumple $f \in \diamondsuit(f)$. Reflexiva
2. $f \in \diamondsuit(g) \implies k \cdot f \in \diamondsuit(g)$ (con k cte.)
3. Regla de la suma:
   $f_1 \in \diamondsuit(g) \wedge f_2 \in \diamondsuit(h) \implies f_1 + f_2 \in \diamondsuit(g+h) = \diamondsuit(\text{máx}\{g, h\})$
4. Regla del producto:
   $f_1 \in \diamondsuit(g) \wedge f_2 \in \diamondsuit(h) \implies f_1 \cdot f_2 \in \diamondsuit(g \cdot h)$

3 y 4 corresponden al **álgebra de órdenes**. Además 4 implica 2.

- $f \in \diamondsuit(g) \wedge g \in \diamondsuit(h) \implies f \in \diamondsuit(h)$ Transitiva
- $f \in \diamondsuit(g) \implies \diamondsuit(f) \subseteq \diamondsuit(g)$
- $\diamondsuit(f) = \diamondsuit(g) \iff f \in \diamondsuit(g) \wedge g \in \diamondsuit(f)$

  Como $f \in \Theta(g) \implies g \in \Theta(f)$ Simétrica
  - $\Theta(f) = \Theta(g) \iff f \in \Theta(g)$

Ejercicios: Inducción

Sea $f(n) = 2^n$, probemos que para todo n dado $f(n) \in \mathcal{O}(1)$

Ejercicios: Inducción

Sea $f(n) = 2^n$, probemos que para todo n dado $f(n) \in \mathcal{O}(1)$

¿Esto significa que $2^n \in \mathcal{O}(1)$?

¿Y cómo se hace?

Probemos que $n2^n \in \mathcal{O}(3^n)$.

Ejercicios

Demostrar que, vistas como funciones de $x \in \mathbb{R}_{\geq 0}$, vale:

$$x^p \in \mathcal{O}(b^x)$$

para toda base $b \in \mathbb{R}$ tal que $b > 1$ y para todo exponente $p \in \mathbb{N}$.

Ejercicios

Demostrar que, vistas como funciones de $x \in \mathbb{R}_{\geq 0}$, vale:

$$x^p \in \mathcal{O}(b^x)$$

para toda base $b \in \mathbb{R}$ tal que $b > 1$ y para todo exponente $p \in \mathbb{N}$.

Ayuda:

Dados $b \in \mathbb{R}$ tal que $b > 1$ y $k \in \mathbb{N}$ tales que $b^k \geq 2$ vale la siguiente desigualdad para todo $x \in \mathbb{R}_{\geq 0}$ y para todo $n, p \in \mathbb{N}$

$$\left(\frac{x}{pk}\right)^n \leq b^{n(\frac{x}{p}+k)}$$

Ejercicios: Algoritmo

DivisoresDePares(A : `arreglo(nat)`)

```
total ← 0
para i ← 0...tam(A) − 1 hacer
    si 2 divide a A[i] entonces
        para j ← 0...tam(A) − 1 hacer
            si A[j] divide a A[i] entonces
                total ← total + 1
    devolver total
```

Obs.: Consideramos operaciones elementales a las verificaciones de divisibilidad.

1. ¿La complejidad del *mejor caso* es $\mathcal{O}(n^2)$?
2. ¿La complejidad del *peor caso* es $\Omega(n)$?

Múltiples parámetros

Definición

$$O(g) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} f : \mathbb{N}^k \to \mathbb{R} \mid \exists \;\; \vec{n_0} \in \mathbb{N}^k, c \in \mathbb{R}_{>0} \\ \qquad\qquad\qquad\qquad f(\vec{n}) \leq c \cdot g(\vec{n}) \qquad \forall \vec{n} > \vec{n_0} \end{array} \right\}$$

Es decir, $f \in O(g)$ si y sólo si existen $\vec{n_0} \in \mathbb{N}^k$ y $c > 0$ tales que para todo $\vec{n} \geq \vec{n_0}$ se tiene:

$$f(\vec{n}) \leq c \cdot g(\vec{n})$$

Ejemplo: $m \log n = O(mn)$

Múltiples parámetros: Algoritmo múltiple

¡Último! Queremos saber si un elemento e está en una matriz de $N \times N$ o no.

BÚSQUEDAMATRICIAL(A : `arreglo(arreglo(nat))`, e : `nat`)

para $i \leftarrow 0 ... tam(A) - 1$ **hacer**
  para $j \leftarrow 0 ... tam(A[0]) - 1$ **hacer**
    si $A[i][j] == e$ **entonces**
      devolver *true*
devolver *false* $=0$

- ¿Cuál es la complejidad del *peor* caso? Fácil, ¿no?

Múltiples parámetros: Algoritmo múltiple

¡Último! Queremos saber si un elemento e está en una matriz de $N \times N$ o no.

BÚSQUEDAMATRICIAL(A : `arreglo(arreglo(nat))`, e : `nat`)

para $i \leftarrow 0 ... tam(A) - 1$ **hacer**
  para $j \leftarrow 0 ... tam(A[0]) - 1$ **hacer**
    si $A[i][j] == e$ **entonces**
      devolver *true*
devolver *false*

- ¿Cuál es la complejidad del *peor* caso? Fácil, ¿no?
- ¿Pero si es de $N \times M$?