Great Algorithms: FFT

Aaron Gorenstein

September 1, 2013

1 Background

For brevity, I assume you understand that the FFT algorithm is an evaluation of an n-degree polynomial on n distinct points, thereby converting it from coefficient to point representation in $O(n \log n)$ time instead of the naïve $O(n^2)$ time. I will *not* explain the significance of the "Fourier" part. I *will* explain how the FFT evaluates a polynomial at n points, then use that to motivate our development of ω values.

Before we get into it, understand that this is more like filled-in notes from reading the algorithms textbook I studied [DPV08]. If you've read that and are confused, perhaps my notes may help. The actual book is great, and there is a good draft online. The FFT algorithm is also in CLRS [CLRS01], in the "advanced topics" section. Look there for another sophisticated treatment that's distinct from the one here or in the Dasgupta book (the initially-mentioned [DPV08]).

Please do email me if you have any questions or suggestions.

2 Intuition

We will use this polynomial for our examples:

$$p(x) = 5x^7 + 8x^6 + 2x^5 + x^4 + 7x^3 + 3x^2 + 4x + 6 \tag{1}$$

which can be expressed in code as an array of coefficients $[6, 4, 3, 7, 1, 2, 8, 5]$ (observe the coefficients appear backwards—this is for convenience). As $n = 8$ for this polynomial, we want to evaluate p at 8 distinct points. In other words, our goal is to compute many evaluations of p *very quickly*.

Our overall technique for computing many evaluations quickly follows from this simple fact: if some polynomial q has all *even* exponents, then $q(x) = q(-x)$. For our purposes, this means that evaluating q at some point x gives us two evaluations for the price of one! We immediately get the evaluation of $q(x)$, but also $q(-x)$. This is really the theme of the whole algorithm: we apply that trick in a "divide-and-conquer" fashion.

The first step in a divide-and-conquer algorithm is to divide our initial problem into subproblems. Here, a *subproblem* is a smaller polynomial. In our case we divide p into the polynomials *even* and *odd*:

$$even(x) = 8x^6 + x^4 + 3x^2 + 6 \tag{2}$$
$$odd(x) = 5x^6 + 2x^4 + 7x^2 + 4 \tag{3}$$

Note that

$$p(x) = even(x) + x \cdot odd(x). \tag{4}$$

The polynomial *even* naturally contains all even-exponent terms of p, and *odd* contains all odd-exponent terms *with an x factored out*. We see that eq. (4) is a very natural, obvious way of combining these terms to get back p. It is critical to note that both sub-polynomials $even, odd$ are even-powered-only polynomials.

If one sits down to implement this algorithm, we will immediately realize a problem: the polynomials $even, odd$ as we defined them inspired by eqs. (2) and (3), are *not actually smaller than the original input*!

Explicitly, $even = [6, 0, 2, 0, 1, 0, 8, 0]$ and $odd = [4, 0, 7, 0, 2, 0, 5, 0]$ (recall the coefficients are reversed). While there are a lot more zeros, we're trying to shrink the *length* of the polynomials for each sub-problem, and this does not cut it. So much for dividing.

But what if we represent *even* as $even' = [6, 2, 1, 8]$? Thus, while $even = 8x^6 + x^4 + 2x^2 + 6$, we define $even' = 8x^3 + x^2 + 2x + 6$. In other words, each monomial in $even'$ corresponds to a monomial in *even*, but with *half* the exponent. We can define odd' in a similar fashion. Of course, this loses us the advantage that $even(x) = even(-x)$. However, we can essentially regain that advantage using the following identity:

$$even'(x^2) = even'((-x)^2) = even(x) = even(-x). \tag{5}$$

In other words, we have "factored out" the fact that *even* has all-even exponents. To show that we can really use eq. (5) to get "two evaluations for the price of one", consider the following identities:'

$$even(x) = even'(x^2) \tag{6}$$

$$odd(x) = odd'(x^2) \tag{7}$$

$$p(x) = even'(x^2) + x \cdot odd'(x^2) \tag{8}$$

$$p(-x) = even'(x^2) - x \cdot odd'(x^2). \tag{9}$$

Note that we've really established a mathematical equality here. Thus, we have both the nice divide-and-conquer structure by really using smaller polynomials such that we can use their evaluations at two-to-one price.

For more explicit computation, say we've evaluated $even'$ and odd' both at 4 points, a, b, c, d. Then we can evaluate $p(x)$ at 8 points simply:

$$p(a) = even'(a^2) + a \cdot odd'(a^2) \qquad p(-a) = even'(a^2) - a \cdot odd'(a^2) \tag{10}$$

$$p(b) = even'(b^2) + b \cdot odd'(b^2) \qquad p(-b) = even'(b^2) - b \cdot odd'(b^2) \tag{11}$$

$$p(c) = even'(c^2) + c \cdot odd'(c^2) \qquad p(-c) = even'(c^2) - c \cdot odd'(c^2) \tag{12}$$

$$p(d) = even'(d^2) + d \cdot odd'(d^2) \qquad p(-d) = even'(d^2) - d \cdot odd'(d^2) \tag{13}$$

To have a fully-working FFT algorithm, we need only deal with one more issue: how can we make sure that we never accidentally set two of our input values (such as a and b) to be negations of one another? If $a = -b$, then eqs. (10) and (11) are the same evaluations of p, and we'll only have evaluated p at 6 distinct points! It is rather hard to keep track of what values we've used to evaluate, as we may be recursing a lot (if our input polynomial is huge). To avoid this pitfall, we use a very special kind of value called roots of unity, the subject of the next section.

3 Choosing the Points

Form a top-down perspective, to evaluate $p(\pm x)$ we evaluate $even(x^2) \pm x \cdot odd(x^2)$. From a bottom-up perspective, when we have finished evaluating our polynomial at a point x, then we help our parent (the next level up in the recursion call-tree) compute its own evaluation at $\pm\sqrt{x}$.

Perhaps it is this bottom-up perspective which best illustrates how FFT is efficient: by evaluating a leaf at one point x, we are able to get two points of evaluation for our "parent". So we choose this perspective to explain the very special points we choose to evaluate p at during our FFT computation.

Say we're evaluating a leaf—that is, we need to choose exactly one point on which to evaluate p. Well, why not choose 1. It seems as good as any. This means our parent (needing 2 points of evalutation) is obligated to evaluate its polynomial at $\pm\sqrt{1} = \{+1, -1\}$. So far so good. We can call this set the *square roots* of 1. And the parent of *that* subcall must in turn evaluate its degree-4 polynomial at $\{\pm\sqrt{+1}, \pm\sqrt{-1}\} = \{+1, -1, +i, -i\}$. We can call this set the *fourth roots* of 1. For a degree-8 polynomial, we have to confront the rather curious question: what is $\sqrt{i}$? As it happens that's $\frac{1+i}{\sqrt{2}}$, but that doesn't seem illuminating. More generally, we want to get the *eighth roots* of 1, and of course as the recursive calls continue we'll want to compute the 2^n-th roots of 1. Recall that we always need distinct points on which to evaluate p. So, are there in fact 2^n completely *distinct* 2^n-th roots of 1?

By the magic of math, it is in fact the case! In particular, there is this famous equation (called Euler's identity):

$$e^{i\pi} = -1. \tag{14}$$

Why this is so is out of the scope of this exposition. However, it suffices to realize this: the square roots of 1 are $\{e^{i\pi}, -e^{i\pi}\}$. The fourth roots of 1 are $\{e^{i\pi}, -e^{i\pi}, e^{i\pi/2}, -e^{i\pi/2}\}$. See how what we once wrote as $i, -i$ is now written as $\pm e^{i\pi/2}$? The critical idea is that denominator in the exponent, the $\frac{1}{2}$ factor. It is simply the case that $e^{i\pi/4}$ is an 8-root of unity. This is simply because $(e^{i\pi/4})^4 = -1$, and so $((e^{i\pi/4})^4)^2 = (-1)^2 = 1$. Moreover, $e^{i\pi/8}$ is a 16-th root of unit by similar reasoning.

Playing around with these exponents, one realizes that *all* n-th roots of unity can be generated simply multiplying $e^{2i\pi/n}$ with itself! To make it concrete, consider $n = 16$. For brevity, call $e^{i\pi/8} = \omega$.

$$\begin{aligned}
\omega^1 &= e^{i\pi/8} && (15)\\
\omega^2 &= e^{2i\pi/8} && (16)\\
&\dots && (17)\\
\omega^7 &= e^{7i\pi/8} && (18)\\
\omega^8 &= e^{8i\pi/8} = e^{i\pi} = -1 && (19)\\
\omega^9 &= e^{9i\pi/8} = \omega^8 \cdot \omega = (-1) \cdot \omega && (20)\\
\omega^{10} &= e^{10i\pi/8} = \omega^8 \cdot \omega^2 = (-1) \cdot \omega^2 && (21)\\
&\dots && (22)\\
\omega^{15} &= e^{15i\pi/8} = (-1)\omega^7 && (23)\\
\omega^{16} &= e^{16i\pi/8} = (-1)(-1) = 1 && (24)
\end{aligned}$$

This idea, and all the math, takes a really long time to digest, so it is definitely worth playing around with these ω values themselves, read other expositions (such as the textbook), and of course ask your friends (including your friendly TA and/or professor!). As a final note, observe that for the "second half" of the ω values listed above, each term is a negation of a previously-appearing term. So, $\omega^{11} = -\omega^3$, for instance. Why, this is exactly the trick we need to have our FFT algorithm work![1]

4 Conclusion

Hopefully this has filled in the most confusing aspects of the FFT algorithm. Again, any-and-all feedback welcome, just send me an email. Thanks for reading!

References

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 2001.

[DPV08] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani. *Algorithms*. McGraw-Hill Higher Education, 2008.

[1] I have a feeling this section is particularly incomprehensible. Do let me know what parts confuse you.