Complejidad de algoritmos

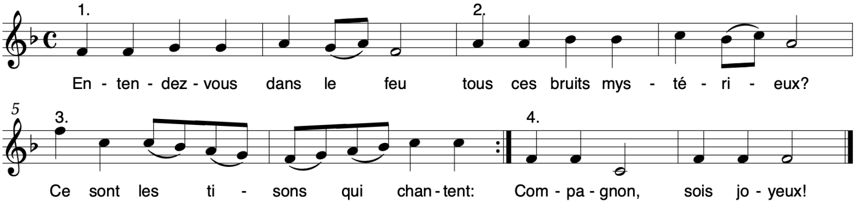
Clase 21

IIC 1253

Prof. Sebastián Bugedo

Outline

Entendez-vous Traditional



Entendez-vous dans le feu
tous ces bruits mystérieux?

Ce sont les tisons qui chantent:
Compagnon, sois joyeux!

Tercer Acto: Aplicaciones

Algoritmos, grafos y números

Playlist Tercer Acto

DiscretiWawos #3

Además sigan en instagram:
@orquesta_tamen

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^+$.

Definición

$$\mathcal{O}(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq c \cdot f(n))\}$$

Diremos que $g \in \mathcal{O}(f)$ es a lo más de orden f o que es $\mathcal{O}(f)$.

Si $g \in \mathcal{O}(f)$, entonces "g **crece más lento o igual** que f"

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^+$.

Definición

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \geq c \cdot f(n))\}$$

Diremos que $g \in \Omega(f)$ es al menos de orden f o que es $\Omega(f)$.

Si $g \in \Omega(f)$, entonces "g **crece más rápido o igual** que f"

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^+$.

Definición

$\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$

Diremos que $g \in \Theta(f)$ es exactamente de orden f o que es $\Theta(f)$.

Si $g \in \Theta(f)$, entonces "g **crece igual** que f"

Ejercicio

Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $\forall n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

Notación asintótica

Ejercicio

Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $\forall n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

$$
\begin{aligned}
& g \in \Theta(f) \\
\Leftrightarrow \quad & g \in \mathcal{O}(f) \wedge g \in \Omega(f) \\
\Leftrightarrow \quad & (\exists d \in \mathbb{R}^+)(\exists n_1 \in \mathbb{N})(\forall n \geq n_1)(g(n) \leq d \cdot f(n) \\
& \wedge (\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)(g(n) \geq c \cdot f(n)) \\
& \text{Tomamos } n_0 = max\{n_1, n_2\} \\
\Leftrightarrow \quad & (\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq d \cdot f(n) \\
& \wedge (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \geq c \cdot f(n)) \\
\Leftrightarrow \quad & (\exists c \in \mathbb{R}^+)(\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c \cdot f(n) \leq g(n) \leq d \cdot f(n)
\end{aligned}
$$

Notación asintótica

Ejercicios

Demuestre que:

1. $f(n) = 60n^2$ es $\Theta(n^2)$.
2. $f(n) = 60n^2 + 5n + 1$ es $\Theta(n^2)$.

¿Qué podemos concluir de estos dos ejemplos?

- Las constantes no influyen.
- En funciones polinomiales, el mayor exponente "manda".

Solución: Apuntes Jorge Pérez, Sección 3.1.2, páginas 102 y 103.

Notación asintótica

Ejercicio

Demuestre que $f(n) = \log_2(n)$ es $\Theta(\log_3(n))$.

¿Qué podemos concluir de este ejemplo?

- Nos podemos independizar de la base del logaritmo.

Solución: Apuntes Jorge Pérez, Sección 3.1.2, página 103.

Notación asintótica

Podemos formalizar las conclusiones anteriores:

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Teorema

Si $f(n) = \log_a(n)$ con $a > 1$, entonces para todo $b > 1$ se cumple que f es $\Theta(\log_b(n))$.

Ejercicio (propuesto ★)

Demuestre los teoremas.

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Es conveniente expresar $f(n)$ como $\sum_{i=0}^{k} a_i n^i$.

Notemos que $\forall x \in \mathbb{R}, x \leq |x|$, por lo que $f(n) \leq \sum_{i=0}^{k} |a_i| n^i$.

Ahora, $\forall n \geq 1$ se cumple que $n^i \geq n^{i-1}$, y luego $f(n) \leq \left(\sum_{i=0}^{k} |a_i| \right) n^k$.

Tomamos entonces $n_0 = 1$ y $c = \sum_{i=0}^{k} |a_i|$, con lo que $f \in \mathcal{O}(n^k)$.

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Para demostrar que $f \in \Omega(n^k)$, debemos encontrar c y n_0 tales que

$$\forall n \geq n_0, c \cdot n^k \leq \sum_{i=0}^{k} a_i n^i \quad (1)$$

Notemos que $\lim_{n \to +\infty} \frac{f(n)}{n^k} = a_k$, y luego asintóticamente tendremos que $c \leq a_k$. Vamos a elegir un c que sea menor que a_k y luego encontraremos el valor de n_0 desde el cual se cumple (1).

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Tomemos $c = \frac{a_k}{2}$:

$$
\begin{aligned}
\frac{a_k}{2} \cdot n^k &\leq \sum_{i=0}^{k} a_i n^i \\
&\leq a_k \cdot n^k + \sum_{i=0}^{k-1} a_i n^i \\
&\leq \frac{a_k}{2} \cdot n^k + \frac{a_k}{2} \cdot n^k + \sum_{i=0}^{k-1} a_i n^i \\
\Rightarrow \frac{a_k}{2} \cdot n^k &\geq - \sum_{i=0}^{k-1} a_i n^i
\end{aligned}
$$

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Podemos relajar la condición:

$$\frac{a_k}{2} \cdot n^k \geq \sum_{i=0}^{k-1} |a_i| n^i \qquad \text{Dividimos por } n^{k-1}$$

$$\frac{a_k}{2} \cdot n \geq \sum_{i=0}^{k-1} |a_i| n^{i-(k-1)} \qquad \text{Como } n^{i-(k-1)} \leq 1, \text{ relajamos de nuevo}$$

$$\frac{a_k}{2} \cdot n \geq \sum_{i=0}^{k-1} |a_i|$$

$$n \geq \frac{2}{a_k} \sum_{i=0}^{k-1} |a_i|$$

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Tomamos entonces $n_0 = \frac{2}{a_k} \sum_{i=0}^{k-1} |a_i|$, con lo que $f \in \Omega(n^k)$, y por lo tanto $f \in \Theta(n^k)$.

Teorema

Si $f(n) = \log_a(n)$ con $a > 1$, entonces para todo $b > 1$ se cumple que f es $\Theta(\log_b(n))$.

Sean $x = \log_a(n)$ e $y = \log_b(n)$. Esto es equivalente a que $a^x = n$ y $b^y = n$, y por lo tanto $a^x = b^y$. Aplicando $\log_a$ a ambos lados, obtenemos que $x = \log_a(b^y)$, y por propiedad de logaritmo se tiene que $x = y \cdot \log_a(b)$. Reemplazando de vuelta x e y, tenemos que $\log_a(n) = \log_b(n) \cdot \log_a(b)$, y por lo tanto para todo $n \geq 1$:

$$\begin{aligned} \log_a(n) &\leq \log_a(b) \cdot \log_b(n) \\ \wedge \log_a(n) &\geq \log_a(b) \cdot \log_b(n) \end{aligned}$$

Tomamos entonces $n_0 = 1$ y $c = \log_a(b)$ y tenemos que

$$\begin{aligned} \forall n \geq n_0 \log_a(n) \leq c \cdot \log_b(n) &\Leftrightarrow \log_a(n) \in \mathcal{O}(\log_b(n)) \\ \forall n \geq n_0 \log_a(n) \geq c \cdot \log_b(n) &\Leftrightarrow \log_a(n) \in \Omega(\log_b(n)) \end{aligned}$$

de donde concluimos que $\log_a(n) \in \Theta(\log_b(n))$. $\square$

Notación asintótica

Las funciones más usadas para los órdenes de notación asintótica tienen nombres típicos:

Notación	Nombre
$\Theta(1)$	Constante
$\Theta(\log n)$	Logarítmico
$\Theta(n)$	Lineal
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^k)$	Polinomial
$\Theta(m^n)$	Exponencial
$\Theta(n!)$	Factorial

con $k \geq 0, m \geq 2$.

Objetivos de la clase

- Obtener la complejidad de algoritmos iterativos
- Deducir ecuaciones de recurrencia para $T(n)$
- Resolver recurrencias con substituciones
- Deducir complejidad sin substituciones
- Conocer el teorema maestro de algoritmos

Outline

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^+$.

Definición

$\mathcal{O}(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq c \cdot f(n))\}$

Diremos que $g \in \mathcal{O}(f)$ es a lo más de orden f o que es $\mathcal{O}(f)$.

Si $g \in \mathcal{O}(f)$, entonces “g **crece más lento o igual** que f”

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^+$.

Definición

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \geq c \cdot f(n))\}$$

Diremos que $g \in \Omega(f)$ es al menos de orden f o que es $\Omega(f)$.

Si $g \in \Omega(f)$, entonces "g **crece más rápido o igual** que f"

Notación asintótica

Sea $f : \mathbb{N} \to \mathbb{R}^{+}$.

Definición

$\Theta(f) = \mathcal{O}(f) \cap \Omega(f)$

Diremos que $g \in \Theta(f)$ es exactamente de orden f o que es $\Theta(f)$.

Si $g \in \Theta(f)$, entonces "g **crece igual** que f"

Ejercicio

Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^{+}$ y $n_0 \in \mathbb{N}$ tales que $\forall n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

Notación asintótica

Ejercicio

Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $\forall n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

$$
\begin{array}{ll}
 & g \in \Theta(f) \\
\Leftrightarrow & g \in \mathcal{O}(f) \wedge g \in \Omega(f) \\
\Leftrightarrow & (\exists d \in \mathbb{R}^+)(\exists n_1 \in \mathbb{N})(\forall n \geq n_1)(g(n) \leq d \cdot f(n) \\
 & \wedge(\exists c \in \mathbb{R}^+)(\exists n_2 \in \mathbb{N})(\forall n \geq n_2)(g(n) \geq c \cdot f(n)) \\
 & \text{Tomamos } n_0 = max\{n_1, n_2\} \\
\Leftrightarrow & (\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \leq d \cdot f(n) \\
 & \wedge(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(g(n) \geq c \cdot f(n)) \\
\Leftrightarrow & (\exists c \in \mathbb{R}^+)(\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(c \cdot f(n) \leq g(n) \leq d \cdot f(n)
\end{array}
$$

Notación asintótica

Ejercicios

Demuestre que:

1. $f(n) = 60n^2$ es $\Theta(n^2)$.
2. $f(n) = 60n^2 + 5n + 1$ es $\Theta(n^2)$.

¿Qué podemos concluir de estos dos ejemplos?

- Las constantes no influyen.
- En funciones polinomiales, el mayor exponente "manda".

Solución: Apuntes Jorge Pérez, Sección 3.1.2, páginas 102 y 103.

Notación asintótica

Ejercicio

Demuestre que $f(n) = \log_2(n)$ es $\Theta(\log_3(n))$.

¿Qué podemos concluir de este ejemplo?

- Nos podemos independizar de la base del logaritmo.

Solución: Apuntes Jorge Pérez, Sección 3.1.2, página 103.

Notación asintótica

Podemos formalizar las conclusiones anteriores:

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Teorema

Si $f(n) = \log_a(n)$ con $a > 1$, entonces para todo $b > 1$ se cumple que f es $\Theta(\log_b(n))$.

Ejercicio (propuesto ★)

Demuestre los teoremas.

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Es conveniente expresar $f(n)$ como $\sum_{i=0}^{k} a_i n^i$.

Notemos que $\forall x \in \mathbb{R}, x \leq |x|$, por lo que $f(n) \leq \sum_{i=0}^{k} |a_i| n^i$.

Ahora, $\forall n \geq 1$ se cumple que $n^i \geq n^{i-1}$, y luego $f(n) \leq \left(\sum_{i=0}^{k} |a_i| \right) n^k$.

Tomamos entonces $n_0 = 1$ y $c = \sum_{i=0}^{k} |a_i|$, con lo que $f \in \mathcal{O}(n^k)$.

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Para demostrar que $f \in \Omega(n^k)$, debemos encontrar c y n_0 tales que

$$\forall n \geq n_0, c \cdot n^k \leq \sum_{i=0}^{k} a_i n^i \quad (1)$$

Notemos que $\lim_{n \to +\infty} \frac{f(n)}{n^k} = a_k$, y luego asintóticamente tendremos que $c \leq a_k$. Vamos a elegir un c que sea menor que a_k y luego encontraremos el valor de n_0 desde el cual se cumple (1).

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Tomemos $c = \frac{a_k}{2}$:

$$\begin{aligned}
\frac{a_k}{2} \cdot n^k &\leq \sum_{i=0}^{k} a_i n^i \\
&\leq a_k \cdot n^k + \sum_{i=0}^{k-1} a_i n^i \\
&\leq \frac{a_k}{2} \cdot n^k + \frac{a_k}{2} \cdot n^k + \sum_{i=0}^{k-1} a_i n^i \\
\Rightarrow \frac{a_k}{2} \cdot n^k &\geq - \sum_{i=0}^{k-1} a_i n^i
\end{aligned}$$

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Podemos relajar la condición:

$$\frac{a_k}{2} \cdot n^k \geq \sum_{i=0}^{k-1} |a_i| n^i \qquad \text{Dividimos por } n^{k-1}$$

$$\frac{a_k}{2} \cdot n \geq \sum_{i=0}^{k-1} |a_i| n^{i-(k-1)} \qquad \text{Como } n^{i-(k-1)} \leq 1, \text{ relajamos de nuevo}$$

$$\frac{a_k}{2} \cdot n \geq \sum_{i=0}^{k-1} |a_i|$$

$$n \geq \frac{2}{a_k} \sum_{i=0}^{k-1} |a_i|$$

Notación asintótica

Teorema

Si $f(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + \ldots + a_2 \cdot n^2 + a_1 \cdot n + a_0$, con $a_i \in \mathbb{R}$ y $a_k > 0$, entonces f es $\Theta(n^k)$.

Tomamos entonces $n_0 = \frac{2}{a_k} \sum_{i=0}^{k-1} |a_i|$, con lo que $f \in \Omega(n^k)$, y por lo tanto $f \in \Theta(n^k)$.

Notación asintótica

Teorema

Si $f(n) = \log_a(n)$ con $a > 1$, entonces para todo $b > 1$ se cumple que f es $\Theta(\log_b(n))$.

Sean $x = \log_a(n)$ e $y = \log_b(n)$. Esto es equivalente a que $a^x = n$ y $b^y = n$, y por lo tanto $a^x = b^y$. Aplicando $\log_a$ a ambos lados, obtenemos que $x = \log_a(b^y)$, y por propiedad de logaritmo se tiene que $x = y \cdot \log_a(b)$. Reemplazando de vuelta x e y, tenemos que $\log_a(n) = \log_b(n) \cdot \log_a(b)$, y por lo tanto para todo $n \geq 1$:

$$\begin{aligned} &\log_a(n) \leq \log_a(b) \cdot \log_b(n) \\ \wedge\ &\log_a(n) \geq \log_a(b) \cdot \log_b(n) \end{aligned}$$

Tomamos entonces $n_0 = 1$ y $c = \log_a(b)$ y tenemos que

$$\forall n \geq n_0 \ \log_a(n) \leq c \cdot \log_b(n) \Leftrightarrow \log_a(n) \in \mathcal{O}(\log_b(n))$$
$$\forall n \geq n_0 \ \log_a(n) \geq c \cdot \log_b(n) \Leftrightarrow \log_a(n) \in \Omega(\log_b(n))$$

de donde concluimos que $\log_a(n) \in \Theta(\log_b(n))$. $\square$

Notación asintótica

Las funciones más usadas para los órdenes de notación asintótica tienen nombres típicos:

Notación	Nombre
$\Theta(1)$	Constante
$\Theta(\log n)$	Logarítmico
$\Theta(n)$	Lineal
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	Cuadrático
$\Theta(n^3)$	Cúbico
$\Theta(n^k)$	Polinomial
$\Theta(m^n)$	Exponencial
$\Theta(n!)$	Factorial

con $k \geq 0, m \geq 2$.

Outline

Volviendo a complejidad...

Queremos encontrar una función $T(n)$ que modele el tiempo de ejecución de un algoritmo.

- Donde n es el tamaño del input.
- No queremos valores exactos de T para cada n, sino que una notación asintótica para ella.
- Para encontrar T, contamos las instrucciones ejecutadas por el algoritmo.
- A veces contaremos cierto tipo de instrucciones que son relevantes para un algoritmo particular.

Contando instrucciones

Ejercicio

Considere el siguiente trozo de código:

```
1 x ← 0
2 for i = 1 to n do
3     for j = 1 to i do
4         x ← x + 1
```

Encuentre una notación asintótica para la cantidad de veces que se ejecuta la instrucción 4 en función de n.

Solución: Apuntes Jorge Pérez, Sección 3.1.3, páginas 104 y 105.

Contando instrucciones

Ejercicio

Considere el siguiente trozo de código:

1 $x \leftarrow 0$
2 $j \leftarrow n$
3 **while** $j \geq 1$ **do**
4   **for** $i = 1$ **to** j **do**
5         $x \leftarrow x + 1$
6     $j \leftarrow \lfloor \frac{j}{2} \rfloor$

Encuentre una notación asintótica para la cantidad de veces que se ejecuta la instrucción 5 en función de n.

Solución: Apuntes Jorge Pérez, Sección 3.1.3, página 105.

Algoritmos iterativos

Consideremos el siguiente algoritmo de búsqueda en arreglos:

input : rreglo de enteros $A = [a_0, \ldots, a_{n-1}]$, un natural $n > 0$ correspondiente al largo del arreglo y un entero k

output: índice de k en A, -1 si no está.

`Búsqueda`(A, n, k)**:**

> **for** $i = 0$ **to** $n - 1$ **do**
>
>   **if** $a_i = k$ **then**
>
>   **return** i
>
> **return** -1

Algoritmos iterativos

¿Qué instrucción(es) contamos?

- Deben ser representativas de lo que hace el problema.
- En este caso, por ejemplo 3 y 4 no lo son (¿por qué?).
- La instrucción 2 si lo sería, y más específicamente la comparación.
  - Las comparaciones están entre las instrucciones que se cuentan típicamente, sobre todo en búsqueda y ordenación.

¿Respecto a qué parámetro buscamos la notación asintótica?

- En el ejemplo, es natural pensar en el tamaño del arreglo n.

En conclusión: queremos encontrar una notación asintótica (ojalá Θ) para la cantidad de veces que se ejecuta la comparación de la línea 2 en función de n. Llamaremos a esta cantidad $T(n)$.

Algoritmos iterativos

Ahora, ¿$T(n)$ depende sólo de n?

- El contenido del arreglo influye en la ejecución del algoritmo.
- Estimaremos entonces el tiempo para el **peor caso** (cuando el input hace que el algoritmo se demore la mayor cantidad de tiempo posible) y el **mejor caso** (lo contrario) para un tamaño de input n.

En nuestro ejemplo:

- **Mejor caso:** $a_0 = k$. Aquí la línea 2 se ejecuta una vez, y luego $T(n)$ es $\Theta(1)$.
- **Peor caso:** k no está en A. La línea 2 se ejecutará tantas veces como elementos en A, y entonces $T(n)$ es $\Theta(n)$.
- Diremos entonces que el algoritmo Búsqueda es de **complejidad** $\Theta(n)$ o lineal en el peor caso, y $\Theta(1)$ o constante en el mejor caso.

Algoritmos iterativos

Ejercicio

Determine la complejidad en el mejor y peor caso:

input : arreglo $A = [a_0, \ldots, a_{n-1}]$ y su largo $n > 0$
output: arreglo está ordenado al terminar el algoritmo.

```
InsertionSort(A, n):
    for i = 1 to n − 1 do
        j ← i
        while a_{j−1} > a_j ∧ j > 0 do
            t ← a_{j−1}
            a_{j−1} ← a_j
            a_j ← t
            j ← j − 1
```

Solución: Apuntes Jorge Pérez, Sección 3.1.3, página 106.

Algoritmos iterativos

En general, nos conformaremos con encontrar la complejidad del peor caso.

- Es la que más interesa, al decirnos qué tan mal se puede comportar un algoritmo en la práctica.

Además, a veces puede ser difícil encontrar una notación Θ.

- ¿Con qué nos basta?
- Es suficiente con una buena estimación O, tanto para el mejor y el peor caso.
- Nos da una cota superior para el tiempo de ejecución del algoritmo.

Outline

Algoritmos recursivos

En el caso de los algoritmos recursivos, el principio es el mismo: contar instrucciones.

- Buscamos alguna(s) instrucción(es) representativa.
- Contamos cuántas veces se ejecuta en cada ejecución del algoritmo.
- ¿Cuál es la diferencia?

Ahora tenemos que considerar llamados recursivos

Algoritmos recursivos: un ejemplo

input : Arreglo ordenado $A[0, \dots, n-1]$, elemento x, índices i, f
output: Índice $m \in \{0, \dots, n-1\}$ o -1

`BinarySearch`(A, x, i, f)**:**

    if $i > f$ **then**
        return -1
    else if $i = f$ **then**
        if $A[i] = a$ **then**
            return i
        else
            return -1
    else
        $m \leftarrow \lfloor (i+f)/2 \rfloor$
        if $A[m] < x$ **then**
            return `BinarySearch`$(A, x, m+1, f)$
        else if $A[m] > x$ **then**
            return `BinarySearch`$(A, x, i, m-1)$
        else
            if $A[m] = x$ **then return** m

Algoritmos recursivos: un ejemplo

- ¿Qué operaciones contamos?
- ¿Cuál es el peor caso?

Ejercicio

Encuentre una función $T(n)$ para la cantidad de comparaciones que realiza el algoritmo `BinarySearch` en el peor caso, en función del tamaño del arreglo.

Respuesta:

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Esta es una **ecuación de recurrencia**.

¿Cómo obtenemos una fórmula explícita?

Algoritmos recursivos: un ejemplo

Ejercicio

Encuentre una función $T(n)$ para la cantidad de comparaciones que realiza el algoritmo `BinarySearch` en el peor caso, en función del tamaño del arreglo.

Contaremos las comparaciones. Dividiremos el análisis del peor caso:

- Si el arreglo tiene largo 1, entramos en la instrucción 3 y luego hay una comparación $\Rightarrow T(n) = 3$, con $n = 1$.
- Si el arreglo tiene largo mayor a 1, el peor caso es entrar en el else de 8 y luego en la segunda llamada recursiva. En tal caso, se hacen las comparaciones de las líneas $1, 3, 10, 12$ a lo que sumamos las comparaciones que haga la llamada recursiva, que serán $T(\lfloor \frac{n}{2} \rfloor)$.

Entonces, nuestra función $T(n)$ será:

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Algoritmos recursivos: ecuaciones de recurrencia

Necesitamos *resolver* esta ecuación de recurrencia.

- Es decir, encontrar una expresión que no dependa de T, sólo de n.
- Técnica básica: sustitución de variables.

¿Cuál sustitución para n nos serviría en el caso anterior?

$$T(n) = \begin{cases} 3 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + 4 & n > 1 \end{cases}$$

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

Respuesta: $T(n) = 4 \cdot \log_2(n) + 3$, con n potencia de 2.

Algoritmos recursivos: ecuaciones de recurrencia

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

$$T(2^k) = \begin{cases} 3 & k = 0 \\ T(2^{k-1}) + 4 & k > 0 \end{cases}$$

Expandiendo el caso recursivo:

$$\begin{aligned} T(2^k) &= T(2^{k-1}) + 4 \\ &= (T(2^{k-2}) + 4) + 4 \\ &= T(2^{k-2}) + 8 \\ &= (T(2^{k-3}) + 4) + 8 \\ &= T(2^{k-3}) + 12 \\ &\vdots \end{aligned}$$

Algoritmos recursivos: ecuaciones de recurrencia

Ejercicio

Resuelva la ecuación ocupando la sustitución $n = 2^k$.

Deducimos una expresión general para $k - i \geq 0$:

$$T(2^k) = T(2^{k-i}) + 4i$$

Tomamos $i = k$:

$$T(2^k) = T(1) + 4k = 3 + 4k$$

Como $k = \log_2(n)$:

$$T(n) = 4 \cdot \log_2(n) + 3, \text{ con } n \text{ potencia de } 2$$

Problema: esto solo es válido cuando $n = 2^k$

Notación asintótica condicional

Sea $P \subseteq \mathbb{N}$.

Definición

$$O(f \mid P) = \{g : \mathbb{N} \to \mathbb{R}^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)$$
$$(n \in P \to g(n) \leq c \cdot f(n))\}$$

Las notaciones $\Omega(f \mid P)$ y $\Theta(f \mid P)$ se definen análogamente.

Estamos restringiendo a un tipo de n particular

Volviendo al ejemplo...

Tenemos que $T(n) = 4 \cdot \log_2(n) + 3$, con n potencia de 2. ¿Qué podemos decir sobre la complejidad de T?

Sea $POTENCIA_2 = \{2^i \mid i \in \mathbb{N}\}$. Entonces:

$$T \in \Theta(\log_2(n) \mid POTENCIA_2)$$

Pero queremos concluir que $T \in \Theta(\log_2(n))$...

Usaremos inducción

Generalización de soluciones usando inducción

Para el ejemplo anterior:

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

Algunas observaciones:

- Demostraremos que $(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c \cdot \log_2(n))$.
- Primero, debemos estimar n_0 y c (expandiendo T por ejemplo).
- ¿Cuál principio de inducción usamos?

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

Veamos los primeros valores de $T(n)$ para estimar c y n_0:

$$
\begin{aligned}
T(1) &= 3 \\
T(2) &= T(1) + 4 = 7 \\
T(3) &= T(1) + 4 = 7 \\
T(4) &= T(2) + 4 = 11
\end{aligned}
$$

Podríamos tomar $c = 7$ y $n_0 = 2$, pues con $n = 1$:

$$T(1) = 3 \not\leq 7 \cdot \log_2(1) = 0$$

y con $n = 2$

$$T(2) = 7 \leq 7 \cdot \log_2(2) = 7$$

La intuición nos dice $n_0 = 2$ y $c = 7$... lo demostraremos

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

PD: $\forall n \geq 2,\ T(n) \leq 7 \cdot \log_2(n)$. Por inducción fuerte:

BI: Además de $n = 2$, debemos mostrar la base para $n = 3$, puesto que depende de $T(1)$ que no está incluido en el resultado que estamos mostrando.

$$T(2) = 7 = 7 \cdot \log_2(2)$$
$$T(3) = 7 < 7 \cdot \log_2(3) \text{ pues el logaritmo es creciente}$$

HI: Supongamos que con $n \geq 4$, $\forall k \in \{2, \cdots, n-1\}$ se cumple que $T(k) \leq 7 \cdot \log_2(k)$.

Generalización de soluciones usando inducción

Ejercicio

Demuestre que si $T \in O(\log_2(n) \mid POTENCIA_2)$, entonces $T \in O(\log n)$.

__**TI:**__ Como $n \geq 4$:

$$
\begin{aligned}
T(n) &= T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 && \text{/ HI} \\
&\leq 7 \cdot \log_2\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 4 && \text{/ log es creciente, sacamos el piso} \\
&\leq 7 \cdot \log_2\left(\frac{n}{2}\right) + 4 && \text{/ log de división} \\
&= 7(\log_2(n) - \log_2(2)) + 4 \\
&= 7 \cdot \log_2(n) - 7 + 4 \\
&= 7 \cdot \log_2(n) - 3 \\
&< 7 \cdot \log_2(n)_{\square}
\end{aligned}
$$

Outline

Objetivos de la clase

- Obtener la complejidad de algoritmos iterativos
- Deducir ecuaciones de recurrencia para $T(n)$
- Resolver recurrencias con substituciones
- Deducir complejidad sin substituciones
- Conocer el teorema maestro de algoritmos