

Estructuras de Datos

Clase 2 – Análisis de algoritmos



Dr. Sergio A. Gómez
<http://cs.uns.edu.ar/~sag>



Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Bahía Blanca, Argentina

Motivaciones

- Estamos interesados en construir *buenas* estructuras de datos y *buenos* algoritmos.
- Estructura de datos = Manera sistemática de organizar y acceder datos.
- Algoritmo = Procedimiento paso a paso para realizar una tarea en una cantidad finita de tiempo.
- ¿Cómo analizar algoritmos y estructuras de datos para decidir si son buenos o no?

Motivaciones

- La herramienta de análisis que usaremos consiste de caracterizar el “tiempo de ejecución” de algoritmos y operaciones de estructuras de datos (con uso de espacio de memoria también de interés).
- Objetivo: Una aplicación debe correr lo más rápidamente posible.

Factores que afectan el tiempo de ejecución

- Aumenta con el tamaño de la entrada de un algoritmo
- Puede variar para distintas entradas del mismo tamaño
- Depende del hardware (velocidad del reloj, procesador, cantidad de memoria, tamaño del disco, ancho de banda de la conexión a la red)
- Depende del sistema operativo
- Depende de la calidad del código generado por el compilador
- Depende de si el código es compilado o interpretado

Cómo medir el tiempo de ejecución: (1) Estudio experimental

- Con un algoritmo implementado, hacer varias corridas sobre distintas entradas y realizar un gráfico de dispersión (n, t) con n=tamaño de la entrada natural y t=tiempo de corrida en milisegundos.
- Problemas:
 - Se puede hacer con un número limitado de datos
 - Dos algoritmos son incomparables a menos que hayan sido testeados en los mismos ambientes de hardware y software
 - Hay que implementar el algoritmo para hacer el test

Estructuras de datos - Dr. Sergio A. Gómez

5

Cómo medir el tiempo de ejecución: (2) Orden del tiempo de ejecución

- Toma en cuenta todas las posibles entradas
- Permite evaluar la eficiencia relativa de dos algoritmos independientemente del ambiente de hardware y software
- Puede realizarse estudiando una versión de alto nivel del algoritmo sin necesidad de implementarlo o hacer experimentos.

Estructuras de datos - Dr. Sergio A. Gómez

6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
"Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2018.

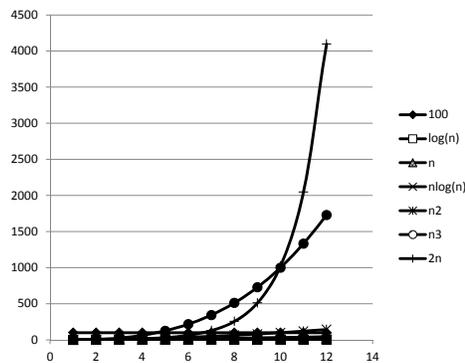
Preliminares: Funciones

- Constante: $f(n) = c$
- Logaritmo: $f(n) = \log_b(n)$ para $b > 1$
- Lineal: $f(n) = n$
- N-LogN: $f(n) = n \log(n)$
- Cuadrática: $f(n) = n^2$
- Cúbica: $f(n) = n^3$
- Polinomial: $f(n) = n^k$, con k natural
- Exponencial: $f(n) = a^n$ con a real positivo y n
- Factorial: $f(n) = n!$

Estructuras de datos - Dr. Sergio A. Gómez

7

Funciones comparadas



n	100log(n)	N	nlog(n)	n ²	n ³	2 ⁿ
1	100	0	1	0	1	2
2	100	1	2	2	4	4
3	100	2	3	5	9	8
4	100	2	4	8	16	16
5	100	2	5	12	25	32
6	100	3	6	16	36	64
7	100	3	7	20	49	128
8	100	3	8	24	64	256
9	100	3	9	29	81	512
10	100	3	10	33	100	1024
11	100	3	11	38	121	2048
12	100	4	12	43	144	4096
13	100	4	13	48	169	8192
14	100	4	14	53	196	16384
15	100	4	15	59	225	32768
16	100	4	16	64	256	65536
17	100	4	17	69	289	131072
18	100	4	18	75	324	262144
19	100	4	19	81	361	524288
20	100	4	20	86	400	1048576
21	100	4	21	92	441	2097152
22	100	4	22	98	484	4194304
23	100	5	23	104	529	8388608
24	100	5	24	110	576	16777216

Estructuras de datos - Dr. Sergio A. Gómez

8

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2018.

Repaso de Análisis Matemático

- Regla de L'Hôpital:

$$\lim_{x \rightarrow a} \frac{f(x)}{g(x)} = \lim_{x \rightarrow a} \frac{f'(x)}{g'(x)} = L$$

Ejemplo:

$$\lim_{x \rightarrow \infty} \frac{\log(x)}{3x} = \lim_{x \rightarrow \infty} \frac{1/x}{3} = \lim_{x \rightarrow \infty} \frac{1}{3x} = 0$$

Repaso de Análisis Matemático

Nota: Recuerde que dadas dos funciones $f(x)$ y $g(x)$ de los reales en los reales, si existe un número a tal que $f(a) \leq g(a)$ y $f'(x) \leq g'(x)$ para todo $x \geq a$, entonces $f(x) \leq g(x)$ para todo $x \geq a$.

Nota: " $f(x) = O(g(x))$ " se lee "f es del orden de g" o "g es una cota asintótica para f" o "f es menor que g asintóticamente" o "g crece más rápidamente que f".

Recuerde que $f(x) = O(g(x))$ si y sólo si

$$\lim_{x \rightarrow \infty} f(x) / g(x) < +\infty$$

Repaso de Análisis Matemático

Nota: $\lim_{x \rightarrow \infty} f(x) / g(x)$ es 0 entonces $g(x)$ crece más rápidamente que $f(x)$.

Ejemplo: El logaritmo crece más lentamente que la función lineal $3x$ porque (usando L'Hôpital):

$$\lim_{x \rightarrow \infty} \frac{\log(x)}{3x} = \lim_{x \rightarrow \infty} \frac{1/x}{3} = \lim_{x \rightarrow \infty} \frac{1}{3x} = 0$$

Repaso de Análisis Matemático

Nota: $\lim_{x \rightarrow \infty} f(x) / g(x)$ es una constante mayor que 0 entonces $f(x)$ y $g(x)$ tienen la mismo orden (o tasa de crecimiento).

Ejemplo: $3x^2$ y $40x^2$ son asintóticamente equivalentes porque:

$$\lim_{x \rightarrow \infty} \frac{3x^2}{40x^2} = \frac{3}{40} = 0,075$$

Repaso de Análisis Matemático

Nota: $\lim_{x \rightarrow \infty} f(x) / g(x)$ es ∞ entonces $f(x)$ crece más rápidamente que $g(x)$.

Ejemplo: $5x^3$ crece más rápidamente que $40x^2$ porque:

$$\lim_{x \rightarrow \infty} \frac{5x^3}{40x^2} = \lim_{x \rightarrow \infty} \frac{5x}{40} = \infty$$

Tiempo de un algoritmo

- El tiempo de ejecución de un algoritmo depende de la cantidad de operaciones primitivas realizadas.
- Las operaciones primitivas toman tiempo constante y son:
 - Asignar un valor a una variable
 - Invocar un método
 - Realizar una operación aritmética
 - Comparar dos números
 - Indexar un arreglo
 - Seguir una referencia de objeto
 - Retornar de un método
- Como estos tiempos dependen del compilador y del hardware subyacente, por ello los notaremos con constantes arbitrarias c_1, c_2, c_3, \dots

Notación asintótica (Big-Oh)

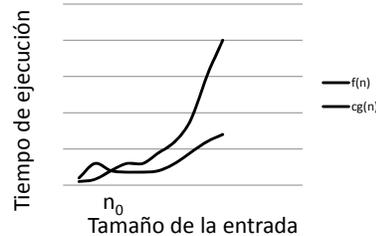
Sean $f(n)$ y $g(n) : \mathbb{N} \rightarrow \mathbb{R}$

$f(n)$ es $O(g(n))$ ssi existen c real con $c > 0$ y n_0 natural con $n_0 \geq 1$ tales que

$$f(n) \leq cg(n) \text{ para todo } n \geq n_0$$

“ $f(n)$ es $O(g(n))$ ” se lee “ $f(n)$ es big-oh de $g(n)$ ” o “ $f(n)$ es del orden de $g(n)$ ”

También se denota como
 $f(n) = O(g(n))$



Estructuras de datos - Dr. Sergio A. Gómez

15

Reglas de la suma y el producto

- Regla de la suma: Si $f_1(n)$ es $O(g_1(n))$ y $f_2(n)$ es $O(g_2(n))$ entonces $f_1(n) + f_2(n)$ es $O(\max(g_1(n), g_2(n)))$
- Regla del producto: Si $f_1(n)$ es $O(g_1(n))$ y $f_2(n)$ es $O(g_2(n))$ entonces $f_1(n) * f_2(n)$ es $O(g_1(n) * g_2(n))$

Estructuras de datos - Dr. Sergio A. Gómez

16

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
“Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2018.

Lema

Si $a \leq b$ y $c \leq d$ entonces $a + c \leq b + d$

Demostración:

Si $a \leq b$ entonces $b - a \geq 0$

Si $c \leq d$ entonces $d - c \geq 0$

La suma de dos números mayores o iguales a 0 es mayor o igual a 0, por lo tanto:

$$(b - a) + (d - c) \geq 0$$

Luego, $a + c \leq b + d$.

- Regla de la suma: Si $f_1(n)$ es $O(g_1(n))$ y $f_2(n)$ es $O(g_2(n))$ entonces $f_1(n) + f_2(n)$ es $O(\max(g_1(n), g_2(n)))$

- Demostración:

Si $f_1(n)$ es $O(g_1(n))$ entonces existen c_1 real y n_1 natural tales que $f_1(n) \leq c_1 g_1(n)$ para todo $n \geq n_1$

Si $f_2(n)$ es $O(g_2(n))$ entonces existen c_2 real y n_2 natural tales que $f_2(n) \leq c_2 g_2(n)$ para todo $n \geq n_2$

Sea $n_0 = \max(n_1, n_2)$, entonces

$$f_1(n) \leq c_1 \max(g_1(n), g_2(n)) \text{ para } n \geq n_0$$

$$f_2(n) \leq c_2 \max(g_1(n), g_2(n)) \text{ para } n \geq n_0$$

Entonces $f_1(n) + f_2(n) \leq (c_1 + c_2) \max(g_1(n), g_2(n))$ para todo $n \geq n_0$.

Luego $f_1(n) + f_2(n)$ es $O(\max(g_1(n), g_2(n)))$, q.e.d.

- Regla del producto: Si $f_1(n)$ es $O(g_1(n))$ y $f_2(n)$ es $O(g_2(n))$ entonces $f_1(n) * f_2(n)$ es $O(g_1(n) * g_2(n))$

- Demostración:

Si $f_1(n)$ es $O(g_1(n))$ entonces existen c_1 real y n_1 natural tales que $f_1(n) \leq c_1 g_1(n)$ para todo $n \geq n_1$

Si $f_2(n)$ es $O(g_2(n))$ entonces existen c_2 real y n_2 natural tales que $f_2(n) \leq c_2 g_2(n)$ para todo $n \geq n_2$

Sea $n_0 = \max(n_1, n_2)$, entonces

$f_1(n) \leq c_1 g_1(n)$ para $n \geq n_0$

$f_2(n) \leq c_2 g_2(n)$ para $n \geq n_0$

Entonces $f_1(n) * f_2(n) \leq (c_1 g_1(n))(c_2 g_2(n)) = (c_1 c_2)(g_1(n) g_2(n))$ para todo $n \geq n_0$.

Luego $f_1(n) * f_2(n)$ es $O(g_1(n) * g_2(n))$, q.e.d.

Propiedades útiles

Para probar por inducción:

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

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1 \quad \sum_{i=0}^n r^i = \frac{r^{n+1} - 1}{r - 1}$$

De las series:

$$\sum_{i=1}^n (a_i + b_i) = \sum_{i=1}^n a_i + \sum_{i=1}^n b_i \quad \sum_{i=1}^n c a_i = c \sum_{i=1}^n a_i$$

Big-Omega y Big-Theta

- **Big-Omega:** Sean $f(n)$ y $g(n)$ funciones de los naturales en los reales. $f(n)$ es $\Omega(g(n))$ ssi existen c real positivo y n_0 natural tales que $f(n) \geq cg(n)$ para todo $n \geq n_0$.
- **Big-Theta:** Sean $f(n)$ y $g(n)$ funciones de los naturales en los reales. $f(n)$ es $\Theta(g(n))$ ssi $f(n)$ es $O(g(n))$ y $f(n)$ es $\Omega(g(n))$.
- **Nota:** Big-Theta quiere decir $c_1g(n) \leq f(n) \leq c_2g(n)$. Por lo tanto, tienen un crecimiento asintótico equivalente.

Ejemplo

Ejercicio: Mostrar que $3n^2+2n+5$ es $O(n^2)$

Proc: Hay que hallar c real y n_0 natural tal que $3n^2+2n+5 \leq cn^2$ para $n \geq n_0$.

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

$$2n \leq 2n^2$$

$$5 \leq 5n^2$$

$$\text{Luego } 3n^2+2n+5 \leq 3n^2 + 2n^2 + 5n^2 = (3+2+5)n^2 = 10n^2$$

Por lo tanto: $c=10$

Para hallar n_0 resolver $10n^2 \geq 3n^2+2n+5$, lo que da $n \leq -5/7$ y $n \geq 1$. Luego $n_0=1$.

Ejemplo

Prop: Mostrar que un polinomio en n de grado k es de orden n^k .

Dem: Sea $P(n) = c_k n^k + c_{k-1} n^{k-1} + \dots + c_1 n + c_0$

Como $C_k n^k \leq |C_k| n^k$, $C_{k-1} n^{k-1} \leq |C_{k-1}| n^k$, ..., $C_1 n \leq |C_1| n^k$, $C_0 \leq |C_0| n^k$

Será que $P(n) \leq \sum_{i=0..k} |C_i| n^k = (\sum_{i=0..k} |C_i|) n^k$

Problema: Para un problema P cuyo tiempo está dado por $T(n)=200n$ expresado en microsegundos, determine el tamaño máximo n de P que puede ser resuelto en 1 segundo.

Solución: Si $T(n) = 200n$ entonces $T(1) = 200 \mu\text{seg}$,
 $T(2) = 400 \mu\text{seg}$.

¿Cuál será n para $T(n) = 1 \text{ seg} = 10^6 \mu\text{seg}$?

$T(n) = 200n = 10^6$.

Luego $n = 10^6 / 200 = 5.000$.

Por lo tanto, el tamaño de la entrada para tener el tiempo de corrida de P acotado en un segundo es cinco mil.

Reglas para calcular $T(n)$ a partir del código fuente de un algoritmo

Paso 1: Determinar el tamaño de la entrada n

Paso 2: Aplicar las siguientes reglas en forma sistemática:

- $T_p(n) = \text{constante}$ si P es una acción primitiva
- $T_{S_1; \dots; S_k}(n) = T_{S_1}(n) + \dots + T_{S_k}(n)$
- $T_{\text{if } B \text{ then } S_1 \text{ else } S_2}(n) = T_B(n) + \max(T_{S_1}(n), T_{S_2}(n))$
- $T_{\text{for}(m; S)}(n) = m * T_S(n)$ donde $m = \text{cant_iteraciones}(\text{for}(m; S))$
- $T_{\text{while } B \text{ do } S}(n) = m * (T_B(n) + T_S(n)) + T_B(n)$ donde $m = \text{cant_iteraciones}(\text{while } B \text{ do } S)$
- $T_{\text{call } P}(n) = T_S(n)$ donde `procedure P; begin S end`
- $T_{f(e)}(n) = T_{x:=e}(n) + T_S(n)$ donde `function f(x); begin S end`

Estructuras de datos - Dr. Sergio A. Gómez

25

Ejercicio: Mostrar que la búsqueda lineal en un arreglo de n enteros tiene orden n .

```
public static int lsearch( int [] a, int n, int x )
{
    int i = 0;
    boolean encuentre = false;
    while( i < n && !encuentre )
        if( a[i] == x )
            encuentre = true;
        else i++;
    if( encuentre ) return i;
    else return -1;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

26

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2018.

Tamaño de la entrada: n = cantidad de componentes de a

Peor caso: x no está en a

```
public static int lsearch( int [] a, int n, int x )
{
    int i = 0;                c1
    boolean encuentre = false; c2
    while( i < n && !encuentre ) Peor caso: n iteraciones
        if( a[i] == x )        Tiempo de condición: c3
            encuentre = true;  Tiempo del cuerpo: c4
        else i++;
    if( encuentre ) return i;   Tiempo de este if: c5
    else return -1;
}
```

$T(n) = c_1 + c_2 + (n(c_3+c_4) + c_3) + c_5 = O(n)$

El tiempo es *lineal* en el tamaño de la entrada.

Ejercicio: Mostrar que la búsqueda binaria (dicotómica) en un arreglo de n componentes ordenadas en forma ascendente tiene orden logarítmico de base 2 en la cantidad de elementos del arreglo.

```
public static int bsearch( int [] a, int n, int x ) {
    int ini = 0, fin = n-1;
    while( ini <= fin ) {
        int medio = (ini + fin) / 2;
        if( a[medio] == x ) return medio;
        else if( a[medio] > x ) fin = medio-1;
        else ini = medio + 1;
    }
    return -1;
}
```

Tamaño de la entrada: n = cantidad de componentes de a
 Peor caso: x no está en a

```
public static int bsearch( int [] a, int n, int x ) {
    int ini = 0, fin = n-1;            $c_1$ 
    while( ini <= fin ) {             Tiempo de la condición:  $c_2$ 
        int medio = (ini + fin) / 2;   Tiempo del cuerpo:  $c_3$ 
        if( a[medio] == x ) return medio;
        else if( a[medio] > x ) fin = medio-1;
        else ini = medio + 1;
    }
    return -1;                          $c_4$ 
}
```

Sea k = cantidad de iteraciones del while, entonces

$$T(n) = c_1 + k(c_2 + c_3) + c_2 + c_4.$$

La pregunta es cómo definir k en función de n .

¿Cómo estimar k en función de n ?

El peor caso es cuando “ x ” no está en “ a ”.

Veamos cómo vamos descartando componentes del arreglo en función del número de iteración del while: Si tenemos n componentes, en cada iteración se descarta la componente del medio del arreglo, entonces de las $n-1$ componentes que falta revisar sólo se va considerar la mitad, entonces quedan $(n-1)/2$ componentes para la siguiente iteración, y así sucesivamente. Calculemos cuál es caso para la iteración genérica k .

Número de iteración	Componentes por revisar
1	n
2	$(n-1)/2$
3	$(n-3)/4$
4	$(n-7)/8$
5	$(n-15)/16$
...	...
k	$\frac{n - (2^{k-1} - 1)}{2^{k-1}}$

Entonces vimos que en la iteración k , la cantidad de componentes que quedan por revisar es:

$$\frac{n - (2^{k-1} - 1)}{2^{k-1}}$$

Como x no está en el arreglo a , en la última iteración completa que realiza el while queda una componente del arreglo por revisar.

Entonces:

$$\frac{n - (2^{k-1} - 1)}{2^{k-1}} = 1;$$

$$n - (2^{k-1} - 1) = 2^{k-1};$$

$$n - 2^{k-1} + 1 = 2^{k-1};$$

$$n + 1 = 2^{k-1} + 2^{k-1};$$

$$n + 1 = 2 \times 2^{k-1};$$

$$n + 1 = 2^k;$$

$$\log_2(n + 1) = k.$$

Con lo que $T(n) = c_1 + \log_2(n+1)(c_2+c_3) + c_2 + c_4$.

Luego, por regla de la suma, $T(n) = O(\log_2(n))$ q.e.d.

Ejercicio: Mostrar que el método de ordenamiento de arreglos de *selección* tiene orden cuadrático en la cantidad de elementos del arreglo.

```
public static void selection_sort( int []a, int n ) {
    for( int i=0; i<n-1; i++ ) { // Repetir n-1 veces
        int k = i; // Hallar k tq a_k es el mínimo de
        for( int j=i+1; j<n; j++ ) // a_i, a_{i+1}, ..., a_{n-1}
            if( a[j] < a[k] )
                k = j;
        swap( a, i, k ); // Intercambiar a_i con a_k
    }
}
```

Tamaño de la entrada: n = cantidad de componentes de a .

```
public static void selection_sort( int []a, int n ) {
    for( int i=0; i<n-1; i++ ) { // n-1 iteraciones (desiguales)
        int k = i; // c1
        for( int j=i+1; j<n; j++ ) // (n-1)-(i+1)+1 iteraciones
            if( a[j] < a[k] ) // Tiempo del if: c2
                k = j;
        swap( a, i, k ); // Tiempo de swap: c3
    }
}
```

$$T(n) = \sum_{i=0}^{n-2} (c_1 + ((n-1) - (i+1) + 1)c_2 + c_3)$$

$$T(n) = \sum_{i=0}^{n-2} (c_1 + ((n-1) - (i+1) + 1)c_2 + c_3) =$$

$$= \sum_{i=0}^{n-2} c_1 + c_2 \sum_{i=0}^{n-2} (n - i - 1) + \sum_{i=0}^{n-2} c_3$$

$$= c_1(n - 2 - 0 + 1) + c_2 \sum_{i=0}^{n-2} (n - i - 1) + c_3(n - 2 - 0 + 1)$$

$$= c_1(n - 1) + c_2 \sum_{i=0}^{n-2} (n - i - 1) + c_3(n - 1)$$

Falta ver cuánto vale el término del medio.

$$\begin{aligned} & \sum_{i=0}^{n-2} (n - i - 1) = \\ & = (n - 1) + (n - 2) + (n - 3) + \dots + (n - (n - 2) - 1) = \\ & = (n - 1) + (n - 2) + (n - 3) + \dots + 1 = \\ & \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} \end{aligned}$$

Entonces, sabíamos que:

$$\begin{aligned} T(n) &= c_1(n-1) + c_2 \sum_{i=0}^{n-2} (n-i-1) + c_3(n-1) = \\ &= c_1(n-1) + c_2 \frac{n(n-1)}{2} + c_3(n-1) = O(n^2) \end{aligned}$$

Bibliografía

- Goodrich & Tamassia. Data Structures and Algorithms. Fourth Edition. Capítulo 4.

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2018.