

Complejidad Asintótica

Dr. Erik Alex Papa Quiroz

Universidad Nacional Mayor de San Marcos
Perú

Setiembre del 2026

1. Complejidad Computacional

Algoritmos, Estructuras de Datos y Programas

1. Algoritmo: proceso sistemático para calcular un resultado a partir de datos de entrada.
2. Estructura de Datos: la manera de organizar datos y operar sobre ellos.
3. Algoritmos + Estructura de Datos = Programas: es la expresión en lenguaje formal de un algoritmo.

Algoritmos aparecen en:

1. Internet: personas de todo el mundo obtengan con rapidez cantidades grandes de información.
2. Comercio electrónico: mercaderías y servicios sean negociados electrónicamente.
3. Seguridad de información: privacidad en tarjetas de bancos, tarjetas de crédito, DNle.
4. En la industria: asignar recursos escasos de manera eficiente (localización, ruta mas corta, etc).

Eficiencia de un algoritmo

Complejidad de tiempo: Cuanto "tiempo" es necesario para calcular el resultado para una instancia del problema de tamaño n

- a) Peor caso: se considera el más alto tiempo de ejecución del algoritmo.
- b) Caso medio: requiere análisis estadístico de los datos para hallar el tiempo promedio de la ejecución en función a probabilidades.
- c) Mejor caso: si los datos de prueba generan el mejor tiempo de ejecución.

Complejidad Asintótica

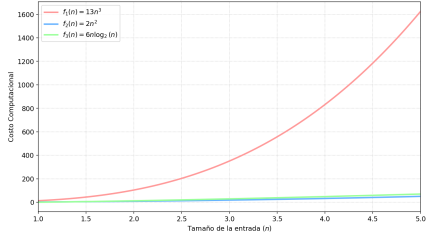
1. Análisis del algoritmo permite estimar una función que depende del tamaño de entrada (número de datos almacenados). Ejemplo:

$$T(n) = 13n^3 + 2n^2 + 6n \log n.$$

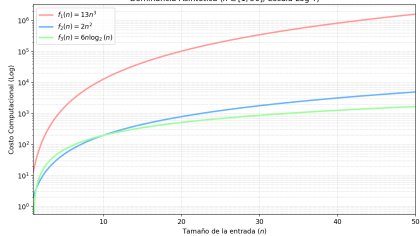
2. La constante que multiplica al término cúbico tiene relativamente la misma importancia que la velocidad del CPU/memoria.

Análisis Comparativo de Funciones de Complejidad Temporal

Comportamiento inicial ($n \in [1, 5]$)



Dominancia Asintótica ($n \in [1, 50]$, Escala Log Y)



Reglas para estimar la función T

1. Asignaciones
2. Comparaciones
3. Operaciones aritméticas
4. Repeticiones

Por simplicidad, en general, si tenemos una operación seguida de una asignación estas dos son consideradas como una unidad de tiempo, por ejemplo:

$$y = x + 2$$

Ejemplo

Cálculo de la función:

$$f(n) = \begin{cases} n^n, & \forall n \in \mathbb{N} \\ 0, & \text{caso contrario} \end{cases}$$

Algoritmo P

Inicio

Leer n

Si $n \leq 0$, entonces escribir $f(n) = 0$

Caso contrario, hacer

$$x = n$$

$$y = n - 1$$

Mientras $y > 0$, hacer

$$x = x * n$$

$$y = y - 1$$

Escribir $f(n) = x$

Fin

Complejidad del Algoritmo P

En la i -ésima iteración la variable x contendrá el valor de n^i .
Luego, como el "Loop" es ejecutado $n - 1$ veces, tendremos la complejidad de

$$T(n) = 3(n - 1) + 3 = 3n.$$

Definición de Limite Superior

Sea $g : \mathbb{N} \rightarrow \mathbb{R}$ una función dada. Definimos el conjunto

$$O(g(n)) = \{f(n) : \exists c \in \mathbb{R}_{++} \text{ y } n_0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}.$$

Ejemplo

La función

$$T(n) = 3n^3 + 2n^2$$

es de orden $O(n^3)$.

En efecto, tomando $n_0 = 1$ y $c = 5$ se tiene

$$3n^3 + 2n^2 \leq 5n^3.$$

Ejemplo

La función

$$T(n) = n^2$$

es de orden $O(n^3)$.

En efecto, tomando $n_0 = 1$ y $c = 1$, se tiene:

$$n^2 \leq n^3.$$

Ejemplo

¿La función

$$T(n) = 3^n$$

es de orden $O(2^n)$?

Respuesta

No. Supongamos que existan constantes n_0 y c tal que $\forall n \geq n_0$:

$$3^n \leq c2^n.$$

Esto implica

$$\left(\frac{3}{2}\right)^n \leq c.$$

Tomando $n \rightarrow +\infty$, se tiene:

$$+\infty \leq c,$$

lo que es una contradicción.

Ejemplo

¿La función

$$T(n) = n^3$$

es de orden $O(n^2)$?

Respuesta

No. Supongamos que existan constantes n_0 y c tal que $\forall n \geq n_0 :$

$$n^3 \leq cn^2.$$

Esto implica

$$n \leq c.$$

Tomando $n \rightarrow +\infty$, se tiene:

$$+\infty \leq c,$$

lo que es una contradicción.

Definición: Limite inferior

Sea $g : \mathbb{N} \rightarrow \mathbb{R}$ una función dada. Definimos el conjunto

$$\Omega(g(n)) = \{f(n) : \exists c \in \mathbb{R}_{++} \text{ y } n_0 : 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$

Definición: Limite superior e inferior

Sea $g : \mathbb{N} \rightarrow \mathbb{R}$ una función dada. Definimos el conjunto

$$\Theta(g(n)) = \Omega(g(n)) \cap O(g(n)).$$

Si $f(n) \in \Theta(g(n))$, entonces el algoritmo se llama óptimo.

Ejemplo

Sea

$$f(n) = 3n^3 + 2n^2.$$

Ya sabemos que $f(n) \in O(n^3)$. Además, tomando $c = 1$ y $n_0 = 1$, se tiene:

$$f(n) \in \Omega(n^3).$$

Por lo tanto $f(n) \in \Theta(n^3)$.

Proposición

Proposición

Si determinado algoritmo se divide en partes independientes, digamos P_1 y P_2 y siendo $T_1(n)$ y $T_2(n)$ respectivamente de orden $O(f(n))$ y $O(g(n))$, entonces

$$T(n) = T_1(n) + T_2(n)$$

y P será de orden

$$O(\max\{f(n), g(n)\})$$

Ejemplo

$$f(n) = \begin{cases} n^4 & \text{si } n \text{ es par} \\ n^2, & \text{si } n \text{ es impar} \end{cases}$$

$$g(n) = \begin{cases} n^2 & \text{si } n \text{ es par} \\ n^3, & \text{si } n \text{ es impar} \end{cases}$$

En este caso,

$$\max\{f(n), g(n)\} = \begin{cases} n^4 & \text{si } n \text{ es par} \\ n^3, & \text{si } n \text{ es impar} \end{cases}$$

Proposición

Si $T_1(n)$ y $T_2(n)$ son de orden $O(f(n))$ y $O(g(n))$ respectivamente, entonces $T(n) = T_1(n).T_2(n)$, es de orden

$$O(f(n).g(n)).$$

Proposición

Sean $f, g, h : \mathbb{N} \rightarrow \mathbb{R}$, entonces

1. $g(n) \in O(f(n))$ y $f(n) \in O(g(n))$ si y sólo si

$$O(f(n)) = O(g(n)).$$

2. $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$, entonces

$$f(n) \in O(h(n)).$$

Definición

Sea $g : \mathbb{N} \rightarrow \mathbb{R}$ una función dada. Definimos el conjunto

$$o(g(n)) = \{f(n) : \forall c \in \mathbb{R}_{++} \exists n_0 > 0 : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$$

Sea $g : \mathbb{N} \rightarrow \mathbb{R}$ una función dada. Definimos el conjunto

$$\omega(g(n)) = \{f(n) : \forall c \in \mathbb{R}_{++} \exists n_0 > 0 : 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$$

Resultados en terminos de límites

Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}$ funciones positivas tales que el siguiente limite existe:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c,$$

donde c puede ser 0 , finito o $+\infty$.

1) Si $c = 0$ ó finito, entonces

$$f(n) \in O(g(n)).$$

2) Si c es finito ó $c = +\infty$, entonces

$$f(n) \in \Omega(g(n)).$$

En terminos de límites

Sean $f, g : \mathbb{N} \rightarrow \mathbb{R}$ funciones positivas tales que $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = c$, donde c puede ser 0, finito o $+\infty$.

3) Si c es finito, entonces $f(n) \in \Theta(g(n))$.

4) Si $c = 0$, entonces

$$f(n) \in o(g(n)).$$

5) Si $c = +\infty$, entonces

$$f(n) \in \omega(g(n)).$$

Ejercicios

Pruebe que:

$$2n \in o(n^2), \text{ pero } 2n^2 \notin o(n^2).$$

En efecto,

- ▶ como $\lim_{n \rightarrow \infty} \frac{2n}{n^2} = 0$, entonces $2n \in O(n^2)$ y $2n \in o(n^2)$.
- ▶ También $\lim_{n \rightarrow \infty} \frac{2n^2}{n^2} = 2$, entonces $2n^2 \in \Theta(n^2)$.
- ▶ Supongamos que $2n^2 \in o(n^2)$, entonces para todo $c > 0$ existe n_0 :

$$2n^2 \leq c(n^2), \forall n \geq n_0.$$

Tomando $c = 1/2$ llegamos a una contradicción.

Ejercicios

Pruebe que:

$$n^2/2 \in \omega(n), \text{ pero } n^2/2 \notin \omega(n^2).$$

Ejercicios

Sea $f(n) = 13n^3 + 2n^2 + 6n \log n$ y $g(n) = n^3$. Pruebe que

$$f(n) \in O(n^3).$$

Ejercicios

Pruebe que para cualquier número $a, b \in \mathbb{R}$ con $b > 0$, se tiene que

$$(n + a)^b \in \Theta(n^b).$$

Ejercicios

¿Es verdad que

$$2^{n+1} \in O(2^n)?$$

Ejercicios

¿Es verdad que

$$2^{2n} \in O(2^n)?$$

Propiedad Transitiva

a) Si $f(n) \in O(g(n))$ y $g(n) \in O(h(n))$, entonces

$$f(n) \in O(h(n)).$$

b) Si $f(n) \in \Omega(g(n))$ y $g(n) \in \Omega(h(n))$, entonces

$$f(n) \in \Omega(h(n)).$$

c) Si $f(n) \in \Theta(g(n))$ y $g(n) \in \Theta(h(n))$, entonces

$$f(n) \in \Theta(h(n)).$$

Propiedad Transitiva

d) Si $f(n) \in o(g(n))$ y $g(n) \in O(h(n))$, entonces

$$f(n) \in o(h(n)).$$

e) Si $f(n) \in \omega(g(n))$ y $g(n) \in \omega(h(n))$, entonces

$$f(n) \in \omega(h(n)).$$

Reflexividad

$$f(n) \in O(f(n))$$

$$f(n) \in \Omega(f(n))$$

$$f(n) \in \Theta(f(n)).$$

Simetría

$$f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n)).$$

Simetría del transpuesto

$$f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)).$$

$$f(n) \in o(g(n)) \iff g(n) \in \omega(f(n)).$$

Similitud con la relación de orden

$$f(n) \in O(g(n)) \sim a \leq b$$

$$f(n) \in \Omega(g(n)) \sim a \geq b$$

$$f(n) \in \Theta(g(n)) \sim a = b$$

$$f(n) \in o(g(n)) \sim a < b$$

$$f(n) \in \omega(g(n)) \sim a > b$$

Similitud con la relación de orden

Si $a, b \in \mathbb{R}$, entonces es verdadera una y sólo una de las proposiciones siguientes:

$$a < b \quad a = b \quad a > b.$$

analogamente, si $f(n), g(n)$ son funciones, puede suceder que

$$f(n) \in O(g(n)) \quad \text{y} \quad f(n) \in \Omega(g(n)),$$

sean ambas falsas.

Algoritmo de Ordenación Bubble-Sort

Inicio

Leer $\{a_1, a_2, a_3, \dots, a_n, n\}$

Para $i = 1 : n - 1$ hacer

 Para $j = i + 1 : n$ hacer

 Si $a_{j-1} > a_j$, entonces

$temp = a_{j-1};$

$a_{j-1} = a_j$

$a_j = temp;$

 Fin (Para).

Fin (Para).

Escribir $\{a_1, a_2, \dots, a_n\}$.

Hallar la complejidad del algoritmo.

Solución

El "Loop" más interno exige, en el peor caso 5 operaciones siendo ejecutado $(n - i)$ veces, luego la complejidad es $5(n - i)$. Como existe un "loop" más externo que ejecuta este último $n - 1$ veces, tenemos:

$$\begin{aligned} T(n) &= \sum_{i=1}^{n-1} \left(\sum_{j=i+1}^n 5 + 1 \right) = \sum_{i=1}^{n-1} [5(n - i) + 1] \\ &= 5 \sum_{i=1}^{n-1} (n - i) + (n - 1) = 5 \frac{n(n - 1)}{2} + (n - 1) \end{aligned}$$

Entonces la complejidad del algoritmo será de

$$O(n^2).$$

Algoritmo: Evaluación de polinomio por potenciación

Inicio

Leer $\{n, a_n, a_{n-1}, a_{n-2}, \dots, a_0, x\}$

$p = a_0;$

Para $i = 1 : n$ hacer

$$p = p + a_i x^i$$

Fin (Para).

Escribir $\{p\}$.

Hallar la complejidad del algoritmo.

Solución

Tenemos n ejecuciones del "Loop" y cada ejecución $(2 + i)$ operaciones son realizadas: una asignación, una suma y i multiplicaciones. Luego,

$$T(n) = \sum_{i=1}^n (2 + i) = \frac{n(n+1)}{2} + 2n.$$

Así, el algoritmo es de orden

$$O(n^2).$$

Algoritmo: Evaluación de polinomio por producto

Inicio

Leer $\{n, a_n, a_{n-1}, a_{n-2}, \dots, a_0, x\}$

$p = a_0;$

$y = x;$

Para $i = 1 : n - 1$ hacer

$p = p + a_i \cdot y$

$y = y \cdot x$

Fin (Para).

$p = p + a_n \cdot y$

Escribir $\{p\}$.

Hallar la complejidad del algoritmo.

Solución

Por cada ejecución del "Loop" tenemos la instrucción, dos productos y una suma, osea 4 operaciones de tiempo constante. Como este es ejecutado $n - 1$ veces tenemos

$$T(n) = 4(n - 1) + 2 = 4n - 2$$

Así, el algoritmo es de orden

$$O(n).$$

Algoritmo: Método de Horner

Inicio

Leer $\{n, a_n, a_{n-1}, a_{n-2}, \dots, a_0, x\}$

$p = a_n;$

Para $i = n - 1 : 0$, hacer

$p = a_i + x.p;$

Fin (Para).

Escribir $\{p\}$.

Hallar la complejidad del algoritmo.

Solución

En este caso cada ejecución del "Loop" realiza, además de la instrucción básica, una suma y un producto, osea 3 operaciones de consumo constante. Así,

$$T(n) = 3(n) + 1$$

Así, el algoritmo es de orden

$$O(n).$$

Ejemplo

Solución de un sistema lineal:

$$Ax = b,$$

donde A es una matriz cuadrada $n \times n$ y b es un vector de dimensión n .

Algoritmo Eliminación Gaussiana

Dados $a_{ij}, b_i, 1 \leq i, j \leq n$.

Para $k = 1 : n - 1$ hacer

encontrar $i \geq k$ tal que $a_{ik} \neq 0$

Si $a_{ii} = 0$ para todo $i \geq k$ entonces A^{-1} no existe

Cambie la línea k con la línea i

Para $i = k + 1 : n$ hacer

$$m = m_{ik} = \frac{a_{ik}}{a_{kk}}$$

$$b_i = b_i - mb_k$$

Para $j = k + 1 : n$ hacer

$$a_{ij} = a_{ij} - ma_{kj}$$

Fin (Para)

Fin(Para)

Fin(Para)

Complejidad de la eliminación de Gauss

- ▶ Para cada valor j en el tercer bloque del algoritmo son realizadas: una multiplicación y una adición. Así en este lazo son necesarias:

$$\sum_{j=k+1}^n 2 = 2(n - k).$$

- ▶ En el segundo bloque (el bloque en i) además de las operaciones contabilizadas anteriormente, para cada i realizamos una división, una multiplicación y una resta. Así el número de operaciones será:

$$\sum_{i=k+1}^n [3 + 2(n - k)] = [3 + 2(n - k)](n - k).$$

Complejidad de la eliminación de Gauss

- ▶ Para obtener el número total de operaciones realizamos la suma en k , correspondiente al bloque externo del algoritmo:

$$\sum_{k=1}^{n-1} 3(n-k) + 2(n-k)(n-k) = 3 \sum_{k=1}^{n-1} (n-k) + 2 \sum_{k=1}^{n-1} (n-k)^2.$$

- ▶ En los cálculos usaremos la fórmula: $\sum_{k=1}^{n-1} k^2 = \frac{(n-1)n(2n-1)}{6}$.
- ▶ Obteniendo así, la cantidad de operaciones

$$\frac{2}{3}n^3 + \frac{n^2}{2} - \frac{7}{6}n.$$

Complejidad de la eliminación de Gauss

- ▶ Para obtener el número total de operaciones en el método de eliminación gaussiana, necesitamos sumar el número de operaciones necesarias para resolver el sistema triangular.
- ▶ Así, la aplicación del método de eliminación de Gauss, el número de operaciones aritméticas es:

$$\frac{2}{3}n^3 + \frac{n^2}{2} - \frac{7}{6}n + n^2 = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n.$$

Si $n = 100$ se necesitan 681550 operaciones aritméticas.

- ▶ La complejidad del número de operaciones es $O(n^3)$.

Importancia del estudio de la Complejidad

Por ejemplo, si un algoritmo tiene complejidad $O(n!)$ y si consideramos que cada unidad de tiempo es de 10^{-9} segundos, entonces:

- ▶ Si $n = 13$, entonces $T(n) \approx 6$ segundos
- ▶ Si $n = 15$, entonces $T(n) \approx 21$ minutos
- ▶ Si $n = 16$, entonces $T(n) \approx 6$ horas
- ▶ Si $n = 17$, entonces $T(n) \approx 4$ días
- ▶ Si $n = 18$, entonces $T(n) \approx 2$ meses y medio
- ▶ Si $n = 20$, entonces $T(n) \approx 39$ años
- ▶ Si $n = 21$, entonces $T(n) \approx 821$ años
- ▶ Si $n = 22$, entonces $T(n) \approx 18\,068$ años

Conclusión

Ante la necesidad de resolver un problema debemos escoger el algoritmo más adecuado:

1. El algoritmo debe ser simple (de accesible implementación)
2. El algoritmo debe ser eficiente (usar el mínimo de recursos computacionales (estructura de datos) y debe ejecutar lo más rápido posible (complejidad asintótica).

6. Referencias

1. T. Cormen, C. Leiserson, R. Rivest, C. Stein (2003), Introduction to Algorithms, 4ta impresión, MIT Press-McGraw-Hill, Sección 3.1.
2. Szwarcfiter Jayme L. y Markenson Lilian, Estrutura de Dados e Seus Algoritmos, 2da edicao, 1994, Capítulo 1.
3. Campelo Eduardo R., y Maculan Nelson, Projeto e Análise de Algoritmos Heurísticos, Escola Brasileira de Otimizacao, 1989.