Computational Number Theory and Algebra April 30, 2012

Lecture 4

Lecturers: Markus Bläser, Chandan Saha *Scribe: Chandan Saha*

Our topic of discussion today is *Discrete Fourier Transform* (DFT). In general, Fourier transform is a highly important tool used in mathematics, computer science, electrical engineering and many other subjects. At a high level, Fourier transform is a method to express vectors by choosing a 'nice' basis so that the coordinates of a vector expressed in this basis carry 'important information' about the vector. Today, we will discuss the following topics:

- Discrete Fourier Transform (DFT),
- Fast Fourier Transform (FFT),
- Polynomial multiplication (assuming that the underlying ring *supports* FFT).

1 Discrete Fourier Transform

This tool is used extensively to design asymptotically fast algorithms for polynomial and integer multiplications. In this lecture and the next, we will see polynomial and integer multiplication algorithms based on DFT. But, at first we need to refresh a few concepts about roots of unity in a ring. The following exposition to DFT and FFT is taken from the book [GG03].

Let $\mathcal{R}$ be a commutative ring. An element $a \in \mathcal{R}$ is a *zero-divisor* if there is a nonzero $b \in \mathcal{R}$ such that $a \cdot b = 0$. An element $a \in \mathcal{R}$ is called a *unit* if there is a $b \in \mathcal{R}$ such that $a \cdot b = 1$.

Definition 1 (Principal root of unity) *An element ω is an n^{th} root of unity in $\mathcal{R}$ if $\omega^n = 1$, where $n \in \mathbb{Z}^+$. Further, ω is a* principal *n^{th} root of unity if $\omega^{n/t} - 1$ is not a zero-divisor in $\mathcal{R}$ for every prime divisor t of n, and n is a unit in $\mathcal{R}$.*

(In order to understand this discussion, you can assume for the moment that $\mathcal{R}$ is a field, in which case, ω (in the above definition) is simply a primitive root of unity. But, it is advisable that you familiarize yourself with the notion of principal roots *over rings*, as we would need this concept while discussing an integer multiplication algorithm in the next class.)

Lemma 2 *If ω is a principal n^{th} root of unity in $\mathcal{R}$ then for every $1 \leq \ell < n$, $\omega^\ell - 1$ is not a zero-divisor in $\mathcal{R}$ and $\sum_{i=0}^{n-1} \omega^{\ell i} = 0$.*

We leave the proof of this lemma as an exercise. Let $\mathbf{f} = (f_0, f_1, \ldots, f_{n-1}) \in \mathcal{R}^n$. Define polynomial $f(x) = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}[x]$. Suppose that $\mathcal{R}$ contains a principal n^{th} root of unity ω. The Discrete Fourier Transform is a map from $\mathcal{R}^n \to \mathcal{R}^n$ defined as:

$$\mathsf{DFT}_\omega : \quad \mathbf{f} \mapsto (f(1), f(\omega), \ldots, f(\omega^{n-1})).$$

Note that the map DFT_ω is defined with respect to a particular ω. Let $\mathsf{D}(\omega)$ be the following matrix:

$$\mathsf{D}(\omega) = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)^2} \end{pmatrix}_{n \times n}$$

Then, $(f(1), f(\omega), \ldots, f(\omega^{n-1})) = \mathbf{f} \cdot \mathsf{D}(\omega)$. Further, observe that since ω is a principal root of unity, $\mathsf{D}(\omega) \cdot \mathsf{D}(\omega^{-1}) = n \cdot I$, where I is the $n \times n$ identity matrix (by Lemma 2). Therefore, $n \cdot \mathbf{f} = (f(1), f(\omega), \ldots, f(\omega^{n-1})) \cdot \mathsf{D}(\omega^{-1})$. Here, $\omega^{-1} = \omega^{n-1}$ is also a principal n^{th} root of unity in $\mathcal{R}$ (why?). This means that the map $\mathsf{DFT}_{\omega^{-1}}$ (Discrete Fourier Transform with respect to ω^{-1}) is such that

$$\mathsf{DFT}_{\omega^{-1}} : \quad (f(1), f(\omega), \ldots, f(\omega^{n-1})) \mapsto n \cdot \mathbf{f} \tag{1}$$

Assuming $\sqrt{n}$ to be a unit in $\mathcal{R}$, do you see that the map DFT_ω is actually a change of basis? The map $\mathsf{DFT}_{\omega^{-1}}$ will also be referred to as the *inverse*-DFT map.

We will identity the vector $\mathbf{f}$ with the polynomial f. Computing the DFT of f (using ω as the root of unity) is the task of finding the vector $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$. Computing inverse-DFT is like interpolating the polynomial f from its values $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$ (except that we get $n \cdot f$ instead of f). This task of computing a DFT can be performed efficiently using an algorithm called the Fast Fourier Transform.

2 Fast Fourier Transform

The Fast Fourier Transform (FFT) was found by Gauss in 1805 and later (re)discovered by Cooley and Tukey [CT65]. The algorithm uses a recursive technique to compute DFT of a polynomial $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}[x]$, using $O(n \log n)$ addition and multiplication operations in $\mathcal{R}$. Assume that n is a power of 2. We wish to compute DFT of f using $\omega \in \mathcal{R}$ as the n^{th} root of unity.

Algorithm 1 Fast Fourier Transform

```
1.  If $n = 1$ return $f(0)$.
2.  Define $g_0$ as $g_0 = \sum_{i=0}^{n/2-1} (f_i + f_{\frac{n}{2}+i}) x^i$.
3.  Define $g_1$ as $g_1 = \sum_{i=0}^{n/2-1} (f_i - f_{\frac{n}{2}+i}) \omega^i x^i$.
4.  Recursively, compute DFT of $g_0$ using $\omega^2$ as the $n/2$-th root of unity.
5.  Recursively, compute DFT of $g_1$ using $\omega^2$ as the $n/2$-th root of unity.
6.  Return $(g_0(1), g_1(1), g_0(\omega^2), g_1(\omega^2), \ldots, g_0(\omega^{n-2}), g_1(\omega^{n-2}))$.
```

Correctness of the algorithm - The correctness follows from the next two equations, for $0 \le j \le \frac{n}{2} - 1$.

$$\begin{aligned} f(\omega^{2j}) &= \sum_{i=0}^{n-1} f_i \omega^{2ij} = \sum_{i=0}^{\frac{n}{2}-1} (f_i + f_{n/2+i}) \cdot (\omega^2)^{ij} \quad \text{and} \\ f(\omega^{2j+1}) &= \sum_{i=0}^{n-1} f_i \omega^{i(2j+1)} = \sum_{i=0}^{\frac{n}{2}-1} (f_i - f_{n/2+i}) \omega^i \cdot (\omega^2)^{ij} \end{aligned}$$

The negative sign in the second equation comes from the fact that $\omega^{n/2} = -1$. Observe that ω^2 is a principal $n/2$-th root of unity in $\mathcal{R}$ (why?). Thus, the problem of computing DFT of a polynomial f of degree bounded by $n-1$ reduces to computing the DFT of two polynomials g_0 and g_1 (as defined in the algorithm) with degree bounded by $n/2 - 1$.

Time complexity - Computing the coefficients of g_0 takes $n/2$ additions in $\mathcal{R}$, while computing the coefficients of g_1 takes $n/2$ additions in $\mathcal{R}$ and $n/2$ multiplications by powers of ω. Each of step 4 and 5 computes a DFT of a polynomial whose degree is bounded by $n/2 - 1$. By solving the recurrence, we get the following lemma.

Lemma 3 (FFT complexity) *Fast Fourier Transform computes the DFT of a polynomial f with degree bounded by $n - 1$ using $O(n \log n)$ additions in $\mathcal{R}$ and $O(n \log n)$ multiplications by powers of ω.*

Let us now see an application of FFT in designing a polynomial multiplication algorithm over a ring that 'supports' FFT. We say that a ring $\mathcal{R}$ supports FFT for a choice of n, if $\mathcal{R}$ contains a principal n^{th} root of unity.

3 Polynomial multiplication over a ring that supports FFT

Suppose $f, g \in \mathcal{R}[x]$ be two polynomials of degree less than $n/2$, where $\mathcal{R}$ contains a principal n^{th} root of unity ω (once again assume that n is a power of 2). The product polynomial $h = fg$ has degree less than n and hence it makes sense to talk about DFT of h using a principal n^{th} root of unity. Let $h = \sum_{i=0}^{n-1} h_i x^i$ and $\mathbf{h} = (h_0, h_1, \ldots, h_{n-1})$. Recall from Equation (1) that the inverse-DFT is a map $\mathsf{DFT}_{\omega^{-1}}$ that sends,

$$\mathsf{DFT}_{\omega^{-1}} : \quad (h(1), h(\omega), \ldots, h(\omega^{n-1})) \mapsto n \cdot \mathbf{h}$$

Basically, $\mathsf{DFT}_{\omega^{-1}}$ is a DFT using ω^{n-1} as the root of unity. This observation suggests the following polynomial multiplication algorithm. Let $\tilde{h} = \sum_{i=0}^{n-1} h(\omega^i) x^i$, where $h(\omega^i) = f(\omega^i) g(\omega^i)$.

Algorithm 2 Polynomial multiplication using FFT

```
1. Compute DFT of $f$ to find the vector $(f(1), f(\omega), \ldots, f(\omega^{n-1}))$.
2. Compute DFT of $g$ to find the vector $(g(1), g(\omega), \ldots, g(\omega^{n-1}))$.
3. Multiply the two vectors component-wise and obtain $(h(1), h(\omega), \ldots, h(\omega^{n-1}))$.
4. Compute DFT of $\tilde{h}$ using $\omega^{n-1}$ as the root of unity to get the vector $n \cdot \mathbf{h}$.
5. Multiply $n \cdot \mathbf{h}$ by the inverse of $n$ in $\mathcal{R}$ to get $\mathbf{h}$.
```

Time complexity - In steps 1, 2 and 4 the algorithm computes three DFTs, each using a principal n^{th} root of unity. The component-wise multiplications in step 3 takes n multiplications in $\mathcal{R}$, and step 5 takes n multiplications by the inverse of n in $\mathcal{R}$ (assume that we know the inverse of n a priori). Therefore, the overall time complexity of the algorithm is $O(n \log n)$ additions and mulitplications in $\mathcal{R}$. (The trivial polynomial multiplication algorithm would have taken $O(n^2)$ additions and multiplications over $\mathcal{R}$.) We summarize this in the following lemma.

Lemma 4 *Multiplication of two polynomials of degree less than $n/2$ (n is a power of 2) in $\mathcal{R}[x]$, where $\mathcal{R}$ contains a principal n^{th} root of unity ω, takes $O(n \log n)$ additions in $\mathcal{R}$, $O(n \log n)$ multiplications by powers of ω, n multiplications by the inverse of n in $\mathcal{R}$, and n multiplications in $\mathcal{R}$.*

The last n multiplications in $\mathcal{R}$ comes from the component-wise multiplications in step 3 of the above algorithm. The assumption we made for the above algorithm to work is that the underlying ring $\mathcal{R}$ contains a principal n^{th} root of unity, where n is a power of 2. What if it doesn't? This is handled by an algorithm due to Schönhage and Strassen [SS71]. The basic idea behind their algorithm is to attach a "virtual" root of unity to the ring $\mathcal{R}$, if $\mathcal{R}$ doesn't contain a suitable root of unity to begin with. We will see a glimpse of this idea at work when we discuss an integer multiplication algorithm in the next class. However, in the process of attaching a "virtual" root, the complexity of the polynomial multiplication algorithm becomes slightly worse.

Theorem 5 [Schönhage and Strassen (1971)] *Over any commutative ring $\mathcal{R}$, two polynomials of degree less than n can be multiplied using $O(n \log n \log\log n)$ additions and multiplications in $\mathcal{R}$.*

Exercises:

1. Prove Lemma 2.
2. Show that if ω is a principal n^{th} root of unity in $\mathcal{R}$ then ω^{n-1} is also a principal n^{th} root of unity in $\mathcal{R}$.
3. Show that if ω is a principal n^{th} root of unity in $\mathcal{R}$ and t divides n, then $\omega^{n/t}$ is a principal t^{th} root of unity in $\mathcal{R}$.

References

[CT65] James W. Cooley and John W. Tukey. An Algorithm for the Machine Calculation of Complex Fourier Series. *Mathematics of Computation*, 19(90):297–301, 1965.

[GG03] Joachim Von Zur Gathen and Jurgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[SS71] A Schönhage and V Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.