Pontificia Universidad Católica de Chile

Departamento de Ciencia de la Computación

Apunte IIC2283

Diseño y Análisis de Algoritmos

Autores
Sebastián Besoain
Pedro Salinas
Ignacio Gutiérrez

En base a clases de
Prof. Diego Arroyuelo
Prof. Juan P. Castillo

Template
Cristóbal Rojas

23 de diciembre de 2025

Índice

1. Introducción

2. Análisis de la eficiencia de un algoritmo

Para esta sección es bueno recordar la definición formal de la notación asintótica:

Definición 2.1

Sea $f : \mathbb{N} \to \mathbb{R}^+$ se define la notación asintótica como los siguientes conjuntos de funciones:

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

No será de mucha utilidad en nuestro caso, pero se puede saber el rango del valor de los siguientes límites:

Teorema 2.1

Sea $f : \mathbb{N} \to \mathbb{R}^+$ *y* $g : \mathbb{N} \to \mathbb{R}^+$, *luego*

- *Si* $f(n) \in \mathcal{O}(g(n))$ *entonces* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in [0, \infty)$
- *Si* $f(n) \in \Omega(g(n))$ *entonces* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty]$
- *Si* $f(n) \in \Theta(g(n))$ *entonces* $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in (0, \infty)$

2.1. Ecuaciones de recurrencia

Una **recurrencia** es una ecuación o desigualdad que describe una función en términos de su valor sobre inputs más pequeños.

Ejemplo 2.1

El peor caso de MERGE-SORT está dado por la ecuación de recurrencia

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{si } n > 1 \end{cases}$$

2.1.1. Sustitución de variables

Existen varios métodos para resolver estas recurrencias, uno de ellos es la **sustitución de variables** que consiste en dos pasos:

1. Adivinar la forma de la solución
2. Usar inducción para encontrar las constantes y mostrar que la solución funciona

Ejemplo 2.2

Considere la recurrencia:

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Adivinamos primero que la solución es $T(n) \in \mathcal{O}(n \log n)$, el método de sustitución requiere demostrar que $T(n) \leq cn \log n$ para algún $c > 0$. Empezamos asumiendo que se cumple para todo $m < n$ con m positivo, en particular para $m = \lfloor n/2 \rfloor$, luego tenemos que $T(\lfloor n/2 \rfloor) \leq c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor)$. Sustituyendo

en la ecuación tenemos que

$$\begin{aligned} T(n) &\leq 2\big(c\lfloor n/2\rfloor \lg(\lfloor n/2\rfloor)\big) + n \\ &\leq cn\lg(n/2) + n && \text{(Notar que } c \text{ sigue siendo una constante)} \\ &= cn\lg n - cn\lg 2 + n \\ &= cn\lg n - cn + n \\ &\leq cn\lg n. \end{aligned}$$

Solo se necesita que $c \geq 1$.

La inducción matemática ahora requiere que mostremos nuestra solución para las condiciones límite. Note que para $n = 1$, entonces:

$$\begin{aligned} T(1) &\leq c1\log 1 \\ &= 0 \end{aligned}$$

Lo que es falso ya que $T(1) = 1$. Aquí gracias a la definición formal podemos fijar un n_0 que funcione en este caso podemos elegir $n_0 = 2$. Al escoger este n_0, debemos demostrar como casos base todos aquellos casos donde la recursión de $T(n)$ resulte en un caso menor a $n_0 = 2$ aquí serían $T(2)$ y $T(3)$ ya que ambos dependen de $T(1)$ directamente (dado la ecuación). Queda como ejercicio para el lector definir un c y demostrar por inducción la cota.

Ejercicio 2.1

Considere nuevamente la recurrencia de MERGE-SORT:

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ T(\lceil n/2\rceil) + T(\lfloor n/2\rfloor) + \Theta(n) & \text{si } n \geq 2 \end{cases}$$

Encuentre el orden $f : \mathbb{N} \to \mathbb{R}^+$ de $\mathcal{O}$ de $T(n)$ y demuestre formalmente que $T(n) \in \mathcal{O}(f(n))$.

Solución: Escogemos $f(n) = n\log n$, en este caso particular nos conviene el reemplazo $T(n) \leq cn\log(n-1)$ debido a las funciones techo en la ecuación, luego

$$\begin{aligned} T(n) &\leq c\lceil n/2 - 1\rceil \log\lceil n/2 - 1\rceil + c\lfloor n/2 - 1\rfloor \log\lfloor n/2 - 1\rfloor + n \\ &\leq c\lceil n/2 - 1\rceil (2\log\lceil n/2 - 1\rceil) \\ &\leq cn\left(\log\frac{(n+1)}{2} - 1\right) \\ &\leq cn\log(n-1) \end{aligned}$$

Ahora falta encontrar c, n_0 y demostrar por inducción que se cumple para todo $n \geq n_0$. En este caso elegiremos $c = 3$ y $n_0 = 3$, ya que para $T(1)$ y $T(2)$ no se cumple. Luego nuestros casos bases son $\{3, 4, 5\}$ para los cuales se cumple (lo puede comprobar), luego para el paso inductivo con $n \geq 6$ tenemos que:

$$\begin{aligned} T(n) &\leq 3\lceil n/2 - 1\rceil \log\lceil n/2 - 1\rceil + \lfloor n/2 - 1\rfloor \log\lfloor n/2 - 1\rfloor + n \\ &\leq 3n(\log\lceil n/2 - 1\rceil) \\ &\leq 3n\left(\log\frac{(n+1)}{2} - 1\right) \\ &\leq 3n\log(n-1) \quad \triangle \end{aligned}$$

Hay casos donde puede resultar problemático encontrar la función solo con definición 2.1:

Ejemplo 2.3

Considere la recurrencia:

$$T(n) = \begin{cases} 0 & \text{si } n = 0 \\ n^2 + n \cdot T(n-1) & \text{si } n > 0 \end{cases}$$

Intentemos inmediatamente usar $f(n) = n!$ y resolver por inducción, suponemos que $T(n) \leq c \cdot n!$ luego tenemos que

$$\begin{aligned} T(n+1) &= (n+1)^2 + (n+1) \cdot T(n) \\ &\leq (n+1)^2 + (n+1) \cdot (c \cdot n!) \\ &= (n+1)^2 + c \cdot (n+1)! \end{aligned}$$

Note que es imposible que $c \cdot (n+1)! \geq (n+1)^2 + c \cdot (n+1)!$, $\forall c \in \mathbb{R}^+$.

Aquí para poder progresar, podemos usar una definición más fuerte

$$(\exists c \in \mathbb{R}^+)(\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c \cdot n! - dn)$$

Esto debido a que $c \cdot n! - dn \leq c \cdot n!$, ahora completando la inducción, tendríamos que

$$\begin{aligned} T(n+1) &= (n+1)^2 + (n+1) \cdot T(n) \\ &\leq (n+1)^2 + (n+1) \cdot (c \cdot n! - dn) \\ &= c \cdot (n+1)! + (n+1)^2 - (n+1)dn \\ &= c \cdot (n+1)! + (n+1)((n+1) - dn) \end{aligned}$$

Ahora necesitamos restringir d

$$\begin{aligned} c \cdot (n+1)! + (n+1)((n+1) - dn) &\leq c(n+1)! - d(n+1) \\ (n+1) - dn &\leq -d \\ n + 1 &\leq d(n-1) \\ \frac{n+1}{n-1} &\leq d \end{aligned}$$

Considerando $n \geq 2$, entonces $d \geq 3$. Como ejercicio para el lector encuentre el valor de n_0 y c para después mostrar el caso base y concluir.

2.1.2. Algunas sutilezas de la sustitución de variables

Es fácil equivocarse al usar notación asintótica, considere por ejemplo la recurrencia del ejemplo 2.2

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Podemos "demostrar" falsamente que $T(n) \in \mathcal{O}(n)$ suponiendo que $T(n) \leq cn$ y luego argumentando

$$\begin{aligned} T(n) &\leq 2(c\lceil n/2 \rceil) + n \\ &\leq cn + n \\ &\in \mathcal{O}(n) \end{aligned}$$

La conclusión de que $T(n) \in \mathcal{O}(n)$ es falsa, para demostrarlo de manera real se debe llegar a la forma exacta de hipótesis de inducción, es decir $T(n) \leq cn$.

Ejemplo 2.4: Cambio de variable

Otro problema que puede surgir al usar este método es que nos den una recurrencia extraña y difícil de manipular, por ejemplo la recurrencia

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \log n$$

Aquí podemos simplificar la recurrencia utilzando un **cambio de variables** similar a como se hace en cálculo, aquí escogemos $m = \log n$ y tenemos que

$$T(2^m) = 2T(2^{(m/2)}) + m$$

Ahora podemos asignar $S(m) = T(2^m)$ y tenemos que

$$S(m) = 2S(m/2) + m$$

Resolviendo esta recurrencia familiar tenemos que $S(m) \in \mathcal{O}(m \log m)$, luego reemplazando devuelta

$$m \log m = \log n \log(\log n)$$

Por lo tanto $T(n) \in \mathcal{O}(\log n \log(\log n))$ (que es una familia de funciones bastante extraña).

2.2. El Teorema Maestro

2.3. Ejercicios del capítulo

3. Análisis de caso promedio

3.1. Ejercicios del capítulo

4. Técnicas para demostrar cotas inferiores

4.1. Ejercicios del capítulo

5. Técnicas fundamentales de diseño de algoritmos

5.1. Ejercicios del capítulo

6. Transformaciones de domino

6.1. Representación de un polinomio

La representación canónica de un polinomio $p(x)$ no nulo es

$$p(x) = \sum_{i=0}^{n-1} a_i x^i$$

Donde $n \geq 1$, $a_{n-1} \neq 0$ y el grado de $p(x)$ es $n - 1$

6.2. Ejercicios del capítulo

7. Algoritmos aleatorizados

7.1. Algoritmos de Monte Carlo

Un algoritmo de **Monte Carlo** es aquel que siempre entrega un resultado, pero hay una probabilidad de que sea incorrecto. Garantizan tiempo de ejecución de peor caso y suelen ser acompañados por un análisis de error. La probabilidad de error puede ser reducida ejecutando el algoritmo múltiples veces de manera independiente.

Ejemplo 7.1

Considere el problema de encontrar un 1 en alguna posición $i \in \{1, ..., n\}$ de un arreglo de bits $B[1...n]$, el cual contiene $n/2$ cantidad de 0's y $n/2$ cantidad de 1's, una solución tipo Monte Carlo sería la siguiente:

Function `Find_1_MC`$(B[1...n], k)$**:**
    $j := 0$
    repeat
        Elegir $i \in \{1, ..., n\}$ uniformemente al azar.
        $j := j + 1$
    until $j = k$ **or** $B[i] = 1$
    return i

- Este algoritmo recibe un parámetro k, que permite limitar el tiempo de ejecución a $\mathcal{O}(k)$
- No siempre encuentra una posición que encuentra un 1, dado que ejecuta a lo más k iteraciones.

Ejercicio 7.1

Encuentre la probabilidad de error del algoritmo, para cualquier
colororange!100!*blackk* $\in \mathbb{N}$

Note que a medida que aumentamos el parámetro k la probabilidad de éxito va aumentando, esta probabilidad está dada por la fórmula

$$\sum_{i=1}^{k} \frac{1}{2^i} = 1 - \left(\frac{1}{2}\right)^k$$

7.2. Ejercicios del capítulo

8. Algoritmos en teoría de números

8.1. Ejercicios del capítulo