Algoritmos y notación $\mathcal{O}$

Clase 16

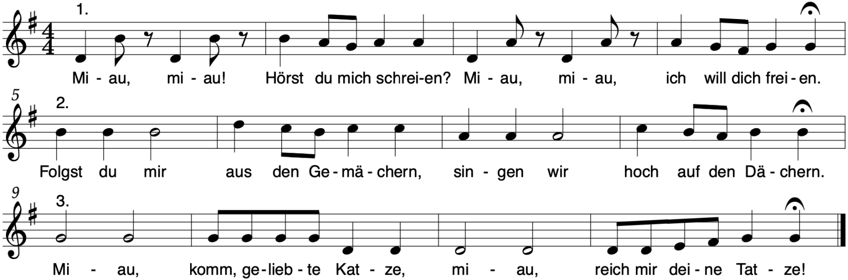
IIC 1253

Prof. Sebastián Bugedo

Outline

Miau

aus Frankreich



Miau, miau, hörst du mich schreien?
Miau, miau, ich will dich freien.

Folgst du mir aus den Gemächern,
singen wir hoch auf den Dächern.

Miau, komm, geliebte Katze,
miau, reich mir deine Tatze!

Tercer Acto: Aplicaciones

Algoritmos, grafos y números

Playlist Tercer Acto

DiscretiWawos #3

Además sigan en instagram:
@orquesta_tamen

Hacia la noción de algoritmo

Necesitamos formalizar la noción de **algoritmo**

Nos interesa la idea de **computación efectiva**

- En el sentido de que *efectivamente puede realizarse*

¿Podemos definir formalmente esta noción?

Intentos de formalización: S. XX

Funciones parcialmente recursivas
por K. Gödel, J. Herbrand, S. Kleene.

λ-calculus
por Alonzo Church.

Sistemas de Post
por Emil Post.

Máquinas de Turing
por Alan Turing.

...

Todos estos métodos son equivalentes

¿Qué es un algoritmo?

Estos métodos capturan la noción intuitiva:

- Algoritmos como secuencias de pasos
- Con precondiciones
- Condiciones de término

Esencialmente, un **algoritmo** es un conjunto de pasos que resuelven un **problema**

Para este curso nos basta con esta intuición

Algoritmos

Diremos entonces que un **algoritmo** es un método para convertir un **INPUT** válido en un **OUTPUT**. A estos métodos les exigiremos ciertas propiedades:

- Precisión: cada instrucción debe ser planteada de forma precisa y no ambigua.
- Determinismo: cada instrucción tiene un único comportamiento que depende sólo del input.
- Finitud: el algoritmo está compuesto por un conjunto finito de instrucciones.

Algoritmos

El análisis de algoritmos es una disciplina de la Ciencia de la Computación que tiene dos objetivos:

- Estudiar cuándo y por qué los algoritmos son **correctos** (es decir, hacen lo que dicen que hacen).
- Estimar la cantidad de **recursos** computacionales que un algoritmo necesita para su ejecución.

De esta manera, podemos, por ejemplo:

- Entender bien los algoritmos, para luego reutilizarlos total o parcialmente.
- Determinar qué mejorar de un algoritmo para que sea más eficiente.

Algoritmos

Usaremos pseudo-código para escribir algoritmos.

- Instrucciones usuales como **if**, **while**, **return**...
- Notaciones cómodas para arreglos, conjuntos, propiedades lógicas, etc.

Consideraremos que los algoritmos tienen:

- **Precondiciones:** representan el input del programa.
- **Postcondiciones:** representan el output del programa, lo que hace el algoritmo con el input.

Objetivos de la clase

- Comprender concepto de algoritmo
- Identificar las componentes del análisis de algoritmos
- Comprender notación asintótica
- Demostrar cotas mediante notación asintótica

Outline

Corrección de algoritmos

Queremos determinar cuándo un algoritmo es correcto; es decir, hace lo que dice que hace.

Definición

Un algoritmo es **correcto** si para todo INPUT válido, el algoritmo se detiene y produce un OUTPUT correcto.

Entonces, ¿cuándo es incorrecto?

Definición

Un algoritmo es **incorrecto** si existe un INPUT válido para el cual el algoritmo no se detiene o produce un OUTPUT incorrecto.

Algoritmos iterativos

Debemos demostrar dos cosas:

- **Corrección parcial:** si el algoritmo se detiene, se cumplen las postcondiciones.
- **Terminación:** el algoritmo se detiene.

Nos preocupamos sólo de los *loops* de los algoritmos (¿por qué?).

Estos loops tienen una condición G que determina si se siguen ejecutando:

while(G)

...

end

Algoritmos iterativos

Para demostrar corrección parcial, buscamos un **invariante** $\mathcal{I}(k)$ para los loops:

- Una propiedad $\mathcal{I}$ que sea verdadera en cada paso k de la iteración.
- Debe relacionar a las variables presentes en el algoritmo.
- Al finalizar la ejecución, debe asegurar que las postcondiciones se cumplan.

Algoritmos iterativos

Una vez que encontramos un invariante, demostramos la corrección del loop inductivamente:

- **Base:** las precondiciones deben implicar que $\mathcal{I}(0)$ es verdadero.
- **Inducción:** para todo natural $k > 0$, si G e $\mathcal{I}(k)$ son verdaderos antes de la iteración, entonces $\mathcal{I}(k+1)$ es verdadero después de la iteración.
- **Corrección:** inmediatamente después de terminado el loop (i.e. cuando G es falso), si $k = N$ e $\mathcal{I}(N)$ es verdadero, entonces la postcondiciones se cumplen.

Y para demostrar terminación, debemos mostrar que existe un k para el cual G es falso.

Algoritmos iterativos

Ejercicio

Escriba un algoritmo que multiplique dos números naturales (sin usar la multiplicación):

- **Pre:** $n, m \in \mathbb{N}$.
- **Post:** $p = n \cdot m$.

Demuestre que su algoritmo es correcto.

Solución: Apuntes Jorge Pérez, Sección 3.1.1, Teorema 3.1.1, páginas 97 a 99.

Algoritmos recursivos

En el caso de los algoritmos recursivos, no necesitamos dividir la demostración en corrección parcial y terminación (¿por qué?).

- Basta demostrar por inducción la propiedad (corrección) deseada.
- En general, la inducción se realiza sobre el tamaño del input.

Ejercicio

Escriba un algoritmo recursivo que encuentre el máximo elemento de un arreglo:

- **Pre:** un arreglo $A = [a_0, a_1, \ldots, a_{n-1}]$, y un natural n (largo del arreglo).
- **Post:** $m = \max(A)$.

Demuestre que el algoritmo es correcto.

Solución: Apuntes Jorge Pérez, Sección 3.1.1, página 101.

Complejidad de algoritmos

Ya vimos cómo determinar cuando un algoritmo era correcto.

- Esto no nos asegura que el algoritmo sea útil en la práctica.
- Necesitamos estimar su tiempo de ejecución.
  - En función del tamaño del input.
  - Independiente de: lenguaje, compilador, hardware...

Lo que nos interesa entonces no es el tiempo *exacto* de ejecución de un algoritmo, sino que su comportamiento a medida que crece el input.

Introduciremos notación que nos permitirá hablar de esto.

Complejidad de algoritmos

Vamos a ocupar funciones de dominio natural ($\mathbb{N}$) y recorrido real positivo ($\mathbb{R}^+$).

- El dominio será el tamaño del input de un algoritmo.
- El recorrido será el tiempo necesario para ejecutar el algoritmo.

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 $\forall n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$.

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

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:

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

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

Objetivos de la clase

- □ Comprender concepto de algoritmo
- □ Identificar las componentes del análisis de algoritmos
- □ Comprender notación asintótica
- □ Demostrar cotas mediante notación asintótica