CS 170: Efficient Algorithms and Intractable Problems

Note 3 - Fast Fourier Transform

Vishnu Iyer January 31, 2019

Polynomial arithmetic is an important problem with applications that arise in many fields. One might expect this function to be important in computer algebra systems and other mathematical tools, but similar concepts arise in areas such as signal processing as well. In this note, we will explore a divide-and-conquer technique to evaluate and interpolate polynomials.

1 Complex Numbers

You may be familiar with complex numbers from previous math classes. In this section, we review some of their main properties and set up results that will be useful to us later.

The section is included for completeness and readers with a strong background in complex numbers and polynomials need only read over the main results presented. Throughout the section, we will denote the set of real numbers as $\mathbb{R}$ and the set of complex numbers as $\mathbb{C}$.

1.1 Basics

A complex number z is a number of the form $a + bi$ where a and b are real numbers and $i^2 = -1$. The form $z = a + bi$ is known as the *rectangular form* of the complex number. This is because we can plot a complex number on what is known as the *complex plane*. The x coordinate is a and the y coordinate is b. Similar to rectangular and polar coordinates, we can also define a *polar form* $z = re^{i\theta}$ where $r, \theta \in \mathbb{R}$, $r > 0$, and $0 \leq \theta < 2\pi$. A representation of a complex number on the complex plane is shown in Figure 1 To convert between forms, we look at Euler's theorem.

Theorem 1.1 (Euler's Theorem)**.**

$$e^{i\theta} = \cos\theta + i\sin\theta$$

Proof. We consider the Taylor series for our functions.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!!}, \qquad \cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)!!}$$

The theorem follows from computing each term of the series $e^{i\theta}$ and $\cos\theta + i\sin\theta$. $\square$

Therefore, if r is such that $r^2 = a^2+b^2$ and θ is such that $\tan\theta = \frac{b}{a}$, we have that $a+bi = re^{i\theta}$. We call r the *magnitude* and θ the *angle*. The main reason we consider this form is to understand the process of multiplying complex numbers. The polar forms show that the product of $z_1 = r_1e^{i\theta_1}$ and $z_2 = r_2e^{i\theta_2}$ is $r_1r_2e^{i(\theta_1+\theta_2)}$. That is, *the magnitudes multiply and the angles add.* In this sense, multiplication by a complex number is equivalent to scaling by the magnitude and rotating counterclockwise by the angle.

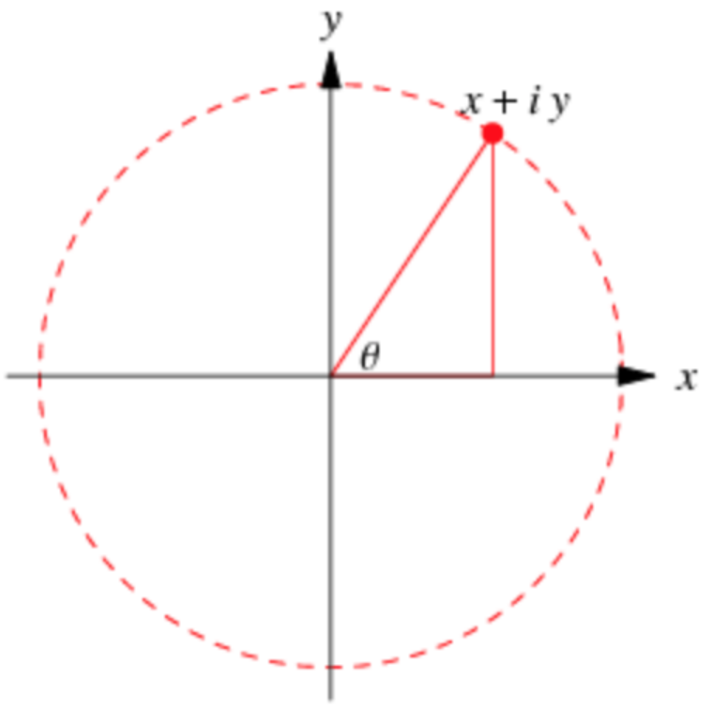

Figure 1: A complex number with rectangular form $x+iy$, magnitude $\sqrt{x^2+y^2}$, and angle θ on the complex plane. Courtesy of Wolfram Mathworld.

1.2 Roots of Unity

Among complex numbers, the *roots of unity* are of particular interest.

Definition 1.2 (Roots of Unity)**.** The nth roots of unity are complex numbers such that $z^n = 1$.

For every number n, there are exactly n distinct roots of unity. We know this from the Fundamental Theorem of Algebra.

Theorem 1.3 (Fundamental Theorem of Algebra)**.** For any polynomial $p(x)$ of degree n with complex coefficients, the equation $p(x) = 0$ has exactly n (not necessarily distinct) complex solutions.

Now we show how the roots

Proposition 1.4. The nth roots of unity are of the form $e^{\frac{2\pi i}{n}k}$, where k is an integer.

Proof. If $z = e^{\frac{2\pi i}{n}k}$, we can see that $z^n = e^{2\pi ik} = \cos(2\pi k) + i\sin(2\pi k) = 1$. Furthermore, for a choice of $k = 0, 1, ..., n-1$, $e^{\frac{2\pi i}{n}k}$ is distinct. □

This brings us to an important point. The roots of unity are, graphically, distributed evenly around the unit circularly. Since $e^{2\pi i} = 1$, we see that if $k_1 = k_2 + n$, then $e^{\frac{2\pi i}{n}k_2} = e^{\frac{2\pi i}{n}k_1}$.

Furthermore, since $e^{\pi i} = 1$, we have that $e^{i(\theta+\pi)} = -e^{i\theta}$. In the context of roots of unity, this means that, for an even number n, if $k_1 = k_2 + \frac{n}{2}$, then $e^{\frac{2\pi i}{n}k_2} = -e^{\frac{2\pi i}{n}k_1}$ and if $z_1 = e^{\frac{2\pi i}{n}k_1}$ and $z_2 = e^{\frac{2\pi i}{n}k_2}$, then $z_1^2 = z_2^2$. This in particular will be useful to us in our divide-and-conquer approach to the Fourier Transform. We summarize our results from this section below.

Properties of Complex Numbers
Let $z_1 = e^{\frac{2\pi i}{n}k_1}$ and $z_2 = e^{\frac{2\pi i}{n}k_2}$.

1. $e^{i\theta} = \cos\theta + i\sin\theta$
2. Multiplication of a complex number z by a complex number $re^{i\theta}$ dilates the magnitude of z by a factor of r and rotates its angle counterclockwise by θ.
3. The nth roots of unity are of the form $e^{\frac{2\pi i}{n}k}$ for any integer k.
4. If $k_2 = k_1 + n$, then $z_1 = z_2$.
5. If n is even and $k_2 = k_1 + \frac{n}{2}$, then $z_1 = -z_2$ and $z_1^2 = z_2^2$.

2 The Algorithm

A polynomial of degree at most n is uniquely determined by either its $n+1$ coefficients (called the coefficient representation) or its value at *any* $n+1$ distinct points (called the value representation). This is significant because given $n+1$ ordered pairs of points, there is *exactly* one polynomial of degree n or less that passes through these points.

Example 2.1. Consider the polynomial $p(x) = x^2 - 2x + 4$. Clearly it is uniquely described by the coefficients $a_2 = 1$, $a_1 = -2$, and $a_0 = 4$. However, since the polynomial is of degree 2, we can also describe the polynomial by the points $(0,4), (1,3), (2,4)$ or the points $(0,4), (i, 3-2i), (-i, 3+2i)$. Any set of 3 distinct points that the polynomial passes through is enough to uniquely identify it amongst degree 2 and lower polynomials. That is, *it is the only one* that passes through any set of 3 points that it passes through. Note in particular that the number of coefficients is equal to the number of points required; this is not a coincidence.

Suppose we want to design an algorithm that can convert between these representations. In particular, we will first consider the task of going from the coefficient representation to the value representation, a process known as *evaluation.* Note that the naive method of evaluation takes time $\Theta(n^2)$.

Our analysis will begin with the following fact about polynomials: any polynomial $p(x)$ can be written as the sum of an even polynomial $p_1(x)$ and an odd polynomial $p_2(x)$ (recall that an even function is one such that $f(-x) = f(x)$ and and odd function is one such that $f(-x) = -f(x)$). This is because we can collect the even degree terms in $p_1(x)$ and the odd degree terms in $p_2(x)$.

Example 2.2. We can break up the polynomial $p(x) = 5x^5 - 2x^4 + 4x^3 + 7x^2 - 6x - 6$ as follows:

$$p_1(x) = -2x^4 + 7x^2 - 6$$

$$p_2(x) = 5x^5 + 4x^3 - 6x$$

Notice that $p_1(x)$ can be written as some polynomial $p_e(x^2)$ and $p_2(x)$ can be written as some polynomial $xp_o(x^2)$.

Example 2.3. From our previous example, we can write

$$p_e(x) = -2x^2 + 7x - 6$$

$$p_o(x) = 5x^2 + 4x - 6$$

Which gives us $p_1(x) = p_e(x^2)$ and $p_2(x) = p_o(x^2)$. Ultimately, $p(x) = p_e(x^2) + xp_o(x^2)$.

Now, if p is of degree d then p_e and p_o are of degree at most $d/2$. Since it is obviously faster to evaluate smaller polynomials, we can use this as the basis for our algorithm.

Another key idea is to reuse computations. Namely, if we know $p_e(x^2)$ and $p_o(x^2)$ then we can write $p(x) = p_e(x^2) + xp_o(x^2)$ and $p(-x) = p_e(x^2) - xp_o(x^2)$. Notice that both computations utilize $p_e(x^2)$ and $p_o(x^2)$. This can offer a significant speedup when combining answers to subproblems.

We now want to look at the correct basis to evaluate the polynomial in. In example 2.1, we used the points $x = 0, 1, 2$ and $x = 0, \pm i$. We want a basis that is conducive to repeatedly taking the square root of numbers to apply the odd and even trick mentioned above. This immediately suggests the use of complex numbers. In fact, we will use the **roots of unity** that we mentioned in the complex number section for a reason that will become apparent in the analysis of the algorithm.

With all this in mind, the algorithm consists of the following basic steps:

1. Base case: if the polynomial is of degree 0, simply return the constant term.
2. Given that the input polynomial is of degree m, compute the nth roots of unity, where n is a power of 2 larger than m.
3. Break up the polynomial into components p_e and p_o.
4. Recursively call the procedure to evaluate the polynomials p_e and p_o on the $n/2$ roots of unity.
5. Using the result of the recursive procedure, evaluate the polynomials at the nth roots of unity using the relation $p(x) = p_e(x^2) + xp_o(x^2)$.

It is incredibly important that the n chosen is a power of 2 for the recursive structure to be maintained. As this note is updated, we will analyze the operation and runtime of the algorithm more closely.