Divide-and-Conquer

(Matrix Multiplication, Large Integer Multiplication, Closest Pair)

Conventional Matrix Multiplication

- Brute-force algorithm

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

$$= \begin{pmatrix} a_{00} * b_{00} + a_{01} * b_{10} & a_{00} * b_{01} + a_{01} * b_{11} \\ a_{10} * b_{00} + a_{11} * b_{10} & a_{10} * b_{01} + a_{11} * b_{11} \end{pmatrix}$$

8 multiplications

4 additions

Efficiency class in general: Θ (n^3)

D&C Matrix Multiplication

Using Divide and Conquer the product of two matrices can be computed in general as follows:

$$\begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} * \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

$$= \begin{pmatrix} A_{00} * B_{00} + A_{01} * B_{10} & A_{00} * B_{01} + A_{01} * B_{11} \\ A_{10} * B_{00} + A_{11} * B_{10} & A_{10} * B_{01} + A_{11} * B_{11} \end{pmatrix}$$

8 multiplications

4 additions

Efficiency class in general: Θ (n^3)

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed in general as follows:

$$\left(\begin{array}{c|c} C_{00} & C_{01} \\ \hline C_{10} & C_{11} \end{array}\right) = \left(\begin{array}{c|c} A_{00} & A_{01} \\ \hline A_{10} & A_{11} \end{array}\right) * \left(\begin{array}{c|c} B_{00} & B_{01} \\ \hline B_{10} & B_{11} \end{array}\right)$$

$$= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

Formulas for Strassen's Algorithm

$M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$

$M_2 = (A_{10} + A_{11}) * B_{00}$

$M_3 = A_{00} * (B_{01} - B_{11})$

$M_4 = A_{11} * (B_{10} - B_{00})$

$M_5 = (A_{00} + A_{01}) * B_{11}$

$M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$

$M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

Strassen's Matrix Multiplication

- Strassen's algorithm for two 2x2 matrices (1969):

$$\begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} * \begin{pmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{pmatrix}$$

$$= \begin{pmatrix} m_1 + m_4 - m_5 + m_7 & m_3 + m_5 \\ m_2 + m_4 & m_1 + m_3 - m_2 + m_6 \end{pmatrix}$$

- $m_1 = (a_{00} + a_{11}) * (b_{00} + b_{11})$
- $m_2 = (a_{10} + a_{11}) * b_{00}$
- $m_3 = a_{00} * (b_{01} - b_{11})$
- $m_4 = a_{11} * (b_{10} - b_{00})$
- $m_5 = (a_{00} + a_{01}) * b_{11}$
- $m_6 = (a_{10} - a_{00}) * (b_{00} + b_{01})$
- $m_7 = (a_{01} - a_{11}) * (b_{10} + b_{11})$

7 multiplications

18 additions

Analysis of Strassen's Algorithm

If n is not a power of 2, matrices can be padded with zeros.

What if we count both multiplications and additions?

Number of multiplications:

$$M(n) = 7M(n/2), \quad M(1) = 1$$

Solution: $M(n) = 7^{\log_2 n} = n^{\log_2 7} \approx n^{2.807}$ vs. n^3 of brute-force alg or divide and conquer alg.

Algorithms with better asymptotic efficiency are known but they are even more complex and not used in practice.

Multiplication of Large Integers

Consider the problem of multiplying two (large) n-digit integers represented by arrays of their digits such as:

A = 12345678901357986429   B = 87654321284820912836

The grade-school algorithm:

$$
\begin{array}{rl}
 & a_1 \; a_2 \ldots \; a_n \\
 & \underline{b_1 \; b_2 \ldots \; b_n} \\
(d_{10}) & d_{11} d_{12} \ldots d_{1n} \\
(d_{20}) \; d_{21} d_{22} \ldots d_{2n} & \\
\ldots \ldots \ldots \ldots \ldots \ldots \ldots & \\
\underline{(d_{n0}) \; d_{n1} d_{n2} \ldots d_{nn}} &
\end{array}
$$

Efficiency: $\Theta(n^2)$ single-digit multiplications

First Divide-and-Conquer Algorithm

A small example: A * B where A = 2135 and B = 4014

$$A = (21 \cdot 10^2 + 35),\ \ B = (40 \cdot 10^2 + 14)$$

So, $A * B = (21 \cdot 10^2 + 35) * (40 \cdot 10^2 + 14)$

$$= 21 * 40 \cdot 10^4 + (21 * 14 + 35 * 40) \cdot 10^2 + 35 * 14$$

In general, if $A = A_1A_2$ and $B = B_1B_2$
(where A and B are n-digit, A_1, A_2, B_1, B_2 are $n/2$-digit numbers),

then, $A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

Recurrence for the number of one-digit multiplications $M(n)$:

$$M(n) = 4M(n/2), \quad M(1) = 1$$

Solution: $M(n) = n^2$

Second Divide-and-Conquer Algorithm

$A * B = A_1 * B_1 \cdot 10^n + (A_1 * B_2 + A_2 * B_1) \cdot 10^{n/2} + A_2 * B_2$

The idea is to decrease the number of multiplications from 4 to 3:

$$(A_1 + A_2) * (B_1 + B_2) = A_1 * B_1 + (A_1 * B_2 + A_2 * B_1) + A_2 * B_2$$

i.e., $(A_1 * B_2 + A_2 * B_1) = (A_1 + A_2) * (B_1 + B_2) - A_1 * B_1 - A_2 * B_2$, which requires only 3 multiplications at the expense of (4-1) extra add/sub.

Recurrence for the number of multiplications M(n):

$$M(n) = 3M(n/2), \quad M(1) = 1$$

Solution: $M(n) = 3^{\log_2 n} = n^{\log_2 3} \approx n^{1.585}$

What if we count both multiplications and additions?

Example of Large-Integer Multiplication

2135 * 4014

$= (21*10^2 + 35) * (40*10^2 + 14)$

$= (21*40)*10^4 + c1*10^2 + 35*14$

where $c1 = (21+35)*(40+14) - 21*40 - 35*14$

$21*40 = (2*10 + 1) * (4*10 + 0)$

$= (2*4)*10^2 + c2*10 + 1*0$

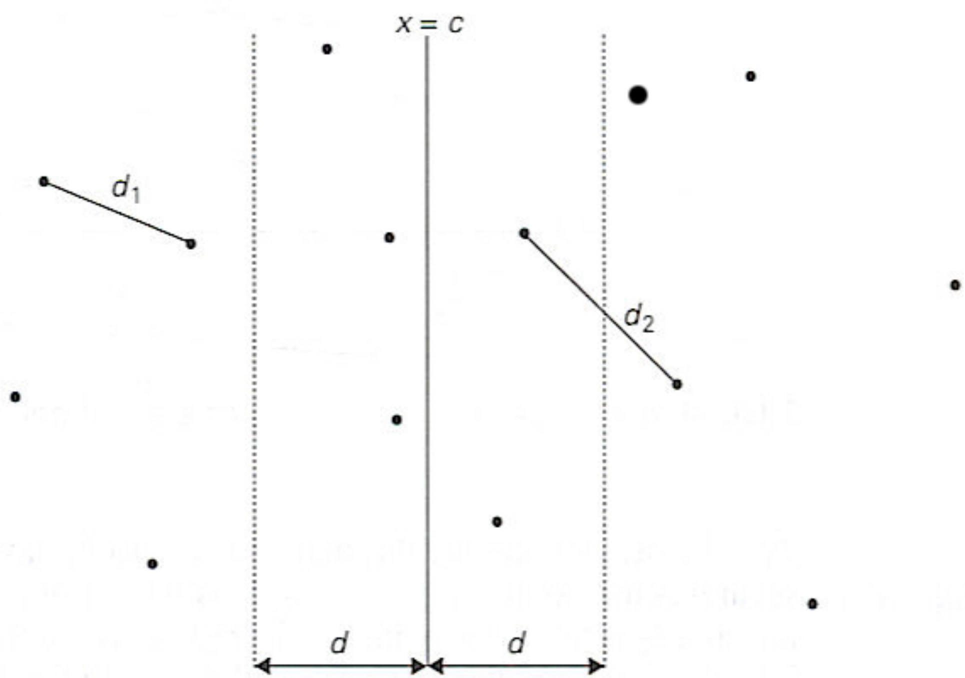
where $c2 = (2+1)*(4+0) - 2*4 - 1*0$, etc.

This process requires 9 digit multiplications as opposed to 16.

Closest-Pair Problem by Divide-and-Conquer

Step 0 Sort the points by x (list one) and then by y (list two).

Step 1 Divide the points given into two subsets S_1 and S_2 by a vertical line $x = c$ so that half the points lie to the left or on the line and half the points lie to the right or on the line.



Closest Pair by Divide-and-Conquer (cont.)

Step 2 Find recursively the closest pairs for the left and right subsets.

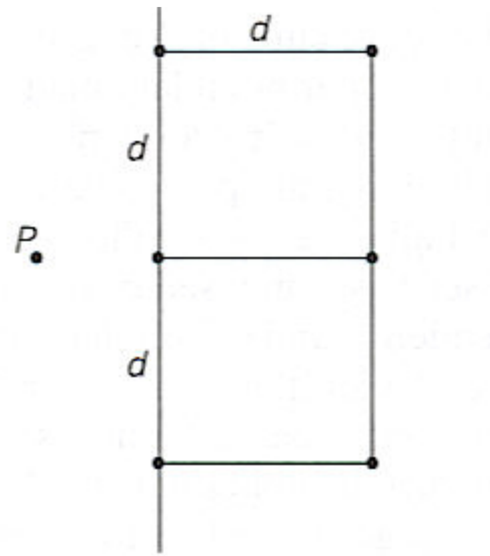
Step 3 Set $d = \min\{d_1, d_2\}$

We can limit our attention to the points in the symmetric vertical strip of width $2d$ as possible closest pair. Let C_1 and C_2 be the subsets of points in the left subset S_1 and of the right subset S_2, respectively, that lie in this vertical strip. The points in C_1 and C_2 are stored in increasing order of their y coordinates, taken from the second list.

Step 4 For every point $P(x,y)$ in C_1, we inspect points in C_2 that may be closer to P than d. There can be no more than 6 such points (because $d \le d_2$)!

Closest Pair by Divide-and-Conquer: Worst Case

The worst case scenario is depicted below:



Efficiency of the Closest-Pair Algorithm

Running time of the algorithm (without sorting) is:

$$T(n) = 2T(n/2) + M(n), \quad \text{where } M(n) \in \Theta(n)$$

By the Master Theorem (with $a = 2$, $b = 2$, $d = 1$)

$$T(n) \in \Theta(n \log n)$$

So the total time is $\Theta(n \log n)$.

Binary Tree Algorithms

Binary tree is a divide-and-conquer ready structure!

Ex. 1: Classic traversals (preorder, inorder, postorder)

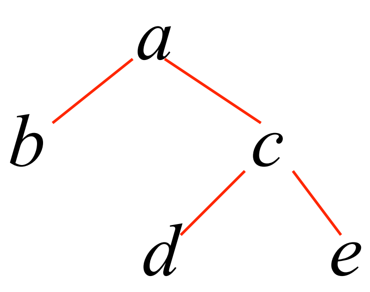
Algorithm *Inorder*(T)

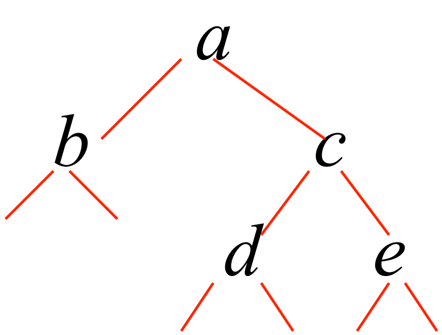
if $T \neq \varnothing$

$\quad$ *Inorder*(T_{left})

$\quad$ print(root of T)

$\quad$ *Inorder*(T_{right})

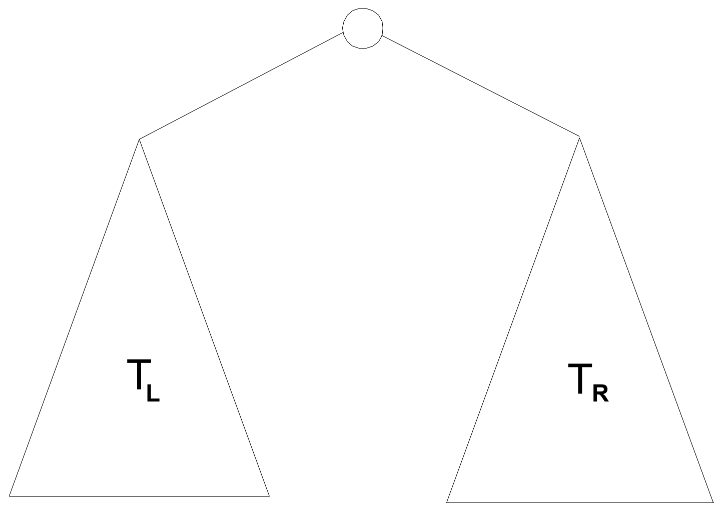




Efficiency: $\Theta(n)$. Why? Each node is visited/printed once.

Binary Tree Algorithms (cont.)

Ex. 2: Computing the height of a binary tree



$h(T) = \max\{h(T_L), h(T_R)\} + 1$ if $T \neq \varnothing$ and $h(\varnothing) = -1$

Efficiency: $\Theta(n)$. Why?