Polynomial Operations

Course: Introduction to Programming and Data Structures

Laltu Sardar

Institute for Advancing Intelligence (IAI),
TCG Centres for Research and Education in Science and Technology (TCG Crest)

October 30, 2023

Polynomial Operations

Topic to be covered

- Representation
- Computing a polynomial
- Addition
- Subtraction
- Multiplication
- Division

We will discuss polynomial of the form $P(x) = \sum_{i=0}^{n} a_i x^i$, i.e., polynomials with one varible.

Representation of Polynomials

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

Different ways

How to store a polynomial?

tcg crest
Inventing Harmonious Future

Representation of Polynomials

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

Different ways

How to store a polynomial?

1. Array: Useful when most of the coefficients are present
2. Linked List: Useful when very few coefficients are present
3. Any disadvantage?
4. Which is better

How to compute a polynomial

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

How many multiplication and additions are required? Can We reduce multiplication further.

tcg crest
Inventing Harmonious Future

How to compute a polynomial

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

How many multiplication and additions are required?

How to compute a polynomial

$$P(x) = \sum_{i=0}^{n} a_i x^i$$

How many multiplication and additions are required?

Can We reduce # multiplications further?

tcg crest
Inventing Harmonious Future

Adding two polynomials

What will be the algorithm?

Adding two polynomials

What will be the algorithm?

What happen to the degree of new polynomial?

Adding two polynomials

What will be the algorithm?

What happen to the degree of new polynomial?

Problem of over computation. Solution?

Adding two polynomials

What will be the algorithm?

What happen to the degree of new polynomial?

Problem of over computation. Solution?

Keep the degree stored.
Structure is required.

Division of a polynomial with another

Consider two polynomials:

$$f(x) = \sum_{i=0}^{n} a_i x^i,\ g(x) = \sum_{i=0}^{m} b_i x^i$$

Multiplication of two polynomials

Consider two polynomials:

$$f(x) = \sum_{i=0}^{n} a_i x^i,\ g(x) = \sum_{i=0}^{m} b_i x^i$$

Divide and Conquer: Polynomial Multiplication

Version of October 7, 2014

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_m x^m \end{aligned}$$

Compute the product $A(x)B(x)$

Example

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

- Assume that the coefficients a_i and b_i are stored in arrays $A[0 \ldots n]$ and $B[0 \ldots m]$
- Cost: number of scalar multiplications and additions

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^{n} a_i x^i$
- $B(x) = \sum_{i=0}^{m} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

$$c_k = \sum_{0 \le i \le n,\ 0 \le j \le m,\ i+j=k} a_i b_j \quad \text{for all } 0 \le k \le m+n$$

Definition

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

While polynomial multiplication is interesting, real goal is to calculate convolutions. *Major* subroutine in digital signal processing

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \le i,j \le n, i+j=k} a_i b_j, \quad \text{for all } 0 \le k \le 2n$$

Direct approach: Compute all c_k's using the formula above

- Total number of multiplications: $\Theta(n^2)$
- Total number of additions: $\Theta(n^2)$
- Complexity: $\Theta(n^2)$

Divide-and-Conquer: Divide

Assume n is a power of 2
Define

$$
\begin{aligned}
A_0(x) &= a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1} \\
A_1(x) &= a_{\frac{n}{2}} + a_{\frac{n}{2}+1} x + \cdots + a_n x^{\frac{n}{2}} \\
A(x) &= A_0(x) + A_1(x) x^{\frac{n}{2}}
\end{aligned}
$$

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

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

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

The original problem (of size n) is divided into
4 problems of input size $n/2$

Example

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

$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2, A(x) = A_0(x) + A_1(x)x^2$$

$$B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2, B(x) = B_0(x) + B_1(x)x^2$$

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

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

Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- compute

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

  by recursively calling the algorithm 4 times

Combine

- adding the following four polynomials

$$A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n$$

- takes $O(n)$ operations (Why?)

The First Divide-and-Conquer Algorithm

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

begin

$A_0(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;

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

$B_0(x) = b_0 + b_1 x + \cdots + b_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$;

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

$U(x) = PolyMulti1(A_0(x), B_0(x))$;

$V(x) = PolyMulti1(A_0(x), B_1(x))$;

$W(x) = PolyMulti1(A_1(x), B_0(x))$;

$Z(x) = PolyMulti1(A_1(x), B_1(x))$;

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

end

Analysis of Running Time

Assume that n is a power of 2

$$T(n) = \begin{cases} 4T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^2).$$

Same order as the brute force approach!
No improvement!

Objective and Outline

Outline:

- Introduction
- The polynomial multiplication problem
- An $O(n^2)$ brute force algorithm
- An $O(n^2)$ first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Two Observations

Observation 1:

We said that we need the 4 terms:

$$A_0 B_0, \; A_0 B_1, \; A_1 B_0, \; A_1 B_.$$

What we really *need are the 3 terms:*

$$A_0 B_0, \; A_0 B_1 + A_1 B_0, \; A_1 B_1!$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0 B_0 \\ Z &= A_1 B_1 \end{aligned}$$

- *We need U and Z and*
- $A_0 B_1 + A_1 B_0 = Y - U - Z$

The Second Divide-and-Conquer Algorithm

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

begin

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

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

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

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

$Y(x) = PolyMulti2(A_0(x) + A_1(x), B_0(x) + B_1(x))$;

$U(x) = PolyMulti2(A_0(x), B_0(x))$;

$Z(x) = PolyMulti2(A_1(x), B_1(x))$;

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

end

Running Time of the Modified Algorithm

$$T(n) = \begin{cases} 3T(n/2) + n, & \text{if } n > 1, \\ 1, & \text{if } n = 1. \end{cases}$$

By the Master Theorem for recurrences

$$T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.58\ldots}).$$

Much better than previous $\Theta(n^2)$ algorithms!

Remarks

- This algorithm can also be used for (long) integer multiplication
  - Really designed by Karatsuba (1960, 1962) for that purpose.
  - Response to conjecture by Kolmogorov, founder of modern probability, that this would require $\Theta(n^2)$.
- Similar to technique deeloped by Strassen a few years later to multiply 2 $n \times n$ matrices in $O(n^{\log_2 7})$ operations, instead of the $\Theta(n^3)$ that a straightforward algorithm would use.
- Takeaway from this lesson is that divide-and-conquer doesn't always give you faster algorithm. Sometimes, you need to be more clever.
- Coming up. An $O(n \log n)$ solution to the polynomial multiplication problem
  - It involves strange recasting of the problem and solution using the Fast Fourier Transform algorithm as a subroutine
  - The FFT is another classic D & C algorithm that we will learn soon.