CS345 : Design and Analysis of Algorithms Spring 2014

Lecture 3: Multiplication of two Polynomials

In this lecture, we study another application of divide and conquer paradigm. This deals with efficient algorithm for multiplying two polynomials.

Let $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ where a_i's are real numbers denotes a polynomial of degree less than n. Similarly let $B(x) = b_0 + b + 1x + b + 2x^2 + \ldots b_{n-1}x^{n-1}$ denote another polynomial of degree less than n. Aim is to compute a polynomial $C(x) = A(x) \cdot B(x)$. Note that $C(x) = c_0 + c_1x + \ldots c_{2n-2}x^{2n-2}$ is a polynomial of degree less than $2n$ (to be precise, less than $2n-1$) where

$$c_i = a_0b_i + a_1b_{i-1} + \ldots + a_ib_0 = \sum_{0 \le j \le i} a_jb_{i-j}$$

It can be observed that computing c_i requires $O(i)$ arithmetic operations. Hence we can multiply two polynomials of degree less than n in $O(n^2)$. We shall design an $O(n \log n)$ time algorithm for this problem.

This problem may appear just as a theoretical problem at first glance with limited applications. But that is not true. After sorting and searching, this algorithm is arguably the most used algorithm in practice. It is used in algorithms related to signal processing (Discrete Fourier Transform). It also played a crucial role in designing $O(n \log n)$ time algorithm for multiplying two n-bit numbers (try to see some connection between the two problems).

Remark: This algorithm was published by Cooley and Tukey in 1965, and since then, this result has remained one of the results with maximum citation till date. Interestingly, it was found that this algorithm was known in slightly different form and context by the famouse mathematician Gauss.

The starting point of this solution is an alternate representation of a polynomial.

(point,value) representation

A polynomial $A(x)$ of degree less than n can be represented uniquely by expressing the coefficients $a_0, a_1, \ldots, a_{n-1}$. This is called the coefficient representation of $A(x)$. Interestingly, there is another way to represent $A(x)$ uniquely. Can you guess ?

Consider a linear polynomial, e.g., $a_0 + a_1x$. If you plot it, it is a line. We know that any line can be expressed uniquely also by mentioning the y-coordinates of its plot for any two distinct x-values. Likewise, cosider any quadratic polynomial, e.g., $a_0 + a_1x + a_2x^2$. We can express it uniquely by mentioning the y-coordinates of its plot for any three distinct x-values. The following theorem generalizes these observations.

Theorem 1. *Let $(r_0, t_0), \ldots, (r_{n-1}, t_{n-1})$ be any n pairs of real numbers with all distinct r_i's. There exists a unique polynomial $A(x)$ of degree less than n such that $A(r_i) = t_i$ for all i.*

Homework 1: Prove Theorem 1. It has a simple proof based on elementary theory of matrices.

It follows from Theorem 1 that we can express a polynomial $A(x)$ of degree less than n uniquely by selecting any n *distinct* points $r_0, \ldots, r_{n-1}$ and finding the values of the polynomial at those points. This representation $\{(r_0, A(r_0)), \ldots, (r_{n-1}, A(r_{n-1}))\}$ is called point value representation of $A(x)$.

The advantage of the (point,value) representation lies in the answer of the following question which you should try to answer.

Question 1: Let $r_0, \ldots, r_{m-1}$ be m distinct real numbers and $A(x)$ and $B(x)$ are two polynomials of degree less than n. Let $\{(r_0, A(r_0), \ldots, (r_{m-1}, A(r_{m-1}))\}$ and $\{(r_0, A(r_0), \ldots, (r_{m-1}, A(r_{m-1}))\}$ be their (point,value) representation. If $C(x) = A(x) \cdot B(x)$, then

1. What is the time complexity of computing $(r_0, C(r_0)), \ldots, (r_{m-1}, C(r_{m-1}))$?
2. What should be the value of m so that $(r_0, C(r_0)), \ldots, (r_{m-1}, C(r_{m-1}))$ is a (point,value) representation of $C(x)$?

Answer:

1. $O(m)$
2. at least $2n - 1$.

A natural way to exploit the above fact in our pursuit of efficient algorithm for multiplying polynomials given in coefficient representation is depicted in Figure 1. The $O(n \log n)$ time algorithm will consist of three steps as shown in the figure. Think over it for a while before moving ahead.

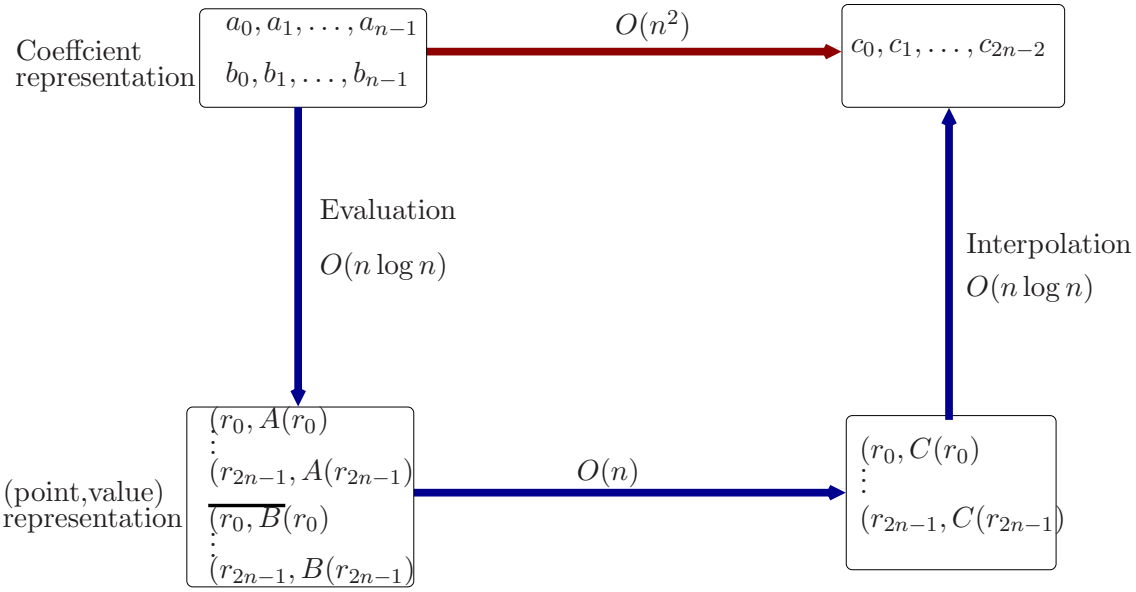

Figure 1: The road map of the $O(n \log n)$ algorithm for polynomial multiplication

So we need an $O(n \log n)$ time algorithm for evaluating a given polynomial $A(x)$ of degree less than n on $2n$ distinct points. We discuss this algorithm in the following section. Another beautiful idea, this time from complex numbers, awaits us there. Interesting, Interpolation, the last step in Figure 1, will turn out to be an Evaluation on certain points. Think over it. We shall cover it in the next class.

Evaluation

First without loss of generality assume n is power of 2. We shall see the reason underlying this assumption soon. We shall use divide and conquer technique. For this purose, let us rearrange the even and odd terms of polynomial $A(x)$ as follows.

$$\begin{aligned} A(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} \\ &= \left(a_0 + a_2x^2 + a_4x^4 + \cdots + a_{n-2}x^{n-2}\right) + \left(a_1x + a_3x^3 + a_5x^5 + \cdots + a_{n-1}x^{n-1}\right) \\ &= \left(a_0 + a_2x^2 + a_4x^4 + \cdots + a_{n-2}x^{n-2}\right) + x\left(a_1 + a_3x^2 + a_5x^4 + \cdots + a_{n-1}x^{n-2}\right) \end{aligned}$$

Let us define two polynomials A_{even} and A_{odd}.

$$A_{even}(x) = a_0 + a_2x + a_4x^2 + \ldots + a_{n-2}x^{(n-2)/2}$$

$$A_{odd}(x) = a_1 + a_3x + a_5x^2 + \cdots + a_{n-1}x^{(n-2)/2}$$

Observe that degree of A_{even} and A_{odd} is less than $n/2$.

Question 1: What is relation among A, A_{even}, and A_{odd} ? How does it help us design divide and conquer algorithm ?
Answer:

$$A(x) = A_{even}(x^2) + xA_{odd}(x^2) \tag{1}$$

This suggests the following divide and conquer approach: In order to evaluate polynomial $A(x)$ of degree $< n$ on $2n$ points, do the following.

1. Recursively evaluate two polynomials of degree $< n/2$ on $2n$ points.
2. Use Equation 1 to evaluate $A(x)$ on $2n$ points. (this takes $O(n)$ time).

Does it really work ? Unfortunateky No. Can you see why ? Pause before proceeding further.

Flaw in the above approach: Visualize the recursion tree associated with the approach. As we go down, the degree of polynomial decreases by a factor of 2, but the number of points on whichit has to be evaluated does not reduce. So towards the leaf level, there will be $\Theta(n)$ polynomials each to be evaluated at $2n$ different points. This itself will take $\Theta(n^2)$ time.

Fixing the flaw: Notice that we have full freedom to select $2n$ distinct points for evaluation. So here is a natural question to fix the above approach ?

Does there exist a set R of $2n$ distinct points $\{r_0, \ldots, r_{2n-1}\}$ such that the set of squares of elements of S is at most n ?

What if we define R by n distinct positive points and their negative counterparts? For example $\{1, 5, 19, -1, -5, -19\}$? This won't work.

Homework 2: What is the problem with the above fix ?
hint: It works for 1st level from top. Try to go beyond this level ...

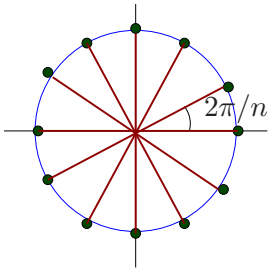

Figure 2: nth roots of unity split the unit circle into n equal parts obtained by angles which are multiple of $2\pi/n$.

The magic of complex numbers

We shall use our elementary knowledge of complex numbers here. In particular, we shall use some nice properties of nth roots of unity. Figure 2 demonstrates the nth roots of unity.

Let S_n denote the set of nth roots of unity. What can we say about the set obtained by squaring each element of S_n. Figure 3 should give you the right direction.

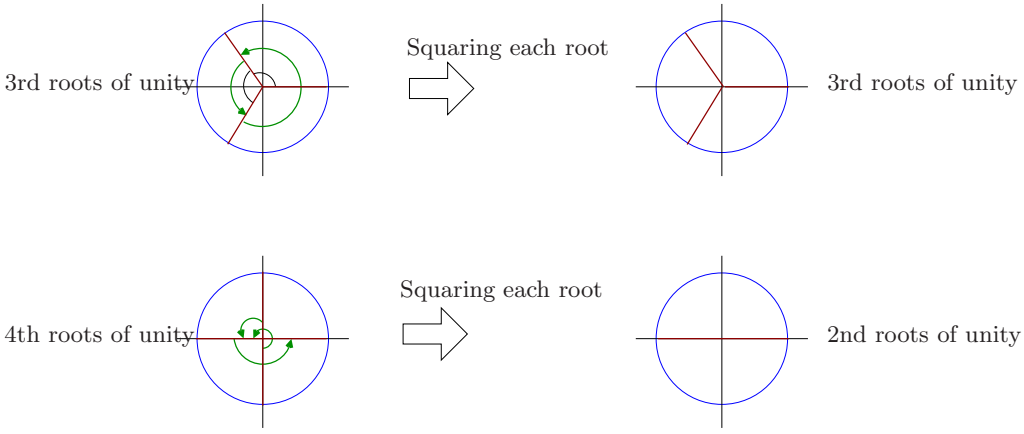

Figure 3: The result of aquaring 3rd units of unity 4th roots of unity.

Realize the following fact from Figure 3.

1. If n is odd, then squaring S_n produces S_n.
2. If n is even, then squaring S_n produces $S_{n/2}$.

The fact (2) plays the key role in our algorithm (Think over it).

So the desired set R of points to be used for the Evaluation step will consist of $2n$th roots of unity. This completes the $O(n \log n)$ time algorithm for the Evaluation step.
Homework 3: Why did we assume n to be power of 2 in the beginning ?
Homework 4: Try to write pseudocode for the Evaluation step.
Homework 5: Interpolation can be seen as Evaluation. Can you see how? We shall show it in the next class.

A small tutorial of complex numbers

In case you forgot the knowledge of complex numbers, here is a brief tutorial on complex number.

A complex number $a + bi$ has a real part a and imaginary part bi where a, b are real numbers and $i = \sqrt{-1}$. Recall that real numbers can be better visualized as points lying on a line. Similarly, it turns out that complex numbers can be better visualized as points or position vectors in x-y plane. For example $a + bi$ corresponds to vector from origin to point (a, b) in this complex plane. This visualization leads to many simple and nice interpretation of operations on complex numbers. For example, addition of complex number is similar to vector addition.

A point in the plane can also be expressed by polar coordianates r and θ instead of cartesian coordinates: Here r is the distance of point from origin and θ the angle that the line joining origin to it makes with the positive x-axis. In terms of complex number, r is called magnitude of the complex number ($r = \sqrt{a^2 + b^2}$) and θ is called argument. A complex number with magnitude r and argument θ represents the complex number $r\cos\theta + ir\sin\theta$. Using Euler's formula, it is $re^{i\theta}$. This leads to the following insight into the multiplication of complex numbers.

Principle of multiplication of complex numbers: When we multiply two complex numbers, their magnitudes gets multiplied and the arguments gets added. In precise words, it can be explained as follows. Let $y = r_1 e^{i\theta_1}$, that is, y is a complex number with magnitude r_1 and argument θ_1. Let $z = r_2 e^{i\theta_2}$, that is, z is a complex number with magnitude r_2 and argument θ_2. The number obtained by multipliying y and z is $r_1 r_2 e^{i(\theta_1+\theta_2)}$.

The nth roots of unity

A number x is called nth root of unity if $x^n = 1$. It follows from the multiplication principle of complex number that magnitude of x must be 1 and the argument must be a multiple of $2\pi/n$. Figure 2 depicts these roots in a nice way.

Squaring nth roots of unity

Using just the principle of multiplication of complex numbers stated above, it can be observed that square of any of the nth roots of unity will also be one of the nth root of unity. If n is odd, then we shall again get all nth roots of unity by squaring. Interestingly, if n is even, we shall only get a proper subset of nth roots of unity. This subset will actually be $n/2$th roots of unity. See Figure 2 for a better understanding.