

TEMA 1: ANÁLISIS DE ALGORITMOS

Existen dos recursos de analizar:

- * Coste o complejidad espacial: la cantidad de memoria que consume.
- * Coste o complejidad temporal: el tiempo que tarda en resolver el problema.

Para estudiar el comportamiento temporal de los algoritmos hay dos aproximaciones:

- * Análisis teórico o a priori: Dado un algoritmo, se trata de intentar establecer una expresión matemática que indique el comportamiento en función de los parámetros que influyan. Su utilidad es generalizar.
- * Análisis experimental o a posteriori: Se trata de ejecutar casos de prueba, midiendo tiempos de ejecución, lengüaje compilación etc... Su utilidad es caracterizar el comportamiento del programa en las condiciones de uso.

TALLA DEL PROBLEMA

Valor o conjunto de valores asociados a la entrada del problema que representa una medida de su tamaño.

PASO

Fraguento de código cuyo tiempo de proceso no depende de la talla del problema y está acotado por alguna cte

Lo * Cualquier operación elemental.

1º Ejemplo: cálculo del cuadrado de número n

Possibles soluciones:

- Realizar operación $n \cdot n$
- Bucle de n iteraciones, sumando n veces n
- Repetir n veces, n incrementos de variable unitaria

Operación $n \cdot n$

Función Solución 1 (n: entero) retorna (p: entero)

Var n: entero p: var

$n = n \cdot n$ → 2 pasos
retorna n → 4 paso

función

DATOS

Talles: n

Nº pasos: 3

* Principio de intuición: Dos implementaciones de un mismo algoritmo no difieren más que en una constante multiplicativa

Iterar n sumando n, n veces

Función Solución 2 (n: entero) retorna (p: entero)

Var n, i: entero p: var

$n = 0$ → 1 paso

para $i = 1$ hasta n hacer

$n = n + n$ → 2 pasos
n veces

retorna n → 1 paso

función

cómparea hasta llegar
a n+1, cuando ya
se salga del bucle.

* 1 paso asignación 1 por i=1
* n+1 pasos comparación n por i=n
* n pasos por i++ n veces
2 + 2n pasos

DATOS

Talles: n

Nº pasos: 4 + 4n

Iteran n veces n incrementos unitarios

Función Solución 3 (n: entero) retorna (p: entero)

Var n, i, j: entero p: var

$n = 0$ → 1 paso

para $i = 1$ hasta n hacer

para $j = 1$ hasta n hacer

$n++$ → n² pasos

↓ veces

↓ para

retorna n → 1 paso

función

+ 1 paso asig 1 por i=1

* n+1 pasos comparación n por i=n

* n pasos por i++ n veces 2n + 2 pasos

DATOS

Talles: n

Nº pasos: 3n² + 4n + 4

Un método de tiempo constante siempre será mejor que uno cuyo tiempo depende linealmente de la talla. NO siempre el coste de un algoritmo depende de la talla, existen otros factores...

*Instancia: Factor con el que varía el coste para una talla fija. Es un caso particular del problema, que hace que el algoritmo se compute una u otra forma.

Para una misma talla
el nº pasos varía

EJEMPLO:

Recorrer los elementos de un vector realizando una búsqueda posibilidades:

- x en $x = A[1]$
- x en $x = A[2]$
- x en $x = A[n]$

No encontrar x en A , (H_i) ($A[i] \neq x : 1 \leq i \leq n$)

En resumen, el nº pasos depende de la instancia.

Ante esto planteamos dos posibilidades:

- MEJOR CASO: instancias en las que el problema se resuelve más rápidamente para cada valor de la talla.
- PEOR CASO: instancias para cada valor de la talla, que se ejecutan con el mayor nº pasos posible.

*El mejor caso es encontrar x en la primera posición, el peor caso es no encontrarlo en el vector.

Tipos de costes:

- Coste mejor caso
- Coste promedio
- Coste peor caso

Notación Asintótica

Herramientas matemáticas que sirven para estudiar el comportamiento del algoritmo cuando es lo suficientemente grande. Characterizan el coste mediante funciones simples, que acotan superior e inferiormente el orden de las instancias para tallas lo suficientemente grandes.

Existen tres: O (o grande), Ω (Omega) y Θ (zeta)

Notación Asintótica O

~~Definición~~ Se dice que $f(n)$ es $O(g(n)) \Leftrightarrow \exists c \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N}$

$$f(n) \leq c g(n) \quad \forall n \geq n_0$$

$g(n)$ es cota superior de $f(n)$, $O(g(n))$ es el conjunto de funciones acotadas superiormente por $g(n)$.

Ejemplo

¿Pertenece $f(n) = 3n + 2$ a $O(n)$?

$c = \text{Coeficiente}$

Si $\forall n \geq 2 \quad 3n + 2 \leq 4n \rightarrow f(n) \in O(n)$ * $f(n)$ NO supera

¿Pertenece $f(n) = 10n^2 + 4n + 2$ a $O(n)$?

NUNCA a un múltiplo de $g(n)$ con los mismos valores grandes de n

* Se trata de encontrar cuando

$f(n) \leq c g(n)$ ocurre siempre a partir de un determinado valor

$3n + 2 \leq 4n \equiv 4n \text{ siempre será mayor que } 3n + 2 \text{ cuando } n \geq 2$

$10n^2 + 4n + 2 \leq cn \equiv 10n^2 + 4n + 2 \text{ siempre será mayor que } cn$, quizás no en una instancia, pero si en el resto de tallas

Ejemplo: $n=1 \Rightarrow f(n)=16 \quad c=20 \Rightarrow cg(n)=20 \quad cg(n) < f(n)$
 $n=2 \Rightarrow f(n)=50 \quad$ c es una constante $\Rightarrow cg(n)=40 \quad$ NO SE CUMPLE

Notación Asintótica Ω

Se dice que $f(n)$ es omega de $g(n) \Leftrightarrow \exists c \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N} \mid f(n) \geq c g(n) \forall n \geq n_0$. $g(n)$ es cota inferior de $f(n)$ // Igual que O pero al revés

Ejemplo:

¿Pertenece $f(n) = 10n^2 + 4n + 2 \in \Omega(n^2)$?

Sí, por ejemplo para $c=10 \Rightarrow f(n) \geq c g(n)$

$10n^2 + 4n + 2 \geq 10n^2$ El lado izq SIEMPRE será mayor que el dcho por lo que se cumple.

¿Pertenece $f(n) = 10n^2 + 4n + 2 \in \Omega(n)$?

Caso similar, si, ya que $f(n) \geq c g(n)$

$10n^2 + 4n + 2 \geq n$ siendo $c=1$ por tanto:

$$f(n) = 10n^2 + 4n + 2 \geq 1n, \forall n \geq 0 \Rightarrow f(n) \in \Omega(n)$$

Notación Asintótica Θ

Se dice que $f(n)$ y $g(n)$ son funciones de \mathbb{N} en \mathbb{R}^+ , sea $f(n)$ la función que expresa la complejidad temporal para tallo n ,

Se dice que $f(n)$ es zeta de $g(n) \Leftrightarrow$

$$\exists c_1 \in \mathbb{R}^+ \wedge \exists c_2 \in \mathbb{R}^+ \wedge \exists n_0 \in \mathbb{N} \mid c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall n \geq n_0$$

$g(n)$ es cota inferior de $f(n)$ y $g(n)$ es cota superior de $f(n)$

Jerarquía de cotas

Ordenes \rightarrow

$$\begin{aligned} O(1) &\subset O(\log n) \subset O(\sqrt{n}) \subset O(n) \subset O(n \log n) \subset O(n^2) \\ &\subset O(n^3) \subset O(2^n) \subset O(n^n) \end{aligned}$$

↑
omega

* Se debe proporcionar como cota el orden que más se ajuste a la función.

Notación

Sublineales:

$$\left\{ \begin{array}{l} \text{Ctes} \quad O(1) \\ \text{Logarítmicas} \quad O(\log n) \\ O(\sqrt{n}) \end{array} \right.$$

Lineales:

$$\left\{ \begin{array}{l} O(n) \end{array} \right.$$

Superlineales:

$$\left\{ \begin{array}{l} O(n \log n) \\ \text{Polinómicas:} \quad \left\{ \begin{array}{l} \text{cuadrática} \quad O(n^2) \\ \text{cúbicas} \quad O(n^3) \end{array} \right. \\ \text{Exponenciales:} \quad \left\{ \begin{array}{l} O(2^n) \\ O(n^n) \end{array} \right. \end{array} \right.$$

PROPIEDADES DE LAS COTAS

* (Cualquier polinomio de grado k es $O(n^k)$ y $\Omega(n^k)$, osea $\Theta(n^k)$)

* Regla Suma: $\Theta(f(n)) + \Theta(g(n)) = \Theta(f(n) + g(n)) = \Theta(\max\{|f(n)|, |g(n)|\})$
↳ También sirve con O y Ω

* Regla Producto: $\Theta(f(n)) \cdot \Theta(g(n)) = \Theta(f(n) \cdot g(n))$
↳ También sirve con O y Ω

* Si $f(n)$ y $g(n) \in \Theta(h(n))$ se deduce $(f(n) + g(n)) \in \Theta(h(n))$.
El conjunto $\Theta(h(n))$ es cerrado respecto a la suma de funciones.

* Si $f(n)$ y $g(n) \in \Theta(h(n))$ NO se deduce $f(n)g(n) \in \Theta(h(n))$.
El conjunto NO es cerrado respecto al producto de funciones.

* Si $f(n) \in \Theta(h(n))$ entonces $a \cdot f(n) + b \in \Theta(h(n)) \quad (\alpha \in \mathbb{R}^+ \wedge b \in \mathbb{R})$
Ejemplo: $f(n) = 3n + 3 \in \Theta(n) \Rightarrow 100(3n + 3) + 7 \in \Theta(n)$

Comparación de ordenes

Sean $f(n)$ y $g(n)$ funciones de \mathbb{N} en \mathbb{R}^+ . Se verifica

* Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = k \wedge k \neq 0$ entonces $f(n) \in \Theta(g(n))$ y $g(n) \in \Theta(f(n))$

* Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$ entonces $f(n)$ crece más rápidamente que $g(n)$

* Si $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$ entonces $g(n)$ crece más rápidamente que $f(n)$

Propiedad

$\forall a, b > 1 \in \mathbb{R}^+$, se cumple $\Theta(\log_a n) = \Theta(\log_b n)$

$$\text{¿ } \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = ? \Rightarrow \lim_{n \rightarrow \infty} \frac{\log_a n}{\log_b n} = \lim_{n \rightarrow \infty} \frac{\log_a [b \log_b n]}{\log_b n} \Rightarrow$$

$$\lim_{n \rightarrow \infty} \frac{\cancel{\log_b n} (\log_a n)}{\cancel{(\log_b n)}} \Rightarrow \lim_{n \rightarrow \infty} \log_a b = \boxed{\log_a b} \stackrel{\text{cte}}{=} \neq 0 \quad \text{ambos log se} \\ \text{duplican respectivamente.}$$

DATOS A ESTUDIAR

Talla: tamaño problema

Paso: operación elemental

Algoritmo: ordenación total pasos

Coste temporal Algoritmo: nº pasos y repeticiones

Instancia: caso concreto que varía el coste dependiendo para una talla fija.

Propiedades de O , Ω y Θ

Complejidad: dos tipos

Algoritmos iterativos: conteo de pasos significativos.

Algoritmos recursivos: métodos sobre relaciones de concurrencia.

25-08-2022

ALGORITMOS RECURSIVOS

$$T(n) = \begin{cases} c_1 & \text{caso base} \\ T(s(n)) + p(n) + c_2 & \text{otro caso} \end{cases}$$

$s(n) < n$ y lo habitual es $s(n) = n - c$ y/o $s(n) = \frac{n}{c} \cdot p(n)$

Esto se llama ecuación de recurrencia.

Ejemplo:

Función Factorial (n : entero) retorna (j : entero)

Si $n=0$ entonces

retorna 1 → CASO BASE

Sino

retorna $n * \text{Factorial}(n-1)$ → OTRO CASO

función

ECUACIÓN DE RECURRENCIA

$$T(n) = \begin{cases} c_1 & \text{si } n=0 \text{ (caso base)} \\ T(n-1) + c_2 & \text{(otro caso)} \end{cases}$$

Ahora por el método de sustitución iteradas expandiendo la ecuación hasta encontrar la expresión general.

$$T(n) = T(n-1) + c_2 = T(n-2) + 2c_2 = T(n-i) + i c_2$$

cuando $i = n$ alcanzaremos la base, $n=0$

$$T(n) = T(n-n) + nc_2 \Rightarrow T(0) + nc_2 = \underbrace{c_1 + nc_2}_{\text{caso base}}$$

Al ser polinomio:

$$\underline{T(n) \in \Theta(n)}$$

25-08-22

ALGORITMIA

Hay recurrencias que son muy complicadas de manejar, en este caso por ejemplo:

Función Ejemplo (n : entero) retorna (f : entero)

si $n=1$ entonces

retorna 1 → CASO BASE

sino

retorna $n * \text{Ejemplo}(n-1) * \text{Ejemplo}(n \text{ div } 2)$

si

Función

* Crecer de una forma exponencial, imposible!

→ OTRU CASO

ECUACIÓN DE RECURRENCIA

$$T(n) = \begin{cases} c_1 & n=1 \\ T(n-1) + T(n \text{ div } 2) + c_2 & n \neq 1 \end{cases}$$

DESARROLLO

$$T(n) = T(n-1) + T(n \text{ div } 2) + c_2 =$$

$$[T(n-2) + T((n-1) \text{ div } 2) + c_2] + [T(n \text{ div } 2 - 1) + T(n \text{ div } 2^2) + c_2] + c_2 *$$

Lo único que se puede hacer es acotar el orden de complejidad obteniendo la cota superior más pequeña y la inferior más grande

Por ejemplo:

$$T_1(n) = \begin{cases} c_1 & \text{si } n=1 \\ 2T_1(n \text{ div } 2) + T_1(n \text{ div } 2) + c_2 & \text{si } n \neq 1 \end{cases}$$

* Falta por resolver

⇒ Esto da que
 $T_1(n) \in \Theta(n)$

$$T_2(n) = \begin{cases} c_1 & \text{si } n=1 \\ 2T_2(n-1) + T_2(n-1) + c_2 & \text{si } n \neq 1 \end{cases}$$

* Falta por resolver

⇒ Esto da que
 $T_2(n) \in \Theta(2^n)$

Por tanto $T(n) \in \Omega(n)$ y $T(n) \in O(2^n)$ ↴

SOLUCIÓN

* T_1 es cota superior por lo que hay que coger la más pequeña, c_1 . Por tanto $T_1(n) \in \Theta(n)$

* T_2 es cota inferior por lo que cogemos más grande $2T_2(n-1) + c_2$ es $T_2(n) \in \Theta(2^n)$ s

FIN TEMA 1

25-08-2022

ALGORITMIA

* Página adicional para entender detalles

EJERCICIOS DE EXAMEN

"Completar huecos con la afirmación: el algoritmo iterativo $E(\log_2(n))$ y el recursivo $E(\Theta(3^n))$ ".

Función Examen1(...; n: entero)

Var i: entero fvar

i = 1

mientras ($i < n$) hacerhacer_algo(i) $\in \Theta(1)$ $\rightarrow i = i + 3$ RESULTADO

mientras

retorna 0

} procedimiento

Ejemplo: $\log_2 4 = 2$ DEBIDO A $2^2 = 4$ $\log_3 9 = 2$ DEBIDO A $3^2 = 9$ En este caso $\log_3 n = x \Rightarrow 3^x = n$ El programa tiene que iterar SIEMPRE sobre una potencia con base 3, 3^i .

SIEMPRE SE TRATA DE RECORRER

LA ITERACIÓN CON LA BASE DELLOGARITMO, DA IGUAL LA OPERACIÓN.

$$T(n) = \begin{cases} c_1 & \text{si } n=1 \\ 3T(n-1) + c_2 & \text{si } n>1 \end{cases}$$

Parce un n° con una potencia
n basta con multiplicar, parece
ser siempre así de sencilla
normalmente es $T(n-1) + c_2$
por tanto ↗

Función Examen2(...; n: entero)

Var i: entero fvar

i = n

mientras ($i > 1$) hacer

hacer_algo

 $\rightarrow i = i \text{ div } 4$

mientras

retorna 0

} función

OBJETIVO: $E\Theta(\log_4(n))$

$$T(n) = \begin{cases} c_1 & n=1 \\ T(c_1(n+1)) + T_2(n \text{ div } 2) & n \neq 1 \end{cases}$$

$$T_2 = \begin{cases} c_2/2 & n=1/2 \\ c_2 & n \neq 1/2 \end{cases}$$

$\log_3(n) \Rightarrow 3^y = n$

$$\log_3 n \Rightarrow 3^y = n$$

↓

$$\begin{cases} 3^1 = 3 \\ 3^2 = 9 \\ 3^3 = 27 \\ 3^4 = 81 \end{cases}$$

$$\log_3 81 \Rightarrow 3^4 = 81$$

$$\log_3 81 = 4$$

El bucle da 4 vueltas
multiplicando de 3 en 3.

$$\log_4 n = y \quad 4^y = n \Rightarrow$$

~~$n=16$~~
 $n=16$

$4 \Rightarrow 4^2 = 16$

RESULTADO $\log_a x = b$

↑ ↓

BASE EXPONENTE

$$T(n) = \begin{cases} c_1 & \text{si } n \leq 1 \\ 2 & \text{otro caso} \end{cases}$$

EJERCICIO 4

La talla del problema es m , el nº de elementos del vector. Existe una distinción entre si n , es par o impar, pero como para una talla fija del problema, el nº de pasos es siempre el mismo. No tiene peor ni mejor caso. Como hace unas pasos v otros hay que acotar la función obteniendo la mayor de las cotas inferiores y la inferior de la cota superior.

$$T_{\text{par}}(m) = \begin{cases} c_1 & m=1 \\ T_{\text{par}}(m-2) + c_2 & m \geq 1 \end{cases} \Rightarrow \underline{T_{\text{par}}(m)} \in \Theta(n)$$

$$\log_2 n = x \quad 2^x = n$$

$$T_{\text{impar}}(n) \cong 3 + \sum_{i=0}^{\frac{n}{2}} 2 = 3 + 2 \left(\frac{n}{2} + 1 \right) \Rightarrow n+1 \text{ por tanto}$$

$$\underline{T(n)} \in \Theta(n)$$

ALGORITMIA

20-09-2022

Ejercicios

$$T_{MC}(n) = 2 \in \Theta(1)$$

$$T_{PC}(n) = 2 + \sum_{i=1}^n 1 = 2+n \in \Theta(n)$$

$$T(n)$$

Ejemplo 3

talla: $n \rightarrow$ tamaño vector, espacio búsqueda

Mc y Pc: NO, para una talla fija tiene el mismo costo temporal

$$T(n) \in \Omega(n)$$

Ejemplo 4

talla: $n \times n$ porque recorremos la matriz, con sus filas y columnas.

Mc y Pc: NO, para una talla fija siempre tendrá un mismo costo temporal.

$$T(n) \in \Theta(n^2) \quad (n \times n)$$

~~Matriz~~

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

$$T(n, n) = \sum_{i=1}^n \left[\sum_{j=1}^n 1 \right] = \sum_{i=1}^n n = n \sum_{i=1}^n 1 = n \times n \in \Theta(n^2)$$

EJEMPLOS = TRAMPA ES EL 6

talla: n , ya que es la entrada del problema

M_C y P_C : NO

$$T_i = T_{i-1} + \sum_{j=1}^n$$

Cota:

$$T(n) = 4 + \sum_{i=1}^n \left[\sum_{j=1}^i [2] + 1 \right] = \sum_{i=1}^n (2i+1) + 4$$

$2 \sum_{i=1}^n i$

$$\Rightarrow \in \Theta(n^2) \Rightarrow \boxed{n}$$

21-09-22

$$\begin{aligned} T_2(n) &= 2T\left(\frac{n}{2}\right) + c_2 = 2 \left[2T\left(\frac{n}{4}\right) + c_2 \right] + c_2 \\ 4T\left(\frac{n}{4}\right) + (2c_2 + c_2) &= 4 \left[2T\left(\frac{n}{8}\right) + c_2 \right] + (2c_2 + c_2) \\ &= 2^{\frac{n}{2}} (T(n \text{ div } 2^{\frac{n}{2}})) + (2^{\frac{n}{2}} - 1)c_2 \\ n \text{ div } 2^{\frac{n}{2}} &= 1 \quad \frac{n}{2^{\frac{n}{2}}} = 1 \quad n = 2^{\frac{n}{2}} \end{aligned}$$