INF221 – Algoritmos y Complejidad

Clase #16
Dividir y Conquistar

Aldo Berrios Valenzuela   Horst H. von Brand

Martes 4 de octubre de 2016

1. Dividir y Conquistar

Una de las mejores estrategias para diseñar algoritmos.

Idea: Dado un problema grande, reducirlo a varios problemas menores del mismo tipo, y combinar resultados.

Ejemplo 1.1 (Merge Sort)**.** Para ordenar N elementos:

- Dividir en "mitades" de $\left\lfloor \frac{N}{2} \right\rfloor$ y $\left\lceil \frac{N}{2} \right\rceil$
- Ordenarlas recursivamente.
- Intercalar resultados.

■

Ejemplo 1.2 (Búsqueda binaria)**.** Un arreglo ordenado de N elementos, y una clave a buscar. Obtener el elemento en la posición $\left\lfloor \frac{N}{2} \right\rfloor$, buscar en la mitad que tiene que contener la clave ■

Ejemplo 1.3 (Multiplicación de Karatsuba)**.** Para multiplicar números de $2n$ dígitos, dividimos ambos en mitades [2]:

$$A = 10^n a + b$$
$$B = 10^n c + d$$

con $a, b, c, d < 10^n$.

Además:

$$A \cdot B = 10^{2n} ac + 10^n (ad + bc) + bd \tag{1.1}$$

Notando que:

$$(a + b) \cdot (c + d) = ac + ad + bc + bd - (ad + bc) + ac + bd$$

Podemos calcular los coeficientes de (1.1) con 3 (no 4) multiplicaciones:

$$ac$$
$$bd$$
$$(a + b)(c + d) - ac - bd$$

■

Ejemplo 1.4. Otro ejemplo de esta estrategia es el algoritmo de Strassen [6] para multiplicar matrices. Consideremos primeramente el producto de dos matrices de 2×2:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

Sabemos que:

$$\begin{aligned} c_{11} &= a_{11} b_{11} + a_{12} b_{21} & c_{12} &= a_{11} b_{12} + a_{12} b_{22} \\ c_{21} &= a_{21} b_{11} + a_{22} b_{21} & c_{22} &= a_{21} b_{12} + a_{22} b_{22} \end{aligned}$$

Esto corresponde a 8 multiplicaciones. Definamos los siguientes productos:

$$\begin{aligned} m_1 &= (a_{11} + a_{22})(b_{11} + b_{22}) & m_2 &= (a_{21} + a_{22})\, b_{11} \\ m_3 &= a_{11}(b_{12} - b_{22}) & m_4 &= a_{22}(b_{21} - b_{11}) \\ m_5 &= (a_{11} + a_{12})\, b_{22} & m_6 &= (a_{21} - a_{11})(b_{11} + b_{12}) \\ m_7 &= (a_{12} - a_{22})(b_{21} + b_{22}) \end{aligned}$$

Entonces podemos expresar:

$$\begin{aligned} c_{11} &= m_1 + m_4 - m_5 + m_7 & c_{12} &= m_3 + m_5 \\ c_{21} &= m_2 + m_4 & c_{22} &= m_1 - m_2 + m_3 + m_6 \end{aligned}$$

Con estas fórmulas se usan 7 multiplicaciones para evaluar el producto de dos matrices. Cabe hacer notar que estas fórmulas no hacen uso de conmutatividad, por lo que son aplicables también para multiplicar matrices de 2×2 cuyos elementos son a su vez matrices. Podemos usar esta fórmula recursivamente para multiplicar matrices de $2^n \times 2^n$. ■

1.1. Estructura común

Un problema de tamaño nb se reduce a a problemas de tamaño n, que se resuelven recursivamente y las soluciones se combinan. Si el trabajo para resolver una instancia de tamaño n la llamamos $t(n)$, y el trabajo para reducir y combinar soluciones lo llamamos f:

$$t(nb) = a\,t(n) + f(n), \qquad t(1) = t_1 \tag{1.2}$$

Buscamos resolver esta recurrencia. Suponiendo que n es una potencia de b

$$\begin{aligned} n = b^k \qquad t\left(b^k\right) &= T(k) \qquad \left(k = \log_b(n)\right) \\ T(k+1) &= aT(k) + f\left(b^k\right) \end{aligned}$$

En la mayoría de los casos de interés, $f(n) = cn^d$

$$T(k+1) = aT(k) + cb^{kd} \qquad T(0) = t_1$$

Definimos:

$$g(z) = \sum_{k \geq 0} T(k)\, z^k$$

Por propiedades:

$$\frac{g(z) - t_1}{z} = ag(z) + \frac{c}{1 - b^d z}$$

Entonces, por propiedades:

$$g(z) = \frac{t_1 - \left(b^d t_1 - c\right) z}{\left(1 - b^d z\right)(1 - az)}$$

Hay que distinguir los casos en que los factores del denominador son iguales o distintos.

Caso $a = b^d$:

$$\begin{aligned}
g(z) &= \frac{t_1 - (at_1 - c)\,z}{(1-az)^2} \\
&= \frac{c}{a} \cdot \frac{1}{(1-az)^2} + \frac{at_1 - c}{a} \cdot \frac{1}{1-az} \\
&= \frac{c}{a} \sum_{k \geq 0} \binom{-2}{k} a^k z^k + \frac{at_1 - c}{a} \cdot \sum_{k \geq 0} a^k z^k \\
&= \frac{c}{a} \cdot \sum_{k \geq 0} \binom{k+1}{1} a^k z^k + \frac{at_1 - c}{a} \cdot \sum_{k \geq 0} a^k z^k \\
\left[z^k\right] g(z) &= \frac{c}{a} (k+1)\, a^k + \frac{at_1 - c}{a} \cdot a^k \\
T(k) &= \frac{c}{a} k a^k + t_1 \cdot a^k \\
&\sim \frac{c}{a} k a^k \\
t(n) &\sim \frac{c}{a} a^{\log_b n} \cdot \log_b n
\end{aligned}$$

Pero:

$$\begin{aligned}
a^{\log_b n} &= \left(b^{\log_b a}\right)^{\log_b n} \\
&= \left(b^{\log_b n}\right)^{\log_b a} \\
&= n^{\log_b a} = n^d
\end{aligned}$$

$$\begin{aligned}
t(n) &\sim \frac{c}{a} n^{\log_b a} \log_b n \\
&\sim \frac{c}{a} n^d \log_b n
\end{aligned}$$

Caso $a \neq b^d$:

$$\begin{aligned}
g(z) &= \frac{c}{b^d - a} \cdot \frac{1}{1 - b^d z} + \frac{\left(b^d - a\right) t_1 - c}{b^d - a} \cdot \frac{1}{1-az} \\
T(k) &= \frac{c}{b^d - a} \cdot b^{kd} + \frac{\left(b^d - a\right) t_1 - c}{b^d - a} \cdot a^k
\end{aligned}$$

Si $a > b^d$:

$$\begin{aligned}
T(k) &\sim \frac{\left(b^d - a\right) t_1 - c}{b^d - a} \cdot a^k \\
t(n) &\sim \frac{\left(b^d - a\right) t_1 - c}{b^d - a} \cdot n^{\log_b a}
\end{aligned}$$

Si $a < b^d$:

$$\begin{aligned}
T(k) &\sim \frac{c}{b^d - a} \cdot b^{kd} \\
t(n) &\sim \frac{c}{b^d - a} \cdot n^d
\end{aligned}$$

Resumiendo ("Teorema Maestro"): Al dividir un problema de tamaño nb en a problemas de tamaño n, que se resuelven recursivamente, donde el trabajo que se hace al dividir y luego combinar soluciones está dado por cn^d, resulta la recurrencia:

$$t(nb) = a\,t(n) + cn^d \qquad t(1) = t_1$$

cuya solución es:

$$t(n) \sim \begin{cases} \dfrac{c}{b^d - a} \cdot n^d & a < b^d \\ \dfrac{c}{a} n^d \log_b n & a = b^d \\ \dfrac{(b^d - a)\,t_1 - c}{a} \cdot n^{\log_n a} & a > b^d \end{cases}$$

Donde:

- nb: tamaño del problema original.
- a: cantidad de subproblemas originados luego de hacer la división.
- b: tamaño relativo de los subproblemas respecto al problema grande.
- c: constante determinada que nos dice lo que "nos cuesta" hacer la división.
- d: costo extra (volver a compaginar la solución completa)
- n: tamaño de los subproblemas originados luego de hacer la división.
- $t(n)$: trabajo necesario para resolver una instancia de tamaño n del problema en cuestión.

Nuestros ejemplos se resumen en el cuadro 1.

	a	b	c	d	$t(n)$
Mergesort	2	2		1	$\Theta\left(n \log n\right)$
Búsqueda binaria	1	2		0	$\Theta\left(\log n\right)$
Karatsuba	3	2		1	$\Theta\left(n^{\log_2 3}\right)$
Strassen	7	2		2	$\Theta\left(n^{\log_2 7}\right)$

Cuadro 1: Complejidad de nuestros ejemplos

Una variante de esto es el teorema de Akra-Bazzi, del que reportamos la variante de Leighton [3].

Teorema 1.1 (Akra-Bazzi)**.** Sea una recurrencia de la forma:

$$T(z) = g(z) + \sum_{1 \le k \le n} a_k T(b_k z + h_k(z)) \quad \text{para } z \ge z_0$$

donde z_0, a_k y b_k son constantes, sujeta a las siguientes condiciones:

- Hay suficientes casos base.
- Para todo k se cumplen $a_k > 0$ y $0 < b_k < 1$.
- Hay una constante c tal que $|g(z)| = O(z^c)$.
- Para todo k se cumple $|h_k(z)| = O(z/(\log z)^2)$.

Entonces, si p es tal que:

$$\sum_{1 \le k \le n} a_k b_k^p = 1$$

la solución a la recurrencia cumple:

$$T(z) = \Theta\left(z^p \left(1 + \int_1^z \frac{g(u)}{u^{p+1}}\,\mathrm{d}u\right)\right)$$

Frente a nuestro tratamiento tiene la ventaja de manejar divisiones desiguales (b_k diferentes), y explícitamente considera pequeñas perturbaciones en los términos, como lo son aplicar pisos o techos, a través de los $h_k(z)$. Diferencias con pisos y techos están acotados por una constante, mientras la cota del teorema permite que crezcan. Por ejemplo, la recurrencia correcta para el número de comparaciones en ordenamiento por intercalación es:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n - 1$$

El teorema de Akra-Bazzi es aplicable. La recurrencia es:

$$T(n) = T(n/2 + h_+(n)) + T(n/2 + h_-(n)) + n - 1$$

Acá $|h_\pm(n)| \le 1/2$, además $a_\pm = 1$ y $b_\pm = 1/2$. Estos cumplen las condiciones del teorema, de:

$$\sum_{1 \le k \le 2} a_k b_k^p = 1$$

resulta $p = 1$, y tenemos la cota:

$$T(z) = \Theta\left(z\left(1 + \int_1^z \frac{u-1}{u^2}\,\mathrm{d}u\right)\right) = \Theta(z \ln z + 1) = \Theta(z \log z)$$

Otro ejemplo son los árboles de búsqueda aleatorizados (*Randomized Search Trees*, ver por ejemplo Aragon y Seidel [1], Martínez y Roura [4] y Seidel y Aragon [5]) en uno de ellos de tamaño n una búsqueda toma tiempo aproximado:

$$T(n) = \frac{1}{4}\,T(n/4) + \frac{3}{4}\,T(3n/4) + 1$$

Nuevamente es aplicable el teorema 1.1, de:

$$\frac{1}{4}\left(\frac{1}{4}\right)^p + \frac{3}{4}\left(\frac{3}{4}\right)^p = 1$$

obtenemos $p = 0$, y por tanto la cota

$$T(z) = \Theta\left(z^0\left(1 + \int_1^z \frac{\mathrm{d}u}{u}\right)\right) = \Theta(\log z)$$

Una variante útil, pero bastante engorrosa, del teorema maestro es la que presenta Yap [7].

Referencias

[1] Cecilia R. Aragon and Raimund G. Seidel: *Randomized search trees.* In *Thirtieth Annual Symposium on Foundations of Computer Science*, pages 540–545, October - November 1989.

[2] A. Karatsuba and Yu. Ofman: *Multiplication of many-digital numbers by automatic computers.* Proceedings of the USSR Academy of Sciences, 145:293–294, 1962.

[3] Tom Leighton: *Notes on better master theorems for divide-and-conquer recurrences.* `http://citeseer.ist.psu.edu/252350.html`, 1996.

[4] Conrado Martínez and Salvador Roura: *Randomized binary search trees.* Journal of the ACM, 45(2):288–323, March 1998.

[5] Raimund G. Seidel and Cecilia A. Aragon: *Randomized search trees.* Algorithmica, 16(4/5):464–497, October 1996.

[6] Volker Strassen: *Gaussian elimination is not optimal.* Numerische Mathematik, 13(4):354–356, August 1969.

[7] Chee K. Yap: *A real elementary approach to the master recurrence and generalizations.* In Mitsunori Ogihara and Jun Tarui (editors): *Theory and Applications of Models of Computation: 8th Annual Conference*, volume 6648 of *Lecture Notes in Computer Science*, pages 14–26, Tokyo, Japan, May 2011. Springer.