

Lecture 3: The Polynomial Multiplication Problem

A More General Divide-and-Conquer Approach

Divide: Divide a given problem into subproblems (ideally of approximately equal size).

No longer only TWO subproblems

Conquer: Solve each subproblem (directly or **recursively**), and

Combine: Combine the solutions of the subproblems into a global solution.

The Polynomial Multiplication Problem

another divide-and-conquer algorithm

Problem:

Given two polynomials of degree n

$$A(x) = a_0 + a_1x + \cdots + a_nx^n$$

$$B(x) = b_0 + b_1x + \cdots + b_nx^n,$$

compute the product $A(x)B(x)$.

Example:

$$A(x) = 1 + 2x + 3x^2$$

$$B(x) = 3 + 2x + 2x^2$$

$$A(x)B(x) = 3 + 8x + 15x^2 + 10x^3 + 6x^4$$

Question: How can we **efficiently** calculate the coefficients of $A(x)B(x)$?

Assume that the coefficients a_i and b_i are stored in arrays $A[0 \dots n]$ and $B[0 \dots n]$.

Cost of any algorithm is number of scalar multiplications and additions performed.

Convolutions

Let $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^m b_i x^i$.

Set $C(x) = \sum_{k=0}^{n+m} c_k x^k = A(x)B(x)$.

Then

$$c_k = \sum_{i=0}^k a_i b_{k-i}$$

for all $0 \leq k \leq m + n$.

Definition: The vector $(c_0, c_1, \dots, c_{m+n})$ is the **convolution** of the vectors (a_0, a_1, \dots, a_n) and (b_0, b_1, \dots, b_m) .

Calculating convolutions (and thus polynomial multiplication) is a major problem in digital signal processing.

The Direct (Brute Force) Approach

Let $A(x) = \sum_{i=0}^n a_i x^i$ and $B(x) = \sum_{i=0}^n b_i x^i$.

Set $C(x) = \sum_{k=0}^{2n} c_k x^k = A(x)B(x)$ with

$$c_k = \sum_{i=0}^k a_i b_{k-i}$$

for all $0 \leq k \leq 2n$.

The direct approach is to compute all c_k using the formula above. The total number of multiplications and additions needed are $\Theta(n^2)$ and $\Theta(n^2)$ respectively. Hence the complexity is $\Theta(n^2)$.

Questions: Can we do better?

Can we apply the divide-and-conquer approach to develop an algorithm?

The Divide-and-Conquer Approach

The Divide Step: Define

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1},$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor}.$$

Then $A(x) = A_0(x) + A_1(x)x^{\lfloor \frac{n}{2} \rfloor}$.

Similarly we define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x)x^{\lfloor \frac{n}{2} \rfloor}.$$

Then

$$A(x)B(x) = A_0(x)B_0(x) + A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} + A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} + A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}.$$

Remark: The original problem of size n is divided into 4 problems of input size $\frac{n}{2}$.

Example:

$$\begin{aligned}A(x) &= 2 + 5x + 3x^2 + x^3 - x^4 \\B(x) &= 1 + 2x + 2x^2 + 3x^3 + 6x^4 \\A(x)B(x) &= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 \\&\quad + 19x^6 + 3x^7 - 6x^8\end{aligned}$$

$$\begin{aligned}A_0(x) &= 2 + 5x, & A_1(x) &= 3 + x - x^2, \\A(x) &= A_0(x) + A_1(x)x^2 \\B_0(x) &= 1 + 2x, & B_1(x) &= 2 + 3x + 6x^2, \\B(x) &= B_0(x) + B_1(x)x^2\end{aligned}$$

$$\begin{aligned}A_0(x)B_0(x) &= 2 + 9x + 10x^2 \\A_1(x)B_1(x) &= 6 + 11x + 19x^2 + 3x^3 - 6x^4 \\A_0(x)B_1(x) &= 4 + 16x + 27x^2 + 30x^3 \\A_1(x)B_0(x) &= 3 + 7x + x^2 - 2x^3 \\A_0(x)B_1(x) + A_1(x)B_0(x) &= 7 + 23x + 28x^2 + 28x^3\end{aligned}$$

$$\begin{aligned}A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4 \\= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 + 19x^6 + 3x^7 - 6x^8\end{aligned}$$

The Divide-and-Conquer Approach

The Conquer Step: Solve the four subproblems, i.e., computing

$$A_0(x)B_0(x), \quad A_0(x)B_1(x), \\ A_1(x)B_0(x), \quad A_1(x)B_1(x)$$

by recursively calling the algorithm **4 times**.

The Divide-and-Conquer Approach

The Combining Step: Adding the following four polynomials

$$\begin{aligned} &A_0(x)B_0(x) \\ &+ A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} \\ &+ A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} \\ &+ A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}. \end{aligned}$$

takes $\Theta(n)$ operations. Why?

The First Divide-and-Conquer Algorithm

PolyMulti1($A(x)$, $B(x)$)

{

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + b_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$$U(x) = \text{PolyMulti1}(A_0(x), B_0(x));$$

$$V(x) = \text{PolyMulti1}(A_0(x), B_1(x));$$

$$W(x) = \text{PolyMulti1}(A_1(x), B_0(x));$$

$$Z(x) = \text{PolyMulti1}(A_1(x), B_1(x));$$

$$\text{return } \left(U(x) + [V(x) + W(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x)x^{2\lfloor \frac{n}{2} \rfloor} \right)$$

}

Running Time of the Algorithm

Assume n is a power of 2, $n = 2^h$. By substitution (expansion),

$$\begin{aligned}T(n) &= 4T\left(\frac{n}{2}\right) + cn \\&= 4\left[4T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\&= 4^2T\left(\frac{n}{2^2}\right) + (1+2)cn \\&= 4^2\left[4T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + (1+2)cn \\&= 4^3T\left(\frac{n}{2^3}\right) + (1+2+2^2)cn \\&\quad \vdots \\&= 4^i T\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j c n \quad (\text{induction}) \\&\quad \vdots \\&= 4^h T\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} 2^j c n \\&= n^2 T(1) + cn(n-1) \\&\quad (\text{since } n = 2^h \text{ and } \sum_{j=0}^{h-1} 2^j = 2^h - 1 = n - 1) \\&= \Theta(n^2).\end{aligned}$$

The same order as the brute force approach!

Comments on the Divide-and-Conquer Algorithm

Comments: The divide-and-conquer approach makes no essential improvement over the brute force approach!

Question: Why does this happen.

Question: Can you improve this divide-and-conquer algorithm?

Problem: Given 4 numbers

$$A_0, A_1, B_0, B_1$$

how many multiplications are needed to calculate the three values

$$A_0B_0, A_0B_1 + A_1B_0, A_1B_1?$$

This can obviously be done using 4 multiplications but there is a way of doing this using only the following 3:

$$Y = (A_0 + A_1)(B_0 + B_1)$$

$$U = A_0B_0$$

$$Z = A_1B_1$$

U and Z are what we originally wanted and

$$A_0B_1 + A_1B_0 = Y - U - Z.$$

Improving the Divide-and-Conquer Algorithm

Define

$$Y(x) = (A_0(x) + A_1(x)) \times (B_0(x) + B_1(x))$$

$$U(x) = A_0(x)B_0(x)$$

$$Z(x) = A_1(x)B_1(x)$$

Then

$$Y(x) - U(x) - Z(x) = A_0(x)B_1(x) + A_1(x)B_0(x).$$

Hence $A(x)B(x)$ is equal to

$$U(x) + [Y(x) - U(x) - Z(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x) \times x^{2\lfloor \frac{n}{2} \rfloor}$$

Conclusion: You need to call the multiplication procedure **3**, rather than **4** times.

The Second Divide-and-Conquer Algorithm

PolyMulti2($A(x)$, $B(x)$)

{

$$A_0(x) = a_0 + a_1x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1}x + \cdots + b_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$$Y(x) = \text{PolyMulti2}(A_0(x) + A_1(x), B_0(x) + B_1(x))$$

$$U(x) = \text{PolyMulti2}(A_0(x), B_0(x));$$

$$Z(x) = \text{PolyMulti2}(A_1(x), B_1(x));$$

$$\text{return } \left(U(x) + [Y(x) - U(x) - Z(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x)x^{2\lfloor \frac{n}{2} \rfloor} \right);$$

}

Running Time of the Modified Algorithm

Assume $n = 2^h$. Let $\lg x$ denote $\log_2 x$.
By the substitution method,

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{2}\right) + cn \\ &= 3\left[3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\ &= 3^2T\left(\frac{n}{2^2}\right) + \left(1 + \frac{3}{2}\right)cn \\ &= 3^2\left[3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + \left(1 + \frac{3}{2}\right)cn \\ &= 3^3T\left(\frac{n}{2^3}\right) + \left(1 + \frac{3}{2} + \left[\frac{3}{2}\right]^2\right)cn \\ &\quad \vdots \\ &= 3^hT\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} \left[\frac{3}{2}\right]^j cn. \end{aligned}$$

We have

$$3^h = (2^{\lg 3})^h = 2^{h \lg 3} = (2^h)^{\lg 3} = n^{\lg 3} \approx n^{1.585},$$

and

$$\sum_{j=0}^{h-1} \left[\frac{3}{2}\right]^j = \frac{(3/2)^h - 1}{3/2 - 1} = 2 \cdot \frac{3^h}{2^h} - 2 = 2n^{\lg 3 - 1} - 2.$$

Hence

$$T(n) = \Theta(n^{\lg 3}T(1) + 2cn^{\lg 3}) = \Theta(n^{\lg 3}).$$

Comments

- The divide-and-conquer approach doesn't always give you the best solution.
Our original D-A-C algorithm was just as bad as brute force.
- There is actually an $O(n \log n)$ solution to the polynomial multiplication problem.
It involves using the *Fast Fourier Transform* algorithm as a subroutine.
The FFT is another classic D-A-C algorithm (Chapt 30 in CLRS gives details).
- The idea of using 3 multiplications instead of 4 is used in large-integer multiplications.
A similar idea is the basis of the classic [Strassen matrix multiplication algorithm](#) (CLRS, Chapter 28).