CSL 356 July 26, 2013

Computing Fibonacci seq efficiently

→ continuation

$$F_n = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$$

Computing the n^{th} power of a 2×2 matrix.

Similar to computing x^n for some number x

$$\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 a_2 + b_1 c_2 & a_1 b_2 + b_1 d_2 \\ c_1 a_2 + d_1 c_2 & c_1 b_2 + d_1 d_2 \end{bmatrix}$$

8 multipl. 4 additions

$O(1)$ multip and $O(1)$ additions

$$x^n = \begin{cases} \text{sqr}(x^{n/2}) & \text{if } n \text{ is even} \\ \text{sqr}(x^{\frac{n-1}{2}}) \cdot x & \text{if } n \text{ is odd} \end{cases}$$

$n = 0$, then

1

$\Rightarrow O(\log n)$ $\underbrace{\text{squaring + multipl.}}_{\text{multiplications}}$

$|x^n| = \underline{n \log x}$ bits

Computing A^n where A is a 2×2 matrix

$\rightarrow x^n$ " x is number

Suppose we start with 2 and square it repeatedly for n steps

$2, \; 2^2, \; (2^2)^2, \; ((2^2)^2)^2 \; \ldots \; \underset{n \text{ times}}{2^{2^n}}$

$|2^{2^n}| = \underline{2^n}$

Uniform model: operand sizes are ignored

Bit level complexity / logarithmic cost

$T_B(n)$: the # steps required to raise a number (a few bits) to power n

$$T_B(n) = T_B\left(\frac{n}{2}\right) + M(n)$$

cost of multiplying 2 n-bit nos. incl squaring

$$T(2) = O(1)$$

$$= T_B\left(\frac{n}{4}\right) + M\left(\frac{n}{2}\right) + M(n)$$

$$\vdots$$

$$= O\left(\sum_{i=1}^{\log n} M(2^i)\right) \qquad 2^{\log n} = n$$

$$M(k) = O(k^2)$$

$$O\left(n^2 + \left(\frac{n}{2}\right)^2 + \left(\frac{n}{4}\right)^2 + \cdots\right)$$

$= O(n^2)$ which also captures the cost of A^n, A is 2×2 matrix.

If the multiplication algorithm has the following property

$M(2i) > 2M(i)$, then the above recurrence has a soln that is dominated by the largest term, namely $M(n)$

For eg. if $M(n)$ is $O(n \log n)$, then Fibonacci no can be computed in $O(n \log n)$

$M(n)$ is $O(n^{1.5})$ then $\Rightarrow O(n^{1.5})$

We have "reduced" the complexity of computing F_n to $M(n)$

Can we multiply faster (than $O(n^2)$)?

$$\underbrace{x_n \cdots x_2}_{n \text{ bits}} \underbrace{x_1 \; x_0} \quad \times \quad \underbrace{y_n \, y_{n-1}}_{n \text{ bits}} \underbrace{\cdots \; y_0}$$

$$X = x_{n-1} 2^{n-1} + x_{n-2} \cdot 2^{n-2} + \cdots + x_0 \cdot 2^0$$

$$x_i \in \{0, 1\}$$

$$Y = y_{n-1} 2^{n-1} + y_{n-2} \cdot \quad + y_0 \cdot 2^0$$

Suppose n is a power of 2

$$X = X^1 \quad X^0$$

$$Y = Y^1 \quad Y^0$$

$$X = \left(x_{n-1} 2^{\frac{n}{2}} + \cdots x_{\frac{n}{2}}\right) \cdot 2^{n/2} + X^0$$

$$Y = \left(y_{n-2} 2^{n/2} + \cdot \cdot \; y_{n/2}\right) \cdot 2^{n/2} + Y_0$$

X^1, X^0, Y^1, Y^0 are all $\frac{n}{2}$ bit no

$$X \cdot Y = \left(X^1 \cdot 2^{n/2} + X^0\right)\left(Y^1 \cdot 2^{n/2} + Y^0\right)$$

$$= X^1 \cdot Y^1 \cdot 2^n + \left(X^1 Y^0 + Y^1 X^0\right) 2^{n/2} + X^0 \cdot Y^0$$

We can write a recurrence to capture this divide-and-conquer algorithm

$$M(n) = \cancel{4}^{\,3}\, M\left(\frac{n}{2}\right) + O(n)$$

multiplying n bit nos,

Shifting and adding n bit nos

$$M(n) = O(n^2) \rightarrow O(n^{\log_2 3})$$

$$\log_2 3 < 2$$

We can obtain all the terms by 3 $\frac{n}{2}$ bit multiplied + 4 additions

Best known mult algo. can multiply in $O(n \log n \cdot \log \log n)$

Schonage - Strassen : 1973