Universidad de Huelva

Grado en Ingeniería Informática

Análisis de la eficiencia de algoritmos. Complejidad temporal y espacial

Resumen

Autor: Alberto Fernández Merchán
Asignatura: Algorítmica y Modelos de Computación

Septiembre 2021

1. Introducción

La elección de los algoritmos está orientada hacia la disminución del costo que implica la solución. Existen dos criterios:

- Minimizar el costo de desarrollo: claridad, sencillez y facilidad de implementación.
- Minimizar el costo de ejecución: Tiempo de procesador y cantidad de memoria.

La eficiencia de los algoritmos es la medida del uso de los recursos en función del tamaño de la entrada.

2. Eficiencia y Complejidad

2.1. Coste Temporal y Espacial

El tiempo que emplea un algoritmo en ejecutarse de refleja la cantidad de trabajo realizado.

La complejidad temporal proporciona una medida de la cantidad de tiempo requerida para cierto algoritmo.

Como los resultados dependen del hardware y del tamaño de la entrada, se realiza un análisis teórico, expresando el resultado en la cantidad de operaciones elementales que realiza.

2.2. Análisis por conteo de operaciones elementales

Consideraremos operaciones elementales las siguientes:

1. Operaciones aritméticas básicas.
2. Asignaciones a variables.
3. Saltos
    a) Llamadas a funciones.
    b) Retorno de funciones.
4. Comparaciones lógicas.
5. Accesos a vectores.

2.3. Elección de operaciones básicas

Para hacer una estimación de la cantidad de tiempo se puede elegir una operación básica. De esta forma el conteo de operaciones básicas será proporcional al tiempo total de ejecución.

La operación básica debe estar relacionada con el tipo de problema que se quiere resolver.

2.4. Caso peor, mejor y medio

El **mejor caso** se presenta cuando para una entrada de tamaño n, el algoritmo ejecuta el mínimo número de operaciones.

El **peor caso** se presenta cuando para una entrada de tamaño n, el algoritmo ejecuta el máximo número de operaciones.

Para el **caso medio** se consideran todos los casos posibles. Se calcula el promedio de operaciones utilizando la probabilidad de que ocurra cada instancia.

3. Cotas de complejidad

El tiempo de ejecución está basado en constantes que dependen de factores externos. Estudiaremos el comportamiento de un algoritmo al aumentar el tamaño de los datos. Esto se conoce como eficiencia asintótica. De esta forma es más sencillo comparar la eficiencia de algoritmos diferentes.

Notaciones Asintóticas:

- $O(t)$**: Orden de complejidad de t. Cota superior.**

  Conjunto de todas las funciones de N en R^+ acotadas superiormente por un múltiplo real positivo de f. Por tanto:

  $O(f) = \{t : N \to R^+ \ / \ \exists\ c\ \in\ R^+, \exists\ n_0 \in\ N, \forall\ n \geq n_0\ ;\ t(n)\ \leq c \cdot f(n)\}$

  Ejemplo

  $T(n) = 9 + 7n$
  $T(n)\ \in\ O(n) \to (c = 8, n_0 = 9)(9 + 7 \cdot 9 \leq (8 \cdot 9))$
  $T(n)\ \in\ O(n^2) \to (c = 1, n_0 = 8 o, c = 2, n_0 = 7)\ (9 + 7 \cdot 8 \leq 8 \cdot 8)$
  $T(n) \notin O(\log_2 n)$

  Se cumple que:

  - $O(c) = O(d)$ siendo c y d constantes positivas.
  - $O(c) \subset O(n)$
  - $O(c \cdot n + b) = O(d \cdot n + e)$
  - $O(p) = O(q)$ si p y q son polinomios del mismo grado.
  - $O(p) \subset O(q)$ si $grado(p) < grado(q)$

- $\Omega(t)$**: Orden inferior de t. Cota inferior.**

  Es el conjunto de todas las funciones acotadas inferiormente por un múltiplo real positivo de f. Por tanto:

  $\Omega(f) = \{t : N \to R^+ \ / \ \exists\ c\ \in\ R^+, \exists\ n_0 \in\ N, \forall\ n \geq n_0\ ;\ t(n)\ \geq c \cdot f(n)\}$

  Ejemplo

  $T(n) = 5n^2 + 8n - 11$
  $T(n) \in O(n^2) \to\ (c = 6, n_0 = 7)(49 \cdot 5 + 8 \cdot 7 - 11 < 6 \cdot 49)$
  $T(n) \in \Omega(n^2) \to\ (c = 1, n_0 = 1)(5 + 8 - 11 > 1)$

- $\Theta(t)$**: Orden exacto de t. Cota exacta.**
  Es el conjunto de todas las funciones que crecen igual que f, asintóticamente y salvo constantes. Por tanto:

  $\Theta(f) = O(f) \cap \Omega(f) = \{t : N \to R^+ \ / \ \exists\, c, d\ \in\ R^+, \exists\, n_0 \in\ N, \forall\, n \geq n_0\ ;\ c{\cdot}f(n)\ \leq t(n)\ \leq d{\cdot}f(n)\}$

  Representa la complejidad exacta de una función.
  Indica que la función de complejidad está acotada tanto inferiormente como superiormente por una misma función.

  Ejemplo

  $T(n) = 5n^2 + 8n - 11$
  $T(n) \in O(n^2) \to\ (c = 6, n_0 = 7)(49 \cdot 5 + 8 \cdot 7 - 11 < 6 \cdot 49)$
  $T(n) \in \Omega(n^2) \to\ (c = 1, n_0 = 1)(5 + 8 - 11 > 1)$
  $T(n) \in \Theta(n^2)$

- Relación límites/órdenes

Para O

- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} \in R^+ \to O(f) = O(g)$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0 \to O(f) \subset O(g)$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = +\infty \to O(f) \supset O(g)$

Para Ω

- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} \in R^+ \to f(n) \in \Omega(g(n))\ y\ g(n) \in \Omega(f(n))$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0 \to g(n) \in \Omega(f(n))\ y\ f(n) \notin \Omega(g(n))$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = +\infty \to f(n) \in \Omega(g(n))\ y\ g(n) \notin \Omega(f(n))$

Para Θ

- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} \in R^+ \to f(n) \in \Theta(g(n))$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0 \to f(n) \in O(g(n)),\ pero\ f(n) \notin \Theta(g(n))$
- $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = +\infty \to O(f) \supset O(g)$

Θ permite clasificar las funciones de complejidad en conjuntos disjuntos. Estas categorías de complejidad se nombran mediante el componente más sencillo.

Alguna relaciones entre órdenes son:
$O(1) \subset O(\log_2 n) \subset O(n) \subset O(n \log 2n) \subset O(n^2) \subset O(n^j) \subset$
$\subset O(n^k) \subset O(a^n) \subset O(b^n) \subset O(n!) \subset O(n^n)$

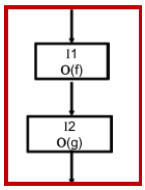
Cosas que recordar

- Los logaritmos son del mismo orden. Independientemente de la base.
- En los casos promedio se usan las probabilidades para calcular el orden de complejidad.
- En un polinomio, el orden es $O(x^n)$. Siendo n el máximo exponente.

4. Análisis de algoritmos iterativos

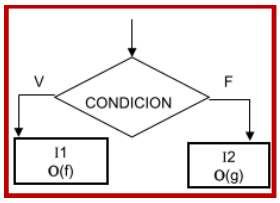
En un algoritmo sin llamdas recursivas, una vez seleccionados la talla y la operación básica, el análisis se realiza utilizando técnicas de conteo (sucesiones, progresiones aritméticas, sumas...).

- **Secuencia:** Siendo I_1, I_2 una secuencia de instrucciones, el orden de la secuencia será $O(max(f, g))$



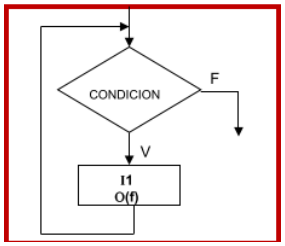
- **Selección Condicional:** Se evalúa la condición.

  - En el **peor** caso, $O(max(f, g))$.
  - En el **mejor** caso, $O(min(f, g))$
  - En el caso **promedio**, $O(Pf) + O((1 - P)g)$



- **Iteración:** Si el bloque de complejidad se realiza n veces. Deducimos que la complejidad será $O(nf(n))$

En algunas ocasiones depende de algún índice (i). En ese caso utilizaremos una progresión geométrica:



$$\sum_{i=1}^{n} f(i)$$

Ejemplo

Para obtener una entrada de c_{ij} se hacen n multiplicaciones.
Como la matriz tiene n entradas, se tiene que:

```
Algoritmo Matriz_Producto (n, A, B)
    para_ i = 1  hasta n  hacer                          I1
        para_ j = 1 hasta n  hacer                       I2
            C [i, j] ← 0;                                I3
            para_k = 1 hasta n  hacer                    I4
              C [i, j] ← C [i, j] + A [i, k] * B [k, j]  I5
            fpara
        fpara
    fpara
f Algoritmo
```

$$T(n) = n(n^2) = n^3 \in O(n^3)$$

Podemos asignar un orden a cada bloque de instrucciones según las estructuras de control.

- $I5 \in O(1)$
- $I5 \subset I4 \rightarrow O(n \cdot 1) = O(n)$
- $I4 \subset I2 \rightarrow O(n \cdot n) = O(n^2)$
- $I2 \subset I1 \rightarrow O(n^2 \cdot n) = O(n^3)$

En consecuencia, el algoritmo es de orden cúbico.

5. Análisis de algoritmos recursivos. Resolución de recurrencias

Al analizar algoritmos recursivos, se obtiene la función compleidad en forma de **sistema recurrente**. Para resolver estos sistemas:

- **Expansión de Recurrencias**
  Sustituir las llamadas recurrentes por su definición con el objetivo de encontrar una regla general.

  1. Se colocan todos los términos con T en el lado izquierdo.
  2. Se realiza la expansión.
  3. Al sumar las ecuaciones obtendremos una expresión para T(n) en términos de n y constantes.

  Ejemplo

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

  $T(n) - T(n-1) = 1$
  $T(n-1) - T(n-2) = 1$
  $T(n-2) - T(n-3) = 1$
  ...
  $T(2) - T(1) = 1$
  $T(1) - T(0) = 1$
  $T(0) = 0$ $(Caso\ Base)$

  Sumamos las ecuaciones:
  $T(n) = 1 + 1 + ... + 1$ $(n\ veces)$
  $T(n) = n \to T(n) \in O(n)$

- **Ecuación Característica**
  Útil para recurrencias lineales, donde n depende de los k valores anteriores. Existen dos casos:

  1. Recurrencias Homogéneas:

  $$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = 0$$

  Haciendo el cambio a $T(n) = x^n$, obtenemos:

  $$a_0x^n + a_1x^{n-1} + ... + a_kx^{n-k} = 0$$
  $$a_0x^k + a_1x^{k-1} + ... + a_k = 0$$

  Sean $r_1, ..., r_k$ las raíces de la ecuación. Según su multiplicidad se da:

  - Raíces Distintas: La forma de la función complejidad queda:
    $\sum_{i=1}^{k}(c_i \cdot r_i^n) = c_1 \cdot r_1^n + c_2 \cdot r_2^n + ... + c_k \cdot r_k^n$

- Raíces con multiplicidad: La forma de la función complejidad queda:
$$\sum_{i=1}^{k} \sum_{j=0}^{m_i-1} (c_{ij} \cdot n^j \cdot r_i^n)$$

2. Recurrencia No Homogéneas

$$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = \mathbf{f(n)}$$
$$a_0T(n) + a_1T(n-1) + ... + a_kT(n-k) = b^n \cdot p^d(n)$$

La ecuación característica es:
$(a_0x^k + a_1x^{k-1} + ... + a_k)(x-b)^{d+1} = 0$

- **Cambio de Variable**
Consiste en hacer un cambio de variable.
Se aplica cuando $n = a^k \leftrightarrow k = \log_2 n$

- **Recurrencias No Lineales**
Se intenta convertir la ecuación no lineal en lineal. Utilizando cambios de variable como: $t_k = T(2^k)$ y $u_k = \log_2 t_k$

- **Recurrencias típicas**

  - **Reducción por sustracción**

    Reducción por sustracción :

$$T(n) = \begin{cases} c & \text{Si } 0 \le n < b \\ a \cdot T(n-b) + p(n) & \text{Si } n \ge b \end{cases}$$

c es el coste en el caso directo,
a es el número de llamadas recursivas,
b es la disminución del tamaño de los datos, y
p(n)=c * n^k es el coste de preparación de las llamadas y de combinación de los resultados.

donde a, $c \in R^+$, p(n) es un polinomio de grado k ($p(n)=cn^k$), y $b \in N$

$$\text{entonces se tiene que } \mathbf{T(n)} \in \begin{cases} \Theta(\mathbf{n^k}) & \text{si } a < 1 \\ \Theta(\mathbf{n^{k+1}}) & \text{si } a = 1 \\ \Theta(\mathbf{a^{n/b}}) & \text{si } a > 1 \end{cases}$$

  - **Reducción por división**

    Reducción por división :

$$T(n) = \begin{cases} c & \text{Si } 1 \le n < b \\ a \cdot T(n/b) + f(n) & \text{Si } n \ge b \end{cases}$$

c es el coste en el caso directo,
a es el número de llamadas recursivas,
b factor de disminución del tamaño de los datos, y
f(n)= c * n^k es el coste de preparación de las llamadas y de combinación de los resultados.

donde a, $c \in R^+$, $f(n) \in \Theta(n^k)$, $b \in N$, $b > 1$

$$\text{entonces se tiene que } T(n) \in \begin{cases} \Theta(n^k) & \text{si } a < b^k \\ \Theta(n^k \log n) & \text{si } a = b^k \\ \Theta(n^{\log_b a}) & \text{si } a > b^k \end{cases}$$