

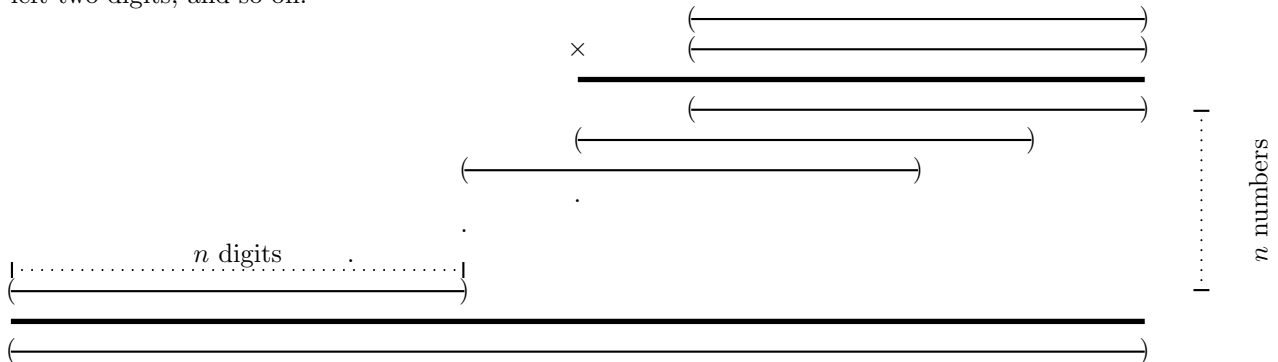
Lecture 17: November 2, 2009

CS 330 Discrete Structures
Fall Semester, 2009

1 Divide-and-conquer algorithms

1.1 Multiplying numbers

How do we multiply two n -digit numbers? First we multiply the first number by the rightmost digit of the second number. Then we multiply the first number by the second digit of the second number, and so on. Finally, we add the first result to the second result shifted left one digit, add that to the third result shifted left two digits, and so on.



Treating additions and multiplications of digits as our atomic operations, we can accomplish this in $\Theta(n^2)$ operations.

Let us consider a divide-and-conquer algorithm.¹ We divide each n -digit number into two $\frac{n}{2}$ -digit numbers:

$$\begin{array}{cc} \left(\begin{array}{c} \overbrace{\hspace{2cm}}^A \\ \underbrace{\hspace{1cm}}_{A_1} \end{array} \right) & \left(\begin{array}{c} \overbrace{\hspace{2cm}}^B \\ \underbrace{\hspace{1cm}}_{B_1} \end{array} \right) \\ \left(\begin{array}{c} \underbrace{\hspace{1cm}}_{A_0} \end{array} \right) & \left(\begin{array}{c} \underbrace{\hspace{1cm}}_{B_0} \end{array} \right) \end{array}$$

Since we are dividing A and B in half, we know that $A = 10^{\frac{n}{2}}A_1 + A_0$ and $B = 10^{\frac{n}{2}}B_1 + B_0$. So:

$$\begin{aligned} AB &= (10^{\frac{n}{2}}A_1 + A_0)(10^{\frac{n}{2}}B_1 + B_0) \\ &= 10^n A_1 B_1 + 10^{\frac{n}{2}}(A_1 B_0 + A_0 B_1) + A_0 B_0 \end{aligned}$$

This yields four multiplications of $\frac{n}{2}$ -digit numbers, as well as $\Theta(n)$ additions and shifts. This leads us to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 4T\left(\frac{n}{2}\right) + kn \end{aligned}$$

As before, let $n = 2^i$ and let $t_i = T(2^i)$. So $t_i = 4t_{i-1} + k2^i$, which is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 4)$. Thus $T(n) = cn + \hat{c}(n^2) = \Theta(n^2)$, which is no better than our original algorithm.

¹This is discussed in Rosen, page 475, example 4.

We can do something about this, however. Notice that $A_1B_0 + A_0B_1 = (A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0$. We can perform the calculation with only three multiplications. This leads to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3T\left(\frac{n}{2}\right) + kn \end{aligned}$$

Let $n = 2^i$ and let $t_i = T(2^i)$. So $t_i = 3t_{i-1} + k2^i$, which is annihilated by $(\mathbf{E} - 2)(\mathbf{E} - 3)$. Thus $T(n) = cn + \hat{c}n^{\lg 3} = \Theta(n^{\lg 3}) \approx \Theta(n^{1.57})$. This is a substantial improvement over our earlier divide-and-conquer algorithm as well as the naive algorithm.

1.2 Matrix multiplication and Strassen's algorithm

We define the product of two matrices A and B as follows, where A is an $m \times n$ matrix and B is an $n \times r$ matrix:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nr} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^n a_{1i}b_{i1} & \sum_{i=0}^n a_{1i}b_{i2} & \cdots & \sum_{i=0}^n a_{1i}b_{ir} \\ \sum_{i=0}^n a_{2i}b_{i1} & \sum_{i=0}^n a_{2i}b_{i2} & \cdots & \sum_{i=0}^n a_{2i}b_{ir} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=0}^n a_{mi}b_{i1} & \sum_{i=0}^n a_{mi}b_{i2} & \cdots & \sum_{i=0}^n a_{mi}b_{ir} \end{pmatrix}$$

That is, to find $(AB)_{ij}$, first locate the i th row of A and the j th column of B . Multiply the first element of the row by the first element of the column, the second element of the row by the second element of the column, and so on, and finally add these products.

Finding one element of AB thus requires $\Theta(n)$ time. Since there are mr elements in AB , multiplying A and B in this fashion requires $\Theta(mnr)$ time, or, if we consider the special case in which all matrices are $n \times n$, it requires $\Theta(n^3)$ time.

Let us consider a simple divide-and-conquer approach. We can divide A , B , and AB each into four $\frac{n}{2} \times \frac{n}{2}$ matrices:

$$\begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & g \\ f & h \end{pmatrix}$$

This corresponds to the four equations:

$$\begin{aligned} r &= ae + bf \\ s &= ag + bh \\ t &= ce + df \\ u &= cg + dh \end{aligned}$$

To find AB , then, we must multiply eight pairs of $\frac{n}{2} \times \frac{n}{2}$ matrices and add four pairs of matrices. This leads us to the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 8T\left(\frac{n}{2}\right) + \Theta(n^2) \end{aligned}$$

Solving this recurrence, unfortunately, yields $T(n) = \Theta(n^3)$, identical to the naive algorithm above.

As before, though, a divide-and-conquer approach allows us to take advantage of clever observations about the structure of matrices. In this case, the actual algorithm is quite complicated and is beyond the scope of

this lecture, but it will suffice to state that Strassen's algorithm requires only seven multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices rather than the eight multiplications we required earlier. Here's how it's done: Define the seven matrix (sub)products

$$\begin{aligned} P_1 &= a(f - h) \\ P_2 &= (a + b)h \\ P_3 &= (c + d)e \\ P_4 &= d(g - e) \\ P_5 &= (a + d)(e + h) \\ P_6 &= (b - d)(g + h) \\ P_7 &= (a - c)(e + f) \end{aligned}$$

Then

$$\begin{aligned} r &= -P_2 + P_4 + P_5 + P_6 \\ s &= P_1 + P_2 \\ t &= P_3 + P_4 \\ u &= P_1 - P_3 + P_5 - P_7 \end{aligned}$$

(For a suggestion of how this particular workable combination of sums/products might have been discovered, see Cormen, Leiserson, Rivest, and Stein, section 28.2.)

Thus we have the recurrence:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 7T\left(\frac{n}{2}\right) + \Theta(n^2) \end{aligned}$$

Solving this recurrence gives us better news: $T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})$. We have found a divide-and-conquer algorithm that is asymptotically faster than the naive algorithm.

On a less joyous note, the constant factor hidden in our analysis of Strassen's algorithm is large, so the naive algorithm is in fact faster on small ($n < 45$ or so) matrices. Faster algorithms than Strassen's have been found, the best one running in $O(n^{2.376})$ time, and the question of whether there are even faster algorithms is still open. (The largest lower bound known is $\Omega(n^2)$.)

1.3 Closest Pair of Points

See Example 12 on pages 479–482 of Rosen.