Notes on Fast Fourier Transform Algorithms & Data Structures

Dr Mary Cryan

1 Introduction

The *Discrete Fourier Transform (DFT)* is a way of representing functions in terms of a point-value representation (a very specific point-value representation).

We consider complex functions of a single variable throughout these notes, though often the function we are really interested in may actually be a *real* function of a single variable (and we just find it useful to evaluate the function at complex values as well as real values).

Suppose that we consider a polynomial $p(x) = a_0 + a_1x + \dots a_{n-1}x^{n-1}$ over the field of complex numbers $\mathbb{C}$. We refer to the description of $p(x)$ given in the last sentence as its *coefficient form*. However, suppose we took n different complex numbers $x_0, \dots, x_{n-1}$ and then computed y_j, for every $j = 0, \dots, n-1$ as follows:

$$\begin{aligned} y_j = p(x_j) &= \sum_{i=0}^{n-1} a_i (x_j)^i \\ &= a_0 + a_1(x_j) + a_2(x_j)^2 + \dots + a_{n-1}(x_j)^{n-1} \end{aligned}$$

The resulting set of input-output pairs

$$< (x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1}) >$$

is called a *point-value representation* of the polynomial p. It is interesting to ask the question - how much information does a point-value representation convey about the underlying polynomial p? In fact we will see in Subsection 2.2 that the answer is EVERYTHING, if we have evaluated the polynomial at n *different* x_j points.

In this set of lecture notes we focus on the point-value representation obtained by looking at a particular set of points, the nth roots of *unity*. In the field of complex numbers $\mathbb{C}$, there are *exactly* n different solutions to the equation $x^n = 1$. We call these solutions the *n-th roots of unity*. One of these roots of unity is $\omega_n = \cos(2\pi/n) + i\sin(2\pi/n)$, and this is called the *principal nth root of unity*. It is not too difficult to show that ω_n generates the entire set of nth roots of unity as follows, for any n:

$$1, \omega_n, \omega_n^2, \omega_n^3, \dots, \omega_n^{n-1}.$$

The complex numbers in the list above are all nth roots of unity, and they are all different (see Lemma 1 later for a proof).

The *Discrete Fourier Transform (DFT)* of a polynomial $p(x)$ of degree AT MOST $n-1$ is defined to be the point-value representation obtained by evaluating $p(x)$ at each of the n-th roots of unity. We may write the DFT as the list of values

$$p(1), p(\omega), p(\omega^2), \dots, p(\omega^{n-1}),$$

where ω^n is any *primitive nth root of unity*. A *primitive nth root of unity* is any nth root of unity ω such that $1, \omega, \omega^2, \dots, \omega^{n-1}$ are all different. It is not difficult to show that ω is a primitive nth root of unity if $\omega = \omega_n^k$, for some value of k which is *relatively prime* to n (n and k have no common factors). In practice, we adopt the convention that we work with the principal nth root of unity ω_n, unless we state otherwise. It is possible to write the operation of computing the DFT

as a matrix multiplication. The DFT is the vector $y = (y_0, \ldots, y_{n-1})$ such that

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{(n-1)} \\ 1 & \omega_n^2 & \omega_n^4 & \ldots & \omega_n^{2(n-1)} \\ 1 & \omega_n^3 & \omega_n^6 & \ldots & \omega_n^{3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{(n-1)} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

We will however see that it is not the best (most-efficient) approach to perform the matrix multiplication directly.

The Discrete Fourier Transform is a tool used in Signal Processing, It has many many applications (in Speech Recognition, Data Processing, Optics, Acoustics). It is used in any application which performs Digital Signal Processing - to read about this, please follow the relevant links on the ADS course webpage. The straightforward way of computing the DFT would be to consider each root of unity separately. This would take $\Omega(n)$ time per root of unity, and therefore computing the entire DFT would take $\Omega(n^2)$ time by this näive method. However, there is an efficient algorithm, called the *Fast Fourier Transform (FFT)*, which uses the special relationship amongst the roots of unity to compute the entire DFT in $O(n \lg n)$ time. Moreover, we will see that it is possible to take the DFT and recover the original polynomial in its coefficient form in $O(n \lg n)$ time, by using the FFT to compute the *Inverse DFT*.

Because of the many important applications of DFT, the Fast Fourier Transform is probably the most important algorithm of the present day. It came to light as a tool for Signal processing in 1965, when Cooley (of IBM) and Tukey (of Princeton) wrote a paper presenting the algorithm. However the basics of the algorithm were known long before that, for example, it was known to Gauss back in 1805.

2 Complex Numbers

The Discrete Fourier Transform uses complex numbers. There are a few facts (but only a few) about Complex numbers that we need to know for this course.

2.1 Working with Complex Numbers

The *imaginary* number i is defined to be the square root of -1: we write $i = \sqrt{-1}$. The *field* $\mathbb{C}$ of *Complex numbers* is the set of all numbers of the form $a + ib$, for $a, b \in \mathbb{R}$. It is easy to show (and you will have seen it in math classes) that the set of complex numbers $\mathbb{C}$ forms a *field* under the operations of multiplication and addition, just like the field $\mathbb{R}$ of real numbers forms a field under those same operations. However, the complex numbers has an extra property that the real numbers do not have:

> For every element $a + ib \in \mathbb{C}$, and for every natural number k, there are solutions for the k-th root $\sqrt[k]{a + ib}$ in the set $\mathbb{C}$ (in fact there are k such solutions).

So the set of complex numbers has the special property of being closed under taking of roots, which is not the case for real numbers - for example $\sqrt{-1}$ *is not* defined in $\mathbb{R}$, even though $\sqrt[3]{-1} = -1$ *is* defined in $\mathbb{R}$.

The main operations on Complex numbers are $+, -, \times, /$. Let $x = a + ib$, $y = c + id$ be any two complex numbers throughout the following definitions:

- $+$: $x + y = (a + c) + i(b + d)$.

- $-$: $x - y = (a - c) + i(b - d)$.
- $\times$: $x \times y = (a + ib)(c + id) = ac + iad + ibc + i^2bd = (ac - bd) + i(ad + bc)$ (we are using the fact that $i^2 = -1$ to come up with the final representation).
- $/$: This one is tricky. We need to eliminate the"imaginary part" of the denominator y in order to get a nice expression for x/y. Notice that if $y = c + id$, then if we define $y' = c - id$, then $y \times y' = (c^2 + d^2)$. Therefore we can write

$$\begin{aligned} \frac{x}{y} &= \frac{x \times y'}{y \times y'} = \frac{x \times y'}{c^2 + d^2} = \frac{(a + ib)(c - id)}{c^2 + d^2} \\ &= \frac{(ac + bd) + i(bc - ad)}{c^2 + d^2} = \frac{ac + bd}{c^2 + d^2} + i\frac{bc - ad}{c^2 + d^2}. \end{aligned}$$

Some examples of these:

- $+$: If $x = (4 + 3i)$, $y = (-1 + 0.5i)$, then $x + y = 3 + 3.5i$.
- $\times$: If $x = 1 + i$, $y = 2 - i$, then $x \times y = 3 + i$.
  If $x = 1 + 3i$, $y = -2 + 6i$, then $x \times y = (-2 - 18) + i(-6 + 6) = -20$.
- $/$: Let $x = -1$, $y = i$. Then $x/y = (-1/i) \times (i/i) = (-i/-1) = i$.
  Let $x = (4 - i), y = (2 + 3i)$. Then $x/y = (4 - i)/(2 + 3i) \times (2 - 3i)/(2 - 3i) = ((4 - i) \times (2 - 3i))/13 = (5 - 14i)/13$.

A special set of Complex Numbers are the roots of unity, that is, the numbers $\omega \in \mathbb{C}$ such that $\omega^n = 1$. For any given n, there are exactly n such roots in $\mathbb{C}$ (see the Fundamental Theorem of Algebra in the next subsection). We define for any given n, the *principal nth root of unity*

$$\omega_n = \cos(2\pi/n) + i\sin(2\pi/n).$$

Lemma 1 *(i) Each of the complex numbers $1 = \omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ is a n-th root of unity.*

(ii) For every $0 \le j < k \le n - 1$, $\omega_n^j \neq \omega_n^k$ (all the roots of unity are different).

Proof: (i) We use one interesting result relating ω_n to the real number e. Recall the Taylor Series expansion of e^x, for any complex number $x \in \mathbb{C}$:

$$e^x = \sum_{j=0}^{\infty} \frac{x^j}{j!}$$

The Taylor series expansions for $\cos(x)$ and $\sin(x)$ are as follows:

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

If you go to the trouble of checking (using the Taylor Series' written above), you can show that for any real number u, the following inequality holds:

$$e^{iu} = \cos(u) + i\sin(u) \tag{1}$$

Therefore, by (1), our definition of ω_n implies that

$$\omega_n = e^{2\pi i/n} = \cos(2\pi/n) + i\sin(2\pi/n).$$

Therefore for any power $k = 0, \ldots, n - 1$,

$$\omega_n^k = e^{k \times 2\pi i/n} = \cos(2\pi k/n) + i\sin(2\pi k/n).$$

Also for any power $k = 0, 1, \ldots, n-1$,

$$(\omega_n^k)^n \;=\; (e^{k\times 2\pi i/n})^n \;=\; e^{k\times 2\pi i} \;=\; \cos(2\pi k) + i\sin(2\pi k) \;=\; 1,$$

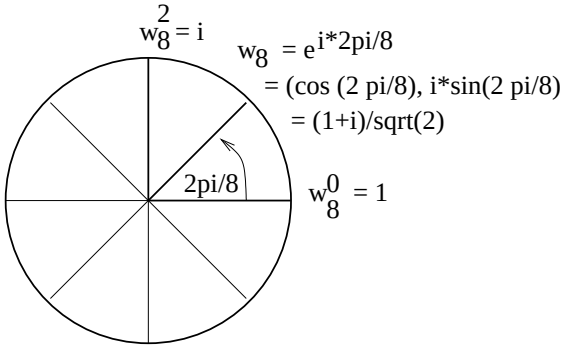
where the second-last step follows by (1), and the last step by $\cos(2\pi k) = 1$ and $\sin(2\pi k) = 0$.

(ii): Equation (1) allows us to deduce the fact that all the elements $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}$ are different. This follows directly from the fact that all the "angles" $1, 2\pi/n, 4\pi/n, 6\pi/n, \ldots, (n-1)2\pi/n$ are all less than 2π, and are all different. Therefore, by the properties of cos and sin, it is *not* possible that we have *both* $\sin(2\pi j/n) = \sin(2\pi k/n)$ *and* $\cos(2\pi j/n) = \cos(2\pi k/n)$ for $0 \le j < k \le n-1$. Therefore the n principal roots of unity defined above are all different.

Corollary 1 *If $\omega = \omega_n^k$ is any* primitive nth root of unity*, then (i) and (ii) also hold for ω.*

Proof: Check yourselves.

We can visualize the n-th roots of unity by interpreting complex numbers as points in the two-dimensional plane. The x-axis represents the "real" part of the complex number and the y-axis represents the "imaginary" part of the complex number. Then the nth roots of unity for any given n correspond to n evenly-spaced points on the unit circle centred around the origin, where ω_n^k is found at a clockwise angle of $2k\pi/n$ from $\omega_n^0 = 1 + 0i$. Note that the principal nth root of unity ω_n is the closest anti-clockwise point to 1 on this wheel



For example, if we consider $n = 4$, then the 4-th roots-of-unity are $1, \omega_4 = i, \omega_4^2 = -1, \omega_4^3 = -i$ (check using the sin and cos representation). It is nice to also check the 8-th roots of unity.

2.2 Fundamental Theorem of Algebra

The most important thing about the field of complex numbers is that it allows us to prove the *Fundamental Theorem of Algebra*:

Theorem 1 (Fundamental Theorem of Algebra) *Let $p(x)$ be any polynomial of degree $n-1$ over the complex numbers (that is, let $p(x) = a_0 + a_1 x + \ldots a_{n-1}x^{n-1}$, for $a_0, \ldots, a_{n-1} \in \mathbb{C}$ and $a_{n-1} \neq 0$). Then there exist roots $\alpha_1, \ldots, \alpha_{n-1} \in \mathbb{C}$ such that*

$$p(x) = \; a_{n-1}(x-\alpha_1)\ldots(x-\alpha_n).$$

That is, the polynomial $p(x)$ has exactly $n-1$ roots (counting multiple occurrences of the same root).

This Theorem does not hold for the field of real numbers (e.g $p(x) = x^2 + 1$ has no real roots).

Clearly, if we are are told that the unknown polynomial $p(x)$ has $n-1$ roots (counting multiplicities) and we are given those roots *and* the leading co-efficient a_{n-1}, then we can construct $p(x)$. In fact, something stronger is true - we do not even need to know the *roots* of the polynomial (and the leading coefficient) to recover it - it is enough to have a point-value representation of the polynomial of size n:

Theorem 2 (Interpolation) *For any set* $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ *of* n *point-value pairs such that the* x_j *values are all different, there is a* unique *polynomial* $A(x)$ *of degree* at most $n-1$ *such that* $y_j = A(x_j)$ *for all* $j = 0, 1, \ldots, n-1$. *Moreover it is possible to compute* $A(x)$ *from the point-value representation.*

Note that this Theorem holds regardless of whether the point-value representation comes from an actual polynomial or is just a list of made-up values. However, if the point-value representation *does* come from an actual polynomial of degree at most $n-1$, then the point-value representation determines that polynomial uniquely.

The general method for converting a point-value representation into a polynomial is based on the idea of inverting a Vandermonde matrix. We will present the Matrix for the Special case when the point-value representation of a polynomial $A(x)$ (of degree at most $n-1$) is calculated at the set of n-th roots of unity.

Then we have a point-value representation of the form

$$< (1, y_0), (\omega_n, y_1), (\omega_n^2, y_2), \ldots, (\omega_n^{n-1}, y_{n-1}) >,$$

where

$$y_j = \sum_{k=0}^{n-1} a_k (\omega_n^j)^k$$

for every $j = 1, \ldots, n-1$. We can write this in Matrix format:

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)^2} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

However, if the $n \times n$ matrix on the left (which we call V_n, and which is actually the DFT matrix) is *invertible*, we can recover the coefficients $a_0, \ldots, a_{n-1}$ from the $y_0, \ldots, y_{n-1}$ values by computing the *inverse* of the $n \times n$ Vandermonde matrix V_n above, and multiplying this inverse by the y-vector.

It can be shown (using the properties of the nth roots of unity), that the matrix above *does* have a inverse, and that this inverse is the following matrix V_n^{-1}:

$$V_n^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \ldots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-4} & \ldots & \omega_n^{-2(n-1)} \\ 1 & \omega_n^{-3} & \omega_n^{-6} & \ldots & \omega_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \ldots & \omega_n^{-(n-1)^2} \end{pmatrix}$$

It is not difficult to check that V_n^{-1} truly is an inverse of V_n. Suppose we check the value of the j,kth entry of $V_n V_n^{-1}$. By Matrix multiplication, this is defined as

$$\begin{aligned} (V_n V_n^{-1})_{jk} &= \frac{1}{n} \sum_{h=0}^{n-1} \omega^{jh} \omega^{-hk} \\ &= \frac{1}{n} \sum_{h=0}^{n-1} \omega^{(j-k)h}. \end{aligned}$$

This is equal to 1 whenever $j = k$, because all the n terms have the value $\omega^0 = 1$, therefore we get the value $n/n = 1$. Whenever $j \neq k$, then we can use the standard formula for the sum of a

geometric series ($\sum_{h=0}^{n} r^n = (1-r^{n+1})/(1-r)$):

$$\begin{aligned}\sum_{h=0}^{n-1} \omega^{(j-k)h} &= (1-\omega^{(j-k)n})/(1-\omega) \\ &= (1-1)/(1-\omega) = 0.\end{aligned}$$

So V_n^{-1} is the inverse of V_n. This implies that a straightforward method for converting from the Discrete Fourier Transform back to the polynomial $A(x)$ is to simply multiply the vector $\mathbf{y}$ of y_j values given by the DFT by V_n^{-1}. This would take $\Omega(n^2)$ time. We will be able to do better by using the fast Fourier Transform.

3 The Fast Fourier Transform (FFT)

Definition 1 *The* Discrete Fourier Transform (DFT) *of a sequence of* n *complex numbers* $a_0, a_1, a_2, \ldots, a_{n-1}$ *is defined to be the sequence of* n *complex numbers*

$$A(1), A(\omega_n), A(\omega_n^2), \ldots, A(\omega_n^{n-1})$$

obtained by evaluating the polynomial

$$A(x) = a_0 + a_1(x) + a_2x^2 + \ldots + a_{n-1}x^{n-1}$$

at each of the n*th roots of unity.*

Consider the task of computing $A(\omega_n^j)$ for every $j = 0, \ldots, n-1$. We are going to use the Divide-and-Conquer approach to set up a recursion for the DFT. Therefore we need to decompose the problem into a number of smaller instances of the DFT. The key to solving the problem lies in making sure that the subproblems that we construct are strictly instances of the DFT problem. We will assume throughout this section that n is a power of 2 (this is ok, if n is not a power of two, we just have a polynomial with some leading zeros at the higher-order coefficients).

We are interested in evaluating:

$$A(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}.$$

Assume that n is even, and partition $A(x)$ into its even parts and its odd parts:

$$\begin{aligned}A(x) &= (a_0 + a_2x^2 + \cdots + a_{n-2}x^{n-2}) + \\ &\quad (a_1x + a_3x^3 + \cdots + a_{n-1}x^{n-1}).\end{aligned}$$

Note that we can also write the sum of odd terms as

$$x(a_1 + a_3x^2 + \cdots + a_{n-1}x^{n-2}),$$

so that we have a sum of even powers of x (multiplied by one extra x). Now make the change of variable from x^2 to y. Therefore we can write

$$\begin{aligned}A_{\text{even}}(y) &= a_0 + a_2y + \cdots + a_{n-2}y^{n/2-1}, \\ A_{\text{odd}}(y) &= a_1 + a_3y + \cdots + a_{n-1}y^{n/2-1}.\end{aligned}$$

and

$$A(x) = A_{\text{even}}(x^2) + x\,A_{\text{odd}}(x^2). \tag{2}$$

To evaluate $A(x)$ at each of the nth roots of unity, it is enough to evaluate $A_{\text{even}}(y)$ and $A_{\text{odd}}(y)$ at each of the points $1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)}$. Once we have done that, then we can use (2) to compute the DFT of $A(x)$ in $O(n)$ extra time (for all roots of unity).

In order to be able to set up a recurrence for the problem, we need to show that each of our two subproblems is a Discrete Fourier Transform problem (as described in Definition 1).

Claim: The following two problems are Discrete Fourier Transforms:
(i) The problem of evaluating $A_{even} = a_0 + a_2 y + \ldots + a_{n-2} y^{n/2-1}$ at the points $1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)}$.
(ii) The problem evaluating $A_{odd} = a_1 + a_3 y + \ldots + a_{n-1} y^{n/2-1}$ at the same points.

We will show that the claim holds for (i) (the second case, (ii), is identical). In order to prove that (i) is a DFT, it is necessary to show that for every $k = 0, \ldots, n-1$, the value ω_n^{2k} is a $n/2$-th root of unity (not just an n-th root). Also it is neccessary to show the values ω_n^{2k}, $k = 0, \ldots, n-1$ are all different (though I'll leave that to the reader). Then the problem of evaluating A_{even} at the given values will satisfy the definition of a DFT.

Lemma 2 *For any $k = 0, \ldots, n-1$, $(\omega_n^k)^2$ is an $n/2$-th root-of-unity.*

Proof:
For any $k = 0, \ldots, n-1$, the relationship between ω_n and e gives:

$$\begin{aligned} \omega_n^{2k} &= e^{(\frac{2\pi i}{n})2k} = e^{\frac{2\pi i}{n/2}k}, \\ &= \omega_{n/2}^k. \end{aligned}$$

Therefore for every $k = 0, 1, \ldots, n/2-1$, certainly it is the case that ω_n^{2k} is a $n/2$-th root of unity - it is equal to $\omega_{n/2}^k$.

Some particular examples of these squares for $k = 0, \ldots, n/2-1$ are $k = 1$ and $k = n/4$:

- $\omega_n^2 = (e^{\frac{2\pi i}{n}})^2 = e^{\frac{2\pi i}{n/2}} = \omega_{n/2}$, and $\omega_n^{n/2} = (e^{\frac{2\pi i}{n}})^{n/2} = e^{\pi i} = -1$.

Alternatively, if $k = n/2, n/2+1, \ldots, n-1$, we have $\omega_n^{2k} = \omega_{n/2}^k = \omega_{n/2}^{n/2}\omega_{n/2}^{k-n/2}$, which is equal to $\omega_{n/2}^{k-n/2}$ (because $\omega_{n/2}^{n/2} = 1$). For these cases we are guaranteed that $k-n/2$ satisfies $0 \leq (k-n/2) \leq n/2-1$, and therefore $\omega_n^{2k} = \omega_{n/2}^{k-n/2}$ is a $n/2$th root of unity. We can represent the relationships between ω_n and $\omega_{n/2}$ diagrammatically:

$$\begin{array}{cccccccc} 1 & \omega_n^2 & \ldots & \omega_n^{n-2} & \omega_n^n & \omega_n^{n+2} & \ldots & \omega_n^{2(n-1)} \\ \| & \| & \ldots & \| & \| & \| & \ldots & \| \\ 1 & \omega_{n/2} & \ldots & \omega_{n/2}^{n/2-1} & 1 & \omega_{n/2} & \ldots & \omega_{n/2}^{n/2-1} \end{array}$$

Therefore, all of our evaluation points $1, \omega_n^2, \omega_n^4, \ldots, \omega_n^{2(n-1)}$ are $n/2$th roots of unity, and we have proved Lemma 2.

Once we have solved the two DFT subproblems of size $n/2$, we can now use (2), and the results of Lemma 2, to compute the DFT values for $A(x)$. The evaluation of $A(x)$ for the first $n/2-1$ nth roots of unity is straightforward:

$$A(1) = A_{even}(1) + 1 \cdot A_{odd}(1)$$

$$A(\omega_n) = A_{even}(\omega_{n/2}) + \omega_n A_{odd}(\omega_{n/2})$$

$$A(\omega_n^2) = A_{even}(\omega_{n/2}^2) + \omega_n^2 A_{odd}(\omega_{n/2}^2)$$

$$\vdots$$

$$A(\omega_n^{n/2-1}) = A_{even}(\omega_{n/2}^{n/2-1}) + \omega_n^{n/2-1} A_{odd}(\omega_{n/2}^{n/2-1})$$

The evaluation of $A(x)$ for the last $n/2$ roots of unity is also straightforward. However, notice that we can replace ω_n^k by $-\omega_n^{k-n/2}$ because $k \geq n/2$ for these cases:

$$A(\omega_n^{n/2}) = A_{\text{even}}(1) - 1 \cdot A_{\text{odd}}(1)$$

$$A(\omega_n^{n/2+1}) = A_{\text{even}}(\omega_{n/2}) - \omega_n A_{\text{odd}}(\omega_{n/2})$$

$$\vdots$$

$$A(\omega_n^{n-1}) = A_{\text{even}}(\omega_{n/2}^{n/2-1}) - \omega_n^{n/2-1} A_{\text{odd}}(\omega_{n/2}^{n/2-1})$$

Observe that the total time to convert the DFT values for A_{even} and A_{odd} into the DFT for the original polynomial A is $O(n)$. Using (2), for each n-th root-of-unity we have to do one multiplication and one addition. The terms being added and multiplied are things that have been given to us by the recursive calls to the two DFTs of size $n/2$, as well as the term ω_n^k (we maintain this by multiplying by another ω_n at every step). So we use a constant extra number of steps for every n-th root-of-unity.

3.1 Fast Fourier Transform (FFT) Algorithm

Here is the *Fast Fourier Transform (FFT)* algorithm, for n a power of 2.

1. If $n = 1$ then the degree of $A(x)$ is at most 0, so $A(x) = a_0$ for a constant a_0. Output is a_0.
2. Alternatively, assume $n \geq 2$.

   (a) Split A into A_{even} and A_{odd}.

   (b) By making two recursive calls to FFT (of size $n/2$) compute the values of $A_{\text{even}}(y)$ and $A_{\text{odd}}(y)$ at the $(n/2)$ points $1, \omega_{n/2}, \omega_{n/2}^2, \ldots, \omega_{n/2}^{n/2-1}$.

   (c) Compute the values of $A(x)$ for every n-th root of unity $1, \omega_n, \ldots, \omega_n^{n-1}$, using the equation
   $$A(x) = A_{\text{even}}(x^2) + xA_{\text{odd}}(x^2),$$
   and our precomputed results for A_{even} and A_{odd}.

   (d) Output these $A(\omega_n^k)$ values in order of increasing k.

We can derive the following recurrence for the running time $T(n)$ of the FFT:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n \geq 2 \end{cases}$$

We can now apply the Master Theorem directly to find the running-time of the FFT. For this case, the critical exponent of the Master Theorem, which is $c = \log_b a$, is $c = \log_2 2 = 1$. and the exponent k from our "extra-work" term $\Theta(n)$ is $k = 1$. Therefore we are in the "middle case" of the Master Theorem (the $c = k$ case), and therefore the running time satisfies

$$T(n) = \Theta(n^c \log n) = \Theta(n \log n) = \Theta(n \lg n),$$

as required.

4 Using FFT to multiply polynomials in $O(n \lg n)$ time

We now show how to use the Discrete Fourier transform to multiply two polynomials of degree at most $n-1$ in $\Theta(n \lg n)$ time.

Suppose that we are given two polynomials $p(x)$ and $q(x)$, where

$$\begin{aligned} p(x) &= a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \quad \text{and} \\ q(x) &= b_0 + b_1 x + \ldots + a_{n-1} x^{m-1}, \end{aligned}$$

and $\max\{m, n\} = n$. If we multiply the polynomials together, clearly the polynomial pq will have degree $n + m - 2$. Suppose that the polynomial $pq(x)$ has the coefficient representation

$$pq(x) = c_0 + c_1 x + \ldots + c_{n+m-2} x^{n+m-2}$$

If we think about the straightforward method for polynomial multiplication, we know that the c_j coefficients will follow the following pattern:

$$\begin{aligned} c_0 &= a_0 b_0 \\ c_1 &= a_1 b_0 + a_0 b_1 \\ c_2 &= a_2 b_0 + a_1 b_1 + a_0 b_2 \\ &\vdots \\ c_{n+m-3} &= a_{n-1} b_{m-2} + a_{n-2} b_{m-1} \\ c_{n+m-2} &= a_{n-1} b_{m-1} \end{aligned}$$

However to do the multiplication in this näive way is computationally expensive - it takes $\Theta(nm)$ time, which could be as bad as $\Theta(n^2)$ when n is about the same size as m. We will show how to compute pq in $O(n \lg n)$ time (assuming $m \leq n$).

Now recall Theorem 2, and the remarks after it. This gives us a hint that there may be a different approach. Remember that Theorem 2 tells us that if we have a point-value representation of size at least $n + m - 1$ (the size of the degree plus 1) for $pq(X)$, then we can recover the polynomial $pq(X)$. So there is an alternative method ...

Moreover, the discussion after Theorem 2 tells us that if n' is the value of the smallest power of 2 which is greater than $n + m - 1$, then if we consider the n'-th roots-of-unity, and if we have a point-value representation of $pq(x)$ in terms of the n'-th roots-of-unity, then we can recover the coefficients of pq by performing the "Inverse DFT". The interesting thing is that we do not need to know the polynomial pq in order to evaluate $pq(1), pq(\omega_{n'}), \ldots, pq(\omega_{n'}^{n'-1})$ (a good thing since we are hoping to *use* the point-value representation to find the polynomial pq). If we know the value of $p(\omega_{n'}^k)$ and $q(\omega_{n'}^k)$ for any $k \in \{0, \ldots, n'-1\}$, then we have $pq(\omega_{n'}^k) = p(\omega_{n'}^k) q(\omega_{n'}^k)$.

Therefore it seems that a good first step is to compute the DFT for p on all n'-th roots of unity, and also to compute the DFT for q on all n'-roots of unity. Both of these steps will take $O(n' \lg n') = O(n \lg n)$ time, so they are good as a first approach.
We can then compute the DFT for pq on all n' roots of unity by setting $pq(\omega_{n'}^k) = p(\omega_{n'}^k) q(\omega_{n'}^k)$. This takes $O(n)$ extra time. Let $y_0, y_1, \ldots, y_{n'-1}$ be the values that we get as a result.
Now recall from Subsection 2.2 that we can recover the coefficients $c_0, c_1, \ldots, c_{n'-1}$ of the polynomial pq (if n' is strictly greater than $n + m - 1$, then some of these leading coefficients will be 0) by evaluating the result of multiplying the inverse DFT $V_{n'}^{-1}$ by $(y_0, y_1, \ldots, y_{n'-1})^T$. We know from Subsection 2.2 that

$$V_{n'}^{-1} y^T = \frac{1}{n'} \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_{n'}^{-1} & \omega_{n'}^{-2} & \ldots & \omega_{n'}^{-(n'-1)} \\ 1 & \omega_{n'}^{-2} & \omega_{n'}^{-4} & \ldots & \omega_{n'}^{-2(n'-1)} \\ 1 & \omega_{n'}^{-3} & \omega_{n'}^{-6} & \ldots & \omega_{n'}^{-3(n'-1)} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega_{n'}^{-(n'-1)} & \omega_{n'}^{-2(n'-1)} & \ldots & \omega_{n'}^{-(n'-1)^2} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n'-1} \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n'-1} \end{pmatrix}.$$

We do not want to evaluate this multiplication directly as it would take $\Theta(n'^2) = \Theta(n^2)$ time. However we do not have to evaluate it directly, because it is actually another DFT!!

Examine the rows of the square matrix. Note that each row is $1 \;\; \omega_{n'}^{-k} \;\; \omega_{n'}^{-2k} \;\; \ldots \;\; \omega_{n'}^{-k(n'-1)}$ for some k. But for any $0 \leq k \leq n'-1$,

$$\omega_{n'}^{-k} \;=\; \omega_{n'}^{n'-k}/\omega_{n'}^{n'} \;=\; \omega_{n'}^{n'-k},$$

where the last step follows by $\omega_{n'}^{n'} = 1$ (since $\omega_{n'}$ is a n'-th root of unity). Observe that although $\omega_{n'}^{-1}$ is not the principal n'th root of unity, it is a *primitive* n'th root of unity. Moreover, if we examine the $k+1$-th row of the Inverse DFT (forgetting the $1/n'$ term outside), this is equivalent to the $n'-k+1$th row of the original DFT table.

Therefore the Inverse DFT matrix is exactly the same as the DFT with row 1 (for $\omega_n^0 = \omega_n^{-0} = 1$) fixed and with the rows $2, \ldots, n'$ flipped, all divided by n'.

4.1 The $\Theta(n \lg(n))$ Inverse FFT Algorithm

Therefore we can **compute the inverse DFT** (for n') of $(y_0, y_1, \ldots, y_{n'-1})$ as follows:

- Compute the DFT (yes, that's right) of $(y_0, y_1, \ldots, y_{n'-1})$ (for n'). Assume the result of this is $d_0, d_1, d_2, \ldots, d_{n'-1}$.
- Now reverse the sequence of terms $d_1, d_2, \ldots, d_{n'-1}$ in this result, keeping d_0 fixed, and then divide every term by n', to obtain $c_0, c_1, \ldots, c_{n'-1}$. So we have

$$c_i \;=\; \begin{cases} \frac{d_0}{n'} & \text{if } i = 0 \\ \frac{d_{n'-i}}{n'} & \text{if } 1 \leq i \leq n'-1 \end{cases}$$

- This all takes $\Theta(n' \lg n') = \Theta(n \lg n)$ time.

4.2 The $\Theta(n \lg(n))$ Polynomial Multiplication Algorithm

Here is the *FFT multiplication* algorithm:

1. (i) Consider the two polynomials p (of degree $n-1$) and q (of degree $m-1$). Let n' be the smallest power of 2 satisfying $n' \geq n+m-1$.

   (ii) Call the FFT algorithm on the polynomial p to calculate the values of $p(\omega_{n'}^k)$ for all the n'-th roots of unity (ie for all $0 \leq k \leq n'-1$).

   (iii) Call the FFT algorithm on the polynomial q to calculate the values of $q(\omega_{n'}^k)$ for all the n'-th roots of unity (ie for all $0 \leq k \leq n'-1$).

2. Compute $y_k = pq(\omega_{n'}^k) = p(\omega_{n'}^k)q(\omega_{n'}^k)$ for every $k = 0, 1, \ldots, n'-1$. This is the DFT for pq.

3. Compute the inverse DFT by a single application of the FFT algorithm. To do this, first compute the *standard DFT* on the pointwise representation $< y_0, y_1, \ldots, y_{n'-1} >$, to obtain the vector $< z_0, z_1, \ldots, z_{n'-1} >$. Then we use this to get the inverse DFT z' as follows:

$$z'_j = \begin{cases} z_0/(n') & \text{if } j = 0 \\ z_{n'-j}/(n') & \text{for } j = 1, \ldots, n'-1 \end{cases}.$$

   The special case for the first entry of z' is due to the fact that the initial row of the inverse DFT matrix $V_{n'}^{-1}$ is exactly $1/(n')$ of the first row of the DFT matrix $V_{n'}$ (the "all-1s" row). For every $j = 1, \ldots, n'-1$, the $j+1$ row of $V_{n'}^{-1}$ (which is written in terms of $\omega_{n'}^{-j}$) instead matches the $n'-j+1$ row of $V_{n'}$ (written for $\omega_{n'}^{n'-j}$, which is equal to $\omega_{n'}^{-j} = \omega_{n'}^{n'}\omega_{n'}^{-j}$).

   Then $z' =< z_0/(n'), z_{n'-1}/(n'), \ldots, z_2/(n'), z_1/(n') >$ is the list of co-efficients $c_0, c_1, \ldots, c_{n'-1}$ of the product polynomial pq.

Steps (1) and (4) take $O(n')$ time altogether. Steps (2), (3) and (5) take $O(n' \lg n')$ time each. Hence the entire algorithm takes $O(n' \lg n') = O(n \lg n)$ time.

4.3 Using the Multiplication Algorithm

Sometimes you will get a question asking you to multiply two polynomials p and q using the $O(n \lg n)$ FFT algorithm.

To answer these questions you do not usually have to perform all the recursive steps of the FFT (unless you are explicitly told to), you can evaluate the DFTs directly using your knowledge of basic arithmetic of complex numbers. However, you do have to carry out the three main steps of the algorithm described in the last section:

Here is an example.

Step 1:
I will not do the first step of computing the DFTs of p and q, since this is an easier step. But suppose we had originally some polynomial p of degree $n-1=2$ and another polynomial of degree $m-1=1$. Then $n+m-1=4$ and therefore our n' (the smallest power of two of size at least $n+m-1$) is also 4. Therefore Step 1 (not done here) will have done the task of computing the DFT of p on all the 4-th roots-of-unity, and of computing the DFT of q on all the 4-th roots-of-unity.

Step 2:
Suppose that we have got the values $(4, i+1, 2, 1-i)$ for the DFT of p, and the values $(4, i+3, 2, -i+3)$ for the DFT of q. This corresponds to the following point-value representations (using $\omega_4^1 = i$, $\omega_4^2 = -1$, $\omega_4^3 = -i$) for p and q respectively:

$$< (1,4), (i, i+1), (-1,2), (-i, 1-i) > \qquad \text{DFT for } p$$
$$< (1,4), (i, i+3), (-1,2), (-i, -i+3) > \qquad \text{DFT for } q$$

We construct the DFT/point-value representation for pq by taking the component-wise multiplication: for 1, we have $4 \times 4 = 16$; for $\omega_4 = i$, we have $(i+1) \times (i+3) = i^2 + 4i + 3 = -1 + 4i + 3 = 4i + 2$; for $\omega_4^2 = -1$, we have $2 \times 2 = 4$; and for $\omega_4^3 = -i$, we have $(1-i)(3-i) = 3 - 4i + i^2 = 3 - 4i - 1 = 2 - 4i$. Therefore we have the following DFT for our unknown polynomial pq:

$$< (1,16), (i, 4i+2), (-1,4), (-i, 2-4i) > \qquad \text{DFT for } pq$$

Step 3:
To find the polynomial $pq(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$, we only need to evaluate the product of the inverse DFT with the matrix $(16 \;\; 4i+2 \;\; 4 \;\; 2-4i)^T$, which is

$$\frac{1}{4} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4^{-1} & \omega_4^{-2} & \omega_4^{-3} \\ 1 & \omega_4^{-2} & \omega_4^{-4} & \omega_4^{-6} \\ 1 & \omega_4^{-3} & \omega_4^{-6} & \omega_4^{-9} \end{pmatrix} \begin{pmatrix} 16 \\ 4i+2 \\ 4 \\ 2-4i \end{pmatrix} = \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix}.$$

We work out the rows one by one:

$c_0 = 1/4(16 + 4i + 2 + 4 + 2 - 4i) = 1/4(24) = 6.$

c_1: Note that $\omega_4^{-1} = \omega_4^3 = -i$. So we have $c_1 = 1/4(16 + (-i)(4i+2) + (-i)^2(4) + (-i)^3(2-4i)) = (1/4)(16 + (4-2i) + (-4) + (2i+4)) = (1/4)(20) = 5.$

c_2: Note that $\omega_4^{-2} = \omega_4^2 = -1$. So we have $c_2 = (1/4)(16 + (-1)(4i+2) + (-1)^2(4) + (-1)^3(2-4i)) = (1/4)(16 - 4i - 2 + 4 - 2 + 4i) = (1/4)(16) = 4.$

c_3: Note that $\omega_4^{-3} = \omega_4 = i$. So we have $c_3 = (1/4)(16 + i(4i+2) + (i)^2(4) + (i)^3(2-4i)) = (1/4)(16 - 4 + 2i - 4 - 2i - 4) = (1/4)(4) = 1.$

Therefore $pq(x) = x^3 + 4x^2 + 5x + 6$, and we have our result.

When you are given one of these questions as an exercise, you (of course) have to perform Step 1, which I assumed.

Mary Cryan