CS170 - Lecture 5
Sanjam Garg
UC Berkeley

Recall: Multiplying polynomials, coefficient/value representation

Evaluation: $O(n \log n)$ if choose $1, \omega, \omega^2, \ldots, \omega^{n-1}$.

Evaluation

Coefficient representation: $a_0, a_1, \ldots, a_{n-1}$ → Value representation: $A(x_0), A(x_1), \ldots, A(x_{n-1})$.

Interpolation

Interpolation: From points $A(x_0), \ldots, A(x_{n-1})$ to coefficients..
We will see this today!

Polynomial Evaluation and Matrices

Evaluation: Compute $A(\cdot)$ from a_i's:

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots x_1^{n-1} \\ & & \vdots & \cdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

Interpolation (going back to coefficient matrix).
How?

Compute inverse of matrix above.
Multiply. $O(n^2)$!

This sounds expensive!!

Also, computing inverse not even easy.

Using roots of unity

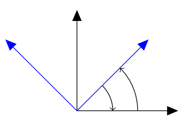
FFT: ω is complex nth root of unity
and matrix is ...

$$M_n(\omega) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{(n-1)} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ \vdots & & \vdots & & \\ 1 & \omega^j & \omega^{2j} & \cdots & \omega^{j(n-1)} \\ \vdots & & \vdots & & \\ 1 & \omega^{(n-1)} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}$$

Compute inverse of $M_n(\omega)$?

Geometry and FFT.

Rows are orthogonal.
Multiply by $M_n(\omega)$: project point onto each row (and scaled.)
Rigid Rotation (and scaling.)!

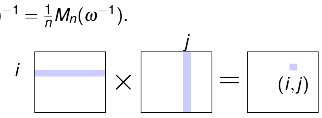


Reverse Rotation is inverse operation.
Scaling: for rotation, axis should have length 1, FFT length n.

Algebraically.

Inversion formula: $(M_n(\omega))^{-1} = \frac{1}{n} M_n(\omega^{-1})$.

$C = M_n(\omega) \times M_n(\omega^{-1})$?



Recall: $\omega = e^{2\pi/n}$.

$$c_{ij} = \sum_k \omega^{ik}\omega^{-kj} = \sum_k \omega^{(ik-kj)} = \sum_k \omega^{k(i-j)} = \sum_k r^k, \quad r = \omega^{(i-j)}$$

Case $i = j$: $r = \omega^0 = 1$ and $c_{ii} = n$.

Case $i \neq j$:

$c_{ij} = 1 + r + r^2 + \cdots + r^{n-1} = \frac{1-r^n}{1-r}$

$r^n = (\omega^{(i-j)})^n = (\omega^n)^{(i-j)} = 1^{(i-j)} \implies c_{ij} = 0$.

For C – diagonals are n and the off-diagonals are 0.
Divide by n get identity!

Inversion formula: $M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$.

Computing inverse.

FFT works with points with basic root of unity: ω or ω^{-1}
$1, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-(n-1)}$.
ω^{-1} is a primitive nth root of unity!

Evaluation: FFT(a,ω).
Interpolation: $\frac{1}{n}$ FFT(a,ω^{-1}).

$\implies O(n \log n)$ time for multiplying degree n polynomials.

Multiplying polynomials?

Evaluation: $O(n \log n)$ if choose $1, \omega, \omega^2, \ldots, \omega^{n-1}$.

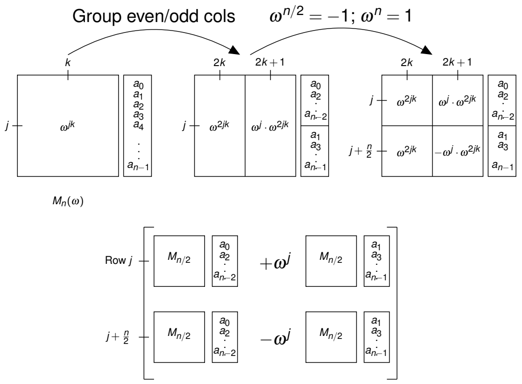
Evaluation with $FFT(\omega)$ $O(n \log n)$

Coefficient representation: $a_0, a_1, \ldots, a_{n-1}$; $+$ is $O(n)$, $*$ is $O(n^2)$ or $O(n^{\log_2 3})$ → Value representation: $A(x_0), A(x_1), \ldots, A(x_{n-1})$. $+$ is $O(n)$, $*$ is $O(n)$

Interpolation with $FFT(\omega^{-1})$ $O(n \log n)$

Interpolation: From points $A(x_0), \ldots, A(x_{n-1})$ to "function".

FFT: a closer look.



Definitive Algorithm: FFT

FFT: "$M(\omega)a$"

Idea:
"$M(\omega)a$" computed from...
"$M(\omega^2)a_o$" and "$M(\omega^2)a_e$."

FFT(a,ω):
if $\omega = 1$ return a

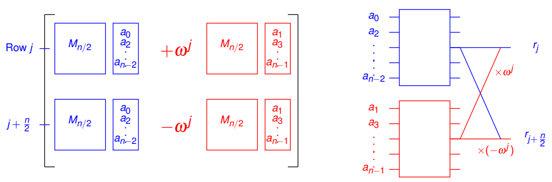
$(s_0, s_1, \ldots, s_{n/2} - 1)$ = FFT$((a_0, a_2, \ldots, a_{n-2}), \omega^2)$
$(s'_0, s'_1, \ldots, s'_{n/2} - 1)$ = FFT$((a_1, a_1, \ldots, a_{n-1}), \omega^2)$

for j = 0 to $n/2 - 1$:
$r_j = s_j + \omega^j s'_j$
$r_{j+n/2} = s_j - \omega^j s'_j$

return $(r_0, r_1, \ldots, r_{n-1})$

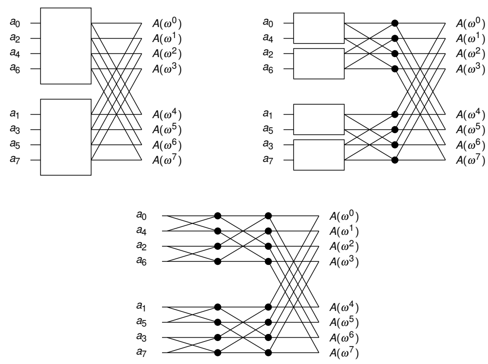
Runtime: $T(n) = 2T(n/2) + O(n)$

Unfolding FFT.



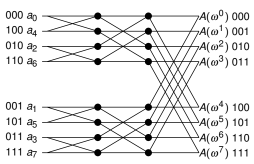
Butterfly switches!

Expanding FFT...

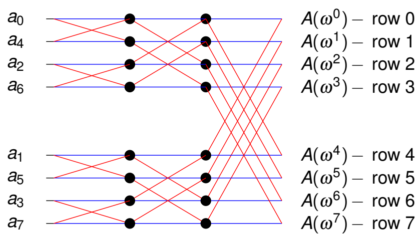


Edges from lower half of FFT have multipliers!

Order on Left

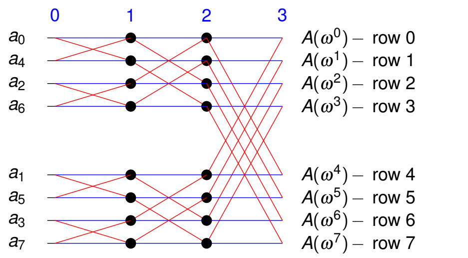


FFT Network.



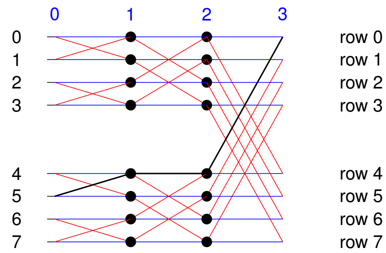
$\log N$ - levels.
N - rows.
In level i:
Row r node is connected to row r node in level $i+1$.
Row r node connected to row $r \pm 2^i$ node in level $i+1$

FFT Network.



Row r node connected to row $r \pm 2^i$ node in level $i+1$
When is it $r + 2^i$?
(A) When $\lfloor r/2^i \rfloor$ is odd.
(B) When $\lfloor r/2^i \rfloor$ is even.
(B). Red edges flip bit!

Unique Paths.



Route from input $i = 101$ to output $j = 000$?

Flip first bit. Red (cross) edge.
Keep second bit. Blue (straight) edge.
Flip third bit. Red (cross edge).

Summary.

Definitive FFT algorithm and code.

$d^{\log_2 3}$ Polynomial Multiplication - Divide and Conquer.

$$A(x) = \sum_{i=0}^{d} a_i x^i = A_L(x) + x^{d/2} A_H(x), A_L(x) := \sum_{i=0}^{d/2} a_i x^i, A_H(x) = \sum_{i=1}^{d/2} a_{i+d/2} x^i$$

$$B(x) = \sum_{i=0}^{d} b_i x^i = B_L(x) + x^{d/2} B_H(x), B_L(x) := \sum_{i=0}^{d/2} b_i x^i, B_H(x) = \sum_{i=1}^{d/2} b_{i+d/2} x^i$$

The product $A(x)B(x)$ is

$$A_L(x)B_L(x) + x^{d/2}(A_L(x)B_H(x) + A_H(x)B_L(x)) + x^d A_H(x)B_H(x)$$

Compute ...

$$A_L(x)B_L(x), \quad A_H(x)B_H(x), \quad (A_L(x) + A_H(x))(B_L(x) + B_H(x))$$

and recurse

Time is $O(d^{\log_2 3})$

FFT does better. (But this is useful to see)