CS 681: Computational Number Theory and Algebra **Lecture 5**
Lecturer: Manindra Agrawal **Notes by: Ashwini Aroskar**

August 11, 2004.

1 Discrete Fourier Transform

Let $f : [0, n-1] \to F$ be a function. F is a field.

Definition 1.1 *The Discrete Fourier Transform of f is defined as*

$$DFT_f(j) = \sum_{i=0}^{n-1} f(i)\omega^{ij}; \quad 0 \leq j < n$$

where ω is a principal n^{th} root of unity, i.e., $\omega^n = 1$ and $\omega^m \neq 1$ for $0 < m < n$

So, $DFT_f : [0, n-1] \to F[\omega]$, in general.

1.1 Evaluating DFT_f

Given $f = (c_0, c_1, \ldots, c_{n-1})$, compute $(d_0, d_1, \ldots, d_{n-1})$ with $d_j = DFT_f(j)$

Time complexity of naïve algorithm $= \bigcirc(n^2)$ operations over F.

1.2 Computing Inverse DFT

Theorem 1.1 $\frac{1}{n} DFT[\omega^{-1}]_{DFT[\omega]_f} = f$

Proof: Suppose $DFT[\omega](c_0, c_1, \ldots, c_{n-1}) = (d_0, d_1, \ldots, d_{n-1})$
Then $d_j = \sum_{i=0}^{n-1} c_i \omega^{ij}$

Let $DFT[\omega^{-1}](d_0, d_1, \ldots, d_{n-1}) = (e_0, e_1, \ldots, e_{n-1})$

$$\begin{aligned}
e_j &= \sum_{i=0}^{n-1} d_i \omega^{-ij} \\
&= \sum_{i=0}^{n-1} (\sum_{t=0}^{n-1} c_t \omega^{ti}) \omega^{-ij} \\
&= \sum_{t=0}^{n-1} \sum_{i=0}^{n-1} (\omega^{(t-j)})^i \\
&= nc_j
\end{aligned}$$

as

$$\begin{aligned}\sum_{i=0}^{n-1}(\omega^{(t-j)})^i &= 0 \quad \forall t \neq j \\ &= n \quad t = j\end{aligned}$$

■

2 Fast Fourier Transform

Proposed by Gauss in the 1800's, Fast Fourier Transforms employ the Divide-and-Conquer technique.
Assume $n = 2^m$ for some $m > 0$

$$\begin{aligned}d_j &= \sum_{i=0}^{2^m-1} c_i\omega^{ij} \\ &= \sum_{i=0}^{2^{m-1}-1} c_{2i}\omega^{2ij} + c_{2i+1}\omega^{(2i+1)j} \\ &= \sum_{i=0}^{2^{m-1}-1} c_{2i}\omega^{2ij} + \omega^j \sum_{i=0}^{2^{m-1}-1} c_{2i+1}(\omega^2)^{ij}\end{aligned}$$

Let $f_0 = (c_0, c_2, \ldots, c_{n-2})$ and $f_1 = (c_1, c_3, \ldots, c_{n-1})$
Let $(e_0, e_1, \ldots, e_{\frac{n}{2}-1}) = DFT_{f_0}$ and $(e'_0, e'_1, \ldots, e'_{\frac{n}{2}-1}) = DFT_{f_1}$

Then

$$\begin{aligned}d_j &= e_j + \omega^j e'_j \qquad 0 < j < \frac{n}{2} \\ &= e_{j-\frac{n}{2}} + \omega^j e'_{j-\frac{n}{2}} \quad j \geq \frac{n}{2}\end{aligned}$$

Time Complexity of $FFT = T(n)$
$T(n) = 2T(\frac{n}{2}) + \bigcirc(n) = \bigcirc(n \log n)$

Therefore, time complexity of computing DFT or $InverseDFT = \bigcirc(nlogn)$

2.1 Polynomial Multiplication

Given polynomials $P(x)$ and $Q(x)$, both of degree $n-1$, compute $P * Q(x)$
Obvious time complexity $= \bigcirc(n^2)$

Let $DFT_P = (d_0, d_1, \ldots, d_{n-1})$ where $d_j = P(\omega^j)$
and $DFT_Q = (e_0, e_1, \ldots, e_{n-1})$ where $e_j = Q(\omega^j)$

$P * Q(\omega^j) = P(\omega^j)Q(\omega^j) = d_j e_j$
$DFT_{P*Q} = (d_0e_0, d_1e_1, \ldots, d_{n-1}e_{n-1})$ and can be computed using $\bigcirc(n)$ operations if DFT_P and DFT_Q are known.

$\deg P * Q = 2n - 2$
Pretend that P and Q are $\deg l$ polynomials with $l \geq n - 2$ and $l = 2^k - 1$ for some k. So, use ω, a principal $2^k th$ root of unity.
$P * Q$ is also a $\deg l$ polynomial.

Time complexity of computing $P * Q$ via DFT
$= \bigcirc(l \log l) + \bigcirc(l) + \bigcirc(l \log l)$
$= \bigcirc(nlogn)$ as $l = \bigcirc(n)$

2.2 Integer Multiplication

Given integers a and b, both n bit long, compute $a * b$.
Obvious Time Complexity = $\bigcirc(n^2)$

Let $a = a_0a_1 \ldots a_{n-1}$ and $b = b_0b_1 \ldots b_{n-1}$.
$a = \sum_{i=0}^{n-1} a_i 2^i$ and $b = \sum_{i=0}^{n-1} b_i 2^i$

Assume $n = 2^k$. Let $l \mid n$. Let $n = l * t$.
Split a and b into t blocks of l bits each.

Let $a = \hat{a}_0\hat{a}_1 \ldots \hat{a}_{t-1}$ and $b = \hat{b}_0\hat{b}_1 \ldots \hat{b}_{t-1}$.

Then $a = \sum_{i=0}^{t-1} \hat{a}_i 2^{il}$ and $b = \sum_{i=0}^{t-1} \hat{b}_i 2^{il}$.

Let $a(x) = \sum_{i=0}^{t-1} \hat{a}_i x^i$ and $b(x) = \sum_{i=0}^{t-1} \hat{b}_i x^i$.