Complejidad algorítmica

Clase práctica

Algoritmos y Estructuras de Datos

Departamento de Computación

DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

1er cuatrimestre 2024

Menú del día

Cotas de complejidad - $\mathcal{O}, \Omega, \Theta$

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

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

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

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

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

Ejercicio

Demostrar que:

- $f \in \mathcal{O}(g) \implies k * f \in O(g), \forall k \in \mathbb{R}_{>0}$
- $\mathcal{O}(n^2) \cap \Omega(1) \neq \Theta(n)$

Ejercicios

Decidir si son verdaderas o falsas y justificar:

1. $2^n = \mathcal{O}(1)$
2. $\Omega(n) \subseteq \mathcal{O}(n^2)$
3. $\mathcal{O}(n) \subseteq \Omega(n)$

Ejercicio

Demostrar que $n^2 + 5n + 3 \in \Omega(n)$, pero $n^2 + 5n + 3 \notin \mathcal{O}(n)$

Propiedad

Dadas las funciones $f,\ g : \mathbb{N} \to \mathbb{R}$. Si existe

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \ell \in \mathbb{R}_{\geq 0} \cup \{+\infty\}$$

Entonces:

- $f \in \Theta(g) \iff 0 < \ell < +\infty$
- $f \in \mathcal{O}(g)$ y $f \notin \Omega(g) \iff \ell = 0$
- $f \in \Omega(g)$ y $f \notin \mathcal{O}(g) \iff \ell = +\infty$

Primeros pasos – Análisis de complejidad

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)},\ e : \texttt{nat}) \longrightarrow \texttt{bool}$

var $i : nat, n : nat$
$n \leftarrow tam(A)$
$i \leftarrow 0$
mientras $i < n \wedge A[i] \neq e$ **hacer**
  $i \leftarrow i + 1$
devolver $(i < n)$

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm BUSQUEDASECUENCIAL(A : `arreglo(nat)`, e : `nat`) $\longrightarrow$ `bool`

```
1: var i : nat, n : nat
2: n ← tam(A)
3: i ← 0                                  ▷ línea 2-3: 3
4: mientras i < n ∧ A[i] ≠ e hacer        ▷ 4
5:     i ← i + 1                          ▷ 2
6: devolver (i < n)                       ▷ 2
```

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm BUSQUEDASECUENCIAL(A : `arreglo(nat)`, e : `nat`) $\longrightarrow$ `bool`

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\triangleright$ línea 2-3: 3
4: **mientras** $i < n \land A[i] \neq e$ **hacer** $\triangleright$ $(n+1) \cdot 4$
5: $\quad i \leftarrow i + 1$ $\triangleright$ 2
6: **devolver** $(i < n)$ $\triangleright$ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)},\ e : \texttt{nat}) \longrightarrow \texttt{bool}$

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ ▷ línea 2-3: 3
4: **mientras** $i < n \land A[i] \neq e$ **hacer** ▷ ciclo: $(n+1) \cdot 4 + 2 \cdot n$
5: $\quad i \leftarrow i + 1$ ▷ 2
6: **devolver** $(i < n)$ ▷ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)}, e : \texttt{nat}) \longrightarrow \texttt{bool}$

```
1: var i : nat, n : nat
2: n ← tam(A)
3: i ← 0                                        ▷ línea 2-3: 3
4: mientras i < n ∧ A[i] ≠ e hacer              ▷ ciclo: (n + 1) · 4 + 2 · n
5:     i ← i + 1                                ▷ 2
6: devolver (i < n)                             ▷ 2
```

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n + 1)\ 4 + 2n + 2 =$

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm BusquedaSecuencial(A : `arreglo(nat)`, e : `nat`) $\longrightarrow$ `bool`

```
1: var i : nat, n : nat
2: n ← tam(A)
3: i ← 0                                    ▷ línea 2-3: 3
4: mientras i < n ∧ A[i] ≠ e hacer          ▷ ciclo: (n + 1) · 4 + 2 · n
5:     i ← i + 1                            ▷ 2
6: devolver (i < n)                         ▷ 2
```

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n+1)\ 4 + 2n + 2 = \ 3 + 4 + 2 + \ (4 + 2)\ n$

Primeros pasos – Análisis del **peor** caso

Precondición: $|A| > 0$ (arreglo no vacío)

Algorithm $\textsc{BusquedaSecuencial}(A : \texttt{arreglo(nat)},\ e : \texttt{nat}) \longrightarrow \texttt{bool}$

1: **var** $i : nat, n : nat$
2: $n \leftarrow tam(A)$
3: $i \leftarrow 0$ $\quad\triangleright$ línea 2-3: 3
4: **mientras** $i < n \wedge A[i] \neq e$ **hacer** $\quad\triangleright$ ciclo: $(n+1) \cdot 4 + 2 \cdot n$
5: $\quad i \leftarrow i + 1$ $\quad\triangleright$ 2
6: **devolver** $(i < n)$ $\quad\triangleright$ 2

- $f_{mejor}(n)$ y $f_{peor}(n)$: **cantidad de operaciones** realizadas para un arreglo de tamaño n en mejor y peor caso respectivamente.
- $f_{peor}(n) = 3 + (n+1)\ 4 + 2n + 2 = 3 + 4 + 2 + (4+2)\ n$
  $= 9 + 6\ n$

Algorithm $\textsc{AlgoritmoQueHaceAlgo}(A : \texttt{arreglo(nat)})$

```
i ← 0;
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

- ¿Cuándo es mejor caso?
- ¿Cuándo es peor caso?

Algorithm $\textsc{AlgQueHaceAlgo}(A)$

$i \leftarrow 1$; $j \leftarrow 1$
$suma \leftarrow 1$; $count \leftarrow 0$
mientras $i \leq tam(A)$ **hacer**
    si $i \neq A[i]$ **entonces**
        $count \leftarrow count + 1$
    $j \leftarrow 1$
    mientras $j \leq count$ **hacer**
        $k \leftarrow 1$
        mientras $k \leq tam(A)$ **hacer**
            $suma \leftarrow suma + A[k]$
            $k \leftarrow k \cdot 2$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
devolver suma

$A[1]$ es la primera posición del arreglo.

Mejor caso: $A[i] = i$ siempre

$$T_{mejor} = \sum_{i=1}^{n} (\Theta(1)) = \Theta(n)$$

Algorithm ALGQUEHACEALGO(A)

```
i ← 1; j ← 1
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

Algorithm $\textsc{AlgQueHaceAlgo}(A)$

$i \leftarrow 1$; $j \leftarrow 1$
$suma \leftarrow 1$; $count \leftarrow 0$
mientras $i \leq tam(A)$ **hacer**
    si $i \neq A[i]$ **entonces**
        $count \leftarrow count + 1$
    $j \leftarrow 1$
    mientras $j \leq count$ **hacer**
        $k \leftarrow 1$
        mientras $k \leq tam(A)$ **hacer**
            $suma \leftarrow suma + A[k]$
            $k \leftarrow k \cdot 2$
        $j \leftarrow j + 1$
    $i \leftarrow i + 1$
devolver suma

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

$$T_{peor}(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \log n$$

Algorithm ALGQUEHACEALGO(A)

```
i ← 1; j ← 1
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

$A[1]$ es la primera posición del arreglo.

Peor caso: $A[i] \neq i$ siempre

$$T_{peor}(n) = \sum_{i=1}^{n} \sum_{j=1}^{i} \log n$$

$$= \log n \sum_{i=1}^{n} i$$

$$= \log n \frac{n(n+1)}{2}$$

$$= \frac{\log n}{2}(n^2 + n)$$

$$T_{peor} \in \Theta(n^2 \log n)$$

Algorithm ALGQUEHACEALGO(A)

```
i ← 1; j ← 1
suma ← 1; count ← 0
mientras i ≤ tam(A) hacer
    si i ≠ A[i] entonces
        count ← count + 1
    j ← 1
    mientras j ≤ count hacer
        k ← 1
        mientras k ≤ tam(A) hacer
            suma ← suma + A[k]
            k ← k · 2
        j ← j + 1
    i ← i + 1
devolver suma
```

$A[1]$ es la primera posición del arreglo.