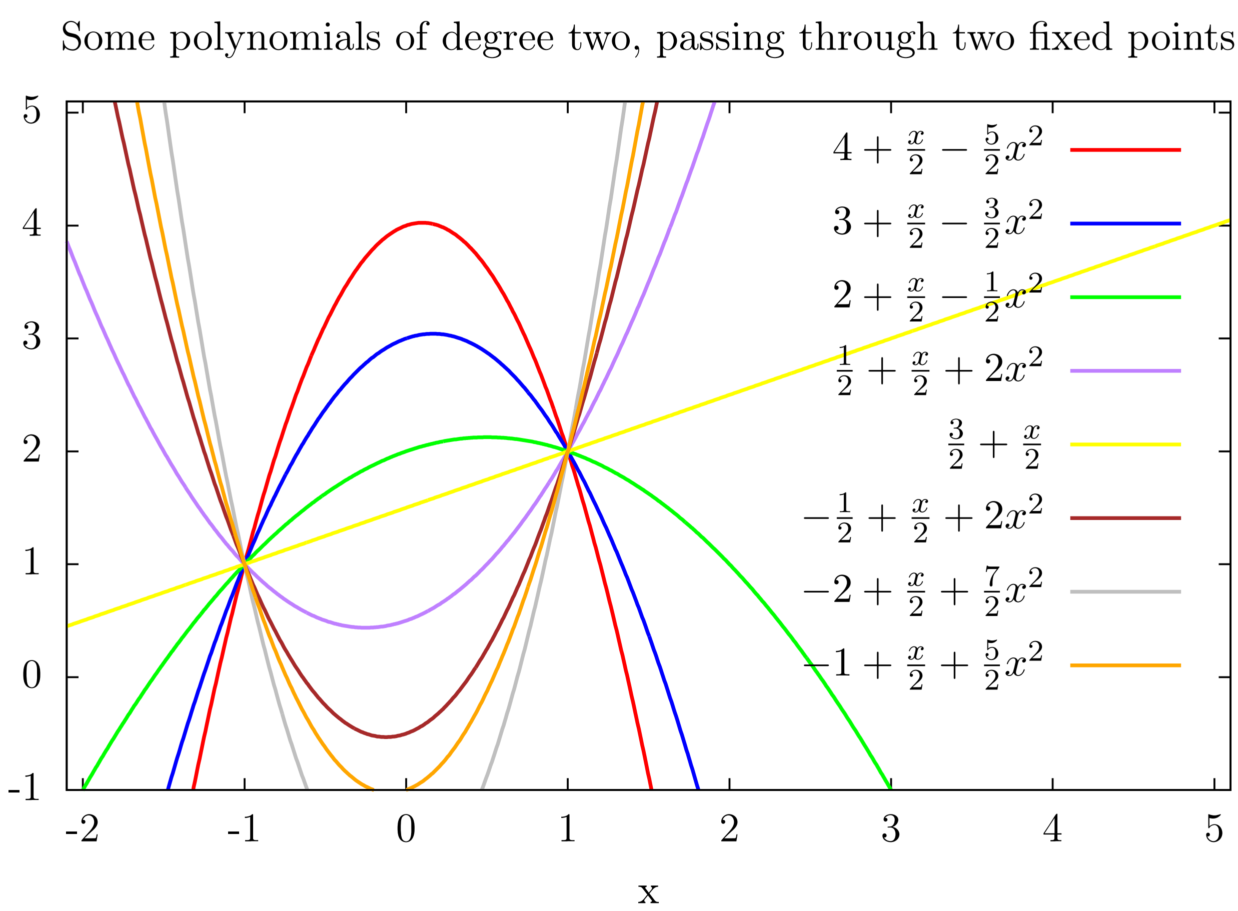
Chapter 19

Fast Fourier Transform

CS 573: Algorithms, Fall 2014
October 30, 2014

19.1 Introduction

19.1.0.1 Polynomials and point value pairs



19.1.0.2 Multiplying polynomials quickly

Definition 19.1.1. ***polynomial*** $p(x)$ of degree n:a function $p(x) = \sum_{j=0}^{n} a_j x^j = a_0 + x(a_1 + x(a_2 + \ldots + xa_n))$.

x_0: $p(x_0)$ can be computed in $O(n)$ time.
"dual" (and equivalent) representation...

Theorem 19.1.2. *For any set* $\left\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\right\}$ *of* n **point-value pairs** *such that all the* x_k *values are distinct, there is a* unique *polynomial* $p(x)$ *of degree* $n-1$*, such that* $y_k = p(x_k)$*, for* $k = 0, \ldots, n-1$*.*

19.1.0.3 Polynomial via point-value

$\left\{(x_0, y_0), (x_1, y_1), (x_2, y_2)\right\}$: polynomial through points:

$$
\begin{aligned}
p(x) = {} & y_0 \frac{\cancel{(x - x_0)}(x - x_1)(x - x_2)}{\cancel{(x_0 - x_0)}(x_0 - x_1)(x_0 - x_2)} \\
& + y_1 \frac{(x - x_0)\cancel{(x - x_1)}(x - x_2)}{(x_1 - x_0)\cancel{(x_1 - x_1)}(x_1 - x_2)} \\
& + y_2 \frac{(x - x_0)(x - x_1)\cancel{(x - x_2)}}{(x_2 - x_0)(x_2 - x_1)\cancel{(x_2 - x_2)}}
\end{aligned}
$$

$$
\begin{aligned}
p(x) = {} & y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} \\
& + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} \\
& + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}
\end{aligned}
$$

19.1.0.4 Polynomial via point-value

$\left\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\right\}$: polynomial through points:

$$
p(x) = \sum_{i=0}^{n-1} y_i \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}.
$$

ith is zero for $x = x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1}$, and is equal to y_i for $x = x_i$.

19.1.1 Polynomials: regular vs. point-value pair representation

19.1.1.1 Just because.

(A) Given n point-value pairs. Can compute $p(x)$ in $O(n^2)$ time.
(B) Point-value pairs representation: Multiply polynomials quickly!
(C) p, q polynomial of degree $n-1$, both represented by $2n$ point-value pairs

$$\left\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\right\} \quad \text{for} \quad p(x),$$
$$\text{and} \quad \left\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\right\} \quad \text{for} \quad q(x).$$

(D) $r(x) = p(x)q(x)$: product.

19.1.2 Polynomials: regular vs. point-value pair representation

19.1.2.1 Just because.

(A) In point-value representation representation of $r(x)$ is

$$\begin{aligned} &\left\{(x_0, r(x_0)), \ldots, (x_{2n-1}, r(x_{2n-1}))\right\} \\ &= \left\{\left(x_0, p(x_0)q(x_0)\right), \ldots, \left(x_{2n-1}, p(x_{2n-1})q(x_{2n-1})\right)\right\} \\ &= \left\{(x_0, y_0 y'_0), \ldots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\right\}. \end{aligned}$$

19.1.2.2 Which implies...

(A) $p(x)$ and $q(x)$: point-value pairs $\implies$ compute $r(x) = p(x)q(x)$ in linear time!
(B) ...but $r(x)$ is in point-value representation. Bummer.
(C) ...but we can compute $r(x)$ from this representation.
(D) Purpose: Translate quickly (i.e., $O(n \log n)$ time) from the standard r to point-value pairs representation of polynomials.
(E) ...and back!
(F) $\implies$ computing product of two polynomials in $O(n \log n)$ time.
(G) ***Fast Fourier Transform*** is a way to do this.
(H) choosing the x_i values carefully, and using divide and conquer.

19.2 Computing a polynomial quickly on n values

19.3 Computing a polynomial quickly on n values

19.3.1 Computing a polynomial quickly on n values

19.3.1.1 Lets just use some magic.

(A) Assume: polynomials have degree $n-1$, where $n = 2^k$.
(B) .. pad polynomials with terms having zero coefficients.

(C) ***Magic set*** of numbers: $\Psi = \{x_1, \ldots, x_n\}$.
Property: $|\mathsf{SQ}(\Psi)| = n/2$, where $\mathsf{SQ}(\Psi) = \left\{ x^2 \;\middle|\; x \in \Psi \right\}$.
(D) $|\text{square}(\Psi)| = |\Psi|/2$.
(E) Easy to find such set...
(F) **Magic**: Have this property repeatedly...
$\mathsf{SQ}(\mathsf{SQ}(\Psi))$ has $n/4$ distinct values.
(G) $\mathsf{SQ}(\mathsf{SQ}(\mathsf{SQ}(\Psi)))$ has $n/8$ values.
(H) $\mathsf{SQ}^i(\Psi)$ has $n/2^i$ distinct values.
(I) Oops: No such set of real numbers.
(J) NO SUCH SET.

19.3.2 Collapsible sets

19.3.2.1 Assume magic...

Let us for the time being ignore this technicality, and fly, for a moment, into the land of fantasy, and assume that we do have such a set of numbers, so that $|\mathsf{SQ}^i(\Psi)| = n/2^i$ numbers, for $i = 0, \ldots, k$. Let us call such a set of numbers ***collapsible***.

19.3.3 Breaking the input polynomial into...

19.3.3.1 ... two polynomials of half the degree

(A) For a set $\mathcal{X} = \{x_0, \ldots, x_n\}$ and polynomial $p(x)$, let

$$p\Big(\mathcal{X}\Big) = \left\langle \Big(x_0, p(x_0)\Big), \ldots, \Big(x_n, p(x_n)\Big) \right\rangle.$$

(B) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$, where

$$u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \qquad \text{and} \qquad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.$$

(C) all even degree terms in $u(\cdot)$, all odd degree terms in $v(\cdot)$.
(D) maximum degree of $u(y)$, $v(y)$ is $n/2$.

19.3.3.2 FFT: The dividing stage

(A) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$.
(B) Ψ: collapsible set of size n.
(C) $p(\Psi)$: compute polynomial of degree $n-1$ on n values.
(D) Decompose:

$$u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \qquad \text{and} \qquad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.$$

(E) Need to compute $u(x^2)$, for all $x \in \Psi$.
(F) Need to compute $v(x^2)$, for all $x \in \Psi$.
(G) $\mathsf{SQ}(\Psi) = \left\{ x^2 \;\middle|\; x \in \Psi \right\}$.
(H) $\implies$ Need to compute $u(\mathsf{SQ}(\Psi)), v(\mathsf{SQ}(\Psi))$.
(I) $u(\mathsf{SQ}(\Psi)), v(\mathsf{SQ}(\Psi))$: comp. poly. degree $\frac{n}{2} - 1$ on $\frac{n}{2}$ values.

19.3.3.3 FFT: The conquering stage

(A) Ψ: Collapsible set of size n.
(B) $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$.
(C) $u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i$ and $v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i$.
(D) $u(\mathsf{SQ}(\Psi)), v(\mathsf{SQ}(\Psi))$: Computed recursively.
(E) Need to compute $p(\Psi)$.
(F) For $x \in \Psi$: Compute $p(x) = u(x^2) + x \cdot v(x^2)$.
(G) Takes constant time per single element $x \in \Psi$.
(H) Takes $O(n)$ time overall.

19.3.3.4 FFT algorithm

```
FFTAlg(p, X)
        input:  p(x):  A polynomial of degree n:  p(x) = sum_{i=0}^{n-1} a_i x^i
                     X:  A collapsible set of n elements.
        output:  p(X)
    u(y) = sum_{i=0}^{n/2-1} a_{2i} y^i
    v(y) = sum_{i=0}^{n/2-1} a_{1+2i} y^i.
    Y = SQ(X) = { x^2 | x ∈ X }.
    U = FFTAlg(u, Y)            /* U = u(Y) */
    V = FFTAlg(v, Y)             /* V = v(Y) */

    Out ← ∅
    for x ∈ X do
        /* p(x) = u(x^2) + x * v(x^2), U[x^2] is the value u(x^2) */
        (x, p(x)) ← (x, U[x^2] + x · V[x^2])
        Out ← Out ∪ {(x, p(x))}

    return Out
```

19.3.4 Running time analysis...

19.3.4.1 ...an old foe emerges once again to serve

(A) $T(m, n)$: Time of computing a polynomial of degree m on n values.
(B) We have that:

$$T(n-1, n) = 2T(n/2 - 1, n/2) + O(n).$$

(C) The solution to this recurrence is $O(n \log n)$.

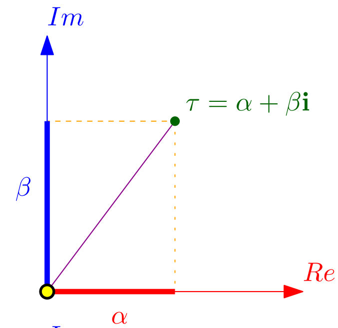
19.3.5 Generating Collapsible Sets

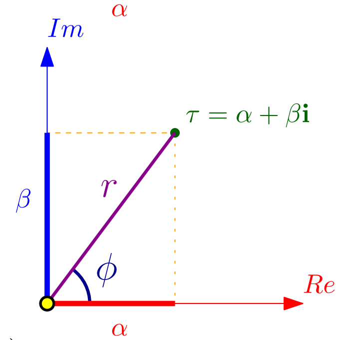
19.3.5.1 Generating Collapsible Sets

(A) How to generate collapsible sets?
(B) Trick: Use complex numbers!

19.3.5.2 Complex numbers – a quick reminder

(A) Complex number: pair (α, β) of real numbers.
Written as $\tau = \alpha + \mathbf{i}\beta$.
(B) α: ***real*** part,
β: ***imaginary*** part.
(C) $\mathbf{i}$ is the root of -1.
(D) Geometrically: a point in the complex plane:





(A) ***polar form***: $\tau = r\cos\phi + \mathbf{i}r\sin\phi = r(\cos\phi + \mathbf{i}\sin\phi)$
(B) $r = \sqrt{\alpha^2 + \beta^2}$ and $\phi = \arcsin(\beta/\alpha)$.

19.3.5.3 A useful formula: $\cos\phi + \mathbf{i}\sin\phi = e^{i\phi}$

(A) By Taylor's expansion:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots,$$
$$\text{and} \qquad e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots.$$

(B) Since $\mathbf{i}^2 = -1$:

$$\begin{aligned} e^{\mathbf{i}x} &= 1 + \mathbf{i}\frac{x}{1!} - \frac{x^2}{2!} - \mathbf{i}\frac{x^3}{3!} + \frac{x^4}{4!} + \mathbf{i}\frac{x^5}{5!} - \frac{x^6}{6!} \cdots \\ &= \cos x + \mathbf{i}\sin x. \end{aligned}$$

19.3.5.4 Back to polar form

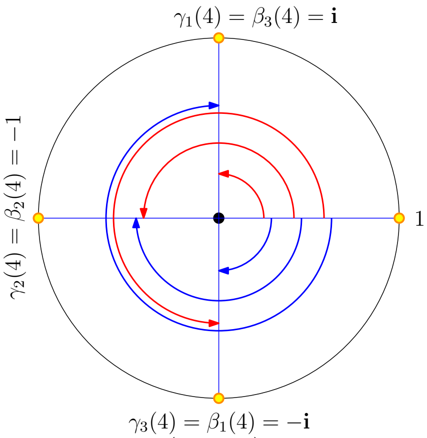
(A) ***polar form***: $\tau = r\cos\phi + \mathbf{i}r\sin\phi = r(\cos\phi + \mathbf{i}\sin\phi) = re^{i\phi}$,
(B) $\tau = re^{\mathbf{i}\phi}, \quad \tau' = r'e^{\mathbf{i}\phi'}$: complex numbers.
(C) $\tau \cdot \tau' = re^{\mathbf{i}\phi} \cdot r'e^{\mathbf{i}\phi'} = rr'e^{\mathbf{i}(\phi+\phi')}$.
(D) $e^{\mathbf{i}\phi}$ is 2π periodic (i.e., $e^{\mathbf{i}\phi} = e^{\mathbf{i}(\phi+2\pi)}$), and $1 = e^{\mathbf{i}0}$.
(E) nth root of 1: complex number τ – raise it to power n get 1.
(F) $\tau = re^{\mathbf{i}\phi}$, such that $\tau^n = r^n e^{\mathbf{i}n\phi} = e^{\mathbf{i}0}$.
(G) $\implies r = 1$, and there must be an integer j, such that

$$n\phi = 0 + 2\pi j \implies \phi = j(2\pi/n).$$

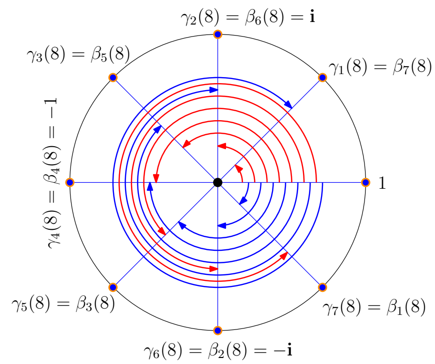
19.3.6 Roots of unity

19.3.6.1 The desire to avoid war?

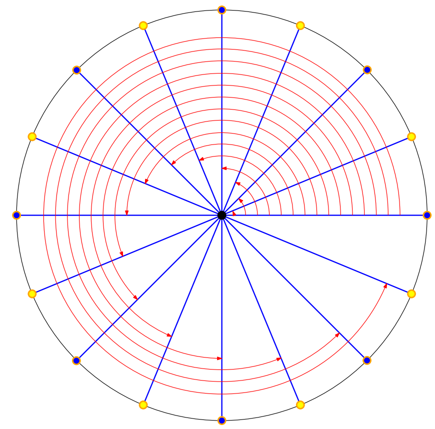
For $j = 0, \ldots, n-1$, we get the n distinct ***roots of unity***.



$(n = 4)$



$(n = 8)$


$(n = 16)$

19.3.6.2 Back to collapsible sets

(A) Can do all basic calculations on complex numbers in $O(1)$ time.
(B) Idea: Work over the complex numbers.
(C) Use roots of unity!
(D) γ: nth root of unity. There are n such roots, and let $\gamma_j(n)$ denote the jth root.

$$\gamma_j(n) = \cos((2\pi j)/n) + \mathbf{i}\sin((2\pi j)/n) = \gamma^j.$$

Let $\mathcal{A}(n) = \{\gamma_0(n), \ldots, \gamma_{n-1}(n)\}$.
(E) $|\mathsf{SQ}(\mathcal{A}(n))|$ has $n/2$ entries.
(F) $\mathsf{SQ}(\mathcal{A}(n)) = \mathcal{A}(n/2)$
(G) n to be a power of 2, then $\mathcal{A}(n)$ is the *required* collapsible set.

19.3.6.3 The first result...

Theorem 19.3.1. *Given polynomial $p(x)$ of degree n, where n is a power of two, then we can compute $p(X)$ in $O(n \log n)$ time, where $X = \mathcal{A}(n)$ is the set of n different powers of the nth root of unity over the complex numbers.*

19.3.7 Problem...

19.3.7.1 We can go, but can we come back?

(A) Can multiply two polynomials quickly
(B) by transforming them to the point-value pairs representation...
(C) over the nth roots of unity.
(D) Q: How to transform this representation back to the regular representation.
(E) A: Do some confusing math...

19.4 Recovering the polynomial

19.4.0.2 Recovering the polynomial

Think about FFT as a matrix multiplication operator.

$p(x) = \sum_{i=0}^{n-1} a_i x^i$. Evaluating $p(\cdot)$ on $\mathcal{A}(n)$:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \gamma_0 & \gamma_0^2 & \gamma_0^3 & \cdots & \gamma_0^{n-1} \\ 1 & \gamma_1 & \gamma_1^2 & \gamma_1^3 & \cdots & \gamma_1^{n-1} \\ 1 & \gamma_2 & \gamma_2^2 & \gamma_2^3 & \cdots & \gamma_2^{n-1} \\ 1 & \gamma_3 & \gamma_3^2 & \gamma_3^3 & \cdots & \gamma_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \gamma_{n-1} & \gamma_{n-1}^2 & \gamma_{n-1}^3 & \cdots & \gamma_{n-1}^{n-1} \end{pmatrix}}_{\text{the matrix } V} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix},$$

where $\gamma_j = \gamma_j(n) = (\gamma_1(n))^j$ is the jth power of the nth root of unity, and $y_j = p(\gamma_j)$.

19.4.1 The Vandermonde matrix

19.4.1.1 Because every matrix needs a name

V is the ***Vandermonde*** matrix.

V^{-1}: inverse matrix of V

Vandermonde matrix. And let multiply the above formula from the left. We get:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = V \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} \qquad \Longrightarrow \qquad \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = V^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

19.4.2 The inverse Vandermonde matrix

19.4.2.1 ..for the rescue

(A) Recover the polynomial $p(x)$ from the point-value pairs

$$\Big\{ (\gamma_0, p(\gamma_0)), (\gamma_1, p(\gamma_1)), \ldots, (\gamma_{n-1}, p(\gamma_{n-1})) \Big\}$$

(B) by doing a single matrix multiplication of V^{-1} by the vector $[y_0, y_1, \ldots, y_{n-1}]$.
(C) Multiplying a vector with n entries with $n \times n$ matrix takes $O(n^2)$ time.
(D) No benefit so far...

19.4.3 What is the inverse of the Vandermonde matrix

19.4.3.1 Vandermonde matrix is famous, beautiful and well known – a celebrity matrix

Claim 19.4.1.

$$V^{-1} = \frac{1}{n}\begin{pmatrix} 1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\ 1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\ 1 & \beta_2 & \beta_2^2 & \beta_2^3 & \cdots & \beta_2^{n-1} \\ 1 & \beta_3 & \beta_3^2 & \beta_3^3 & \cdots & \beta_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1} \end{pmatrix},$$

where $\beta_j = (\gamma_j(n))^{-1}$.

19.4.3.2 Proof

Consider the (u, v) entry in the matrix $C = V^{-1}V$. We have

$$C_{u,v} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j (\gamma_j)^v}{n}.$$

As $\gamma_j = (\gamma_1)^j$.Thus,

$$C_{u,v} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j ((\gamma_1)^j)^v}{n} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j ((\gamma_1)^v)^j}{n} = \sum_{j=0}^{n-1} \frac{(\beta_u \gamma_v)^j}{n}.$$

Clearly, if $u = v$ then

$$C_{u,u} = \frac{1}{n}\sum_{j=0}^{n-1} (\beta_u \gamma_u)^j = \frac{1}{n}\sum_{j=0}^{n-1} (1)^j = \frac{n}{n} = 1.$$

19.4.3.3 Proof continued...

If $u \neq v$ then,

$$\beta_u \gamma_v = (\gamma_u)^{-1}\gamma_v = (\gamma_1)^{-u}\gamma_1^v = (\gamma_1)^{v-u} = \gamma_{v-u}.$$

And

$$C_{u,v} = \frac{1}{n}\sum_{j=0}^{n-1} (\gamma_{v-u})^j = \frac{1}{n} \cdot \frac{\gamma_{v-u}^n - 1}{\gamma_{v-u} - 1} = \frac{1}{n} \cdot \frac{1 - 1}{\gamma_{v-u} - 1} = 0,$$

Proved that the matrix C have ones on the diagonal and zero everywhere else. ■

19.4.3.4 Recap...

(A) n point-value pairs $\{(\gamma_0, y_0), \ldots, (\gamma_{n-1}, y_{n-1})\}$: of polynomial $p(x) = \sum_{i=0}^{n-1} a_i x^i$ over nth roots of unity.
(B) Recover coefficients of polynomial by multiplying $[y_0, y_1, \ldots, y_n]$ by V^{-1}:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \frac{1}{n} \underbrace{\begin{pmatrix} 1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\ 1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\ 1 & \beta_2 & \beta_2^2 & \beta_2^3 & \cdots & \beta_2^{n-1} \\ 1 & \beta_3 & \beta_3^2 & \beta_3^3 & \cdots & \beta_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1} \end{pmatrix}}_{V^{-1}} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

(C) $W(x) = \sum_{i=0}^{n-1} (y_i/n) x^i$: $\qquad a_i = W(\beta_i)$.

19.4.3.5 Recovering continued...

(A) recover coefficients of $p(\cdot)$...
(B) ... compute $W(\cdot)$ on n values: $\beta_0, \ldots, \beta_{n-1}$.
(C) $\{\beta_0, \ldots, \beta_{n-1}\} = \{\gamma_0, \ldots, \gamma_{n-1}\}$.
(D) Indeed $\beta_i^n = (\gamma_i^{-1})^n = (\gamma_i^n)^{-1} = 1^{-1} = 1$.
(E) Apply the **FFTAlg** algorithm on $W(x)$ to compute $a_0, \ldots, a_{n-1}$.

19.4.3.6 Result

Theorem 19.4.2. *Given n point-value pairs of a polynomial $p(x)$ of degree $n-1$ over the set of n powers of the nth roots of unity, we can recover the polynomial $p(x)$ in $O(n \log n)$ time.*

Theorem 19.4.3. *Given two polynomials of degree n, they can be multiplied in $O(n \log n)$ time.*

19.5 Convolutions

19.5.0.7 Convolutions

(A) Two vectors: $A = [a_0, a_1, \ldots, a_n]$ and $B = [b_0, \ldots, b_n]$.
(B) ***dot product*** $A \cdot B = \langle A, B \rangle = \sum_{i=0}^{n} a_i b_i$.
(C) A_r: shifting of A by $n - r$ locations to the left
(D) Padded with zeros:, $a_j = 0$ for $j \notin \{0, \ldots, n\}$).
(E) $A_r = \left[a_{n-r}, a_{n+1-r}, a_{n+2-r}, \ldots, a_{2n-r}\right]$
where $a_j = 0$ if $j \notin \left[0, \ldots, n\right]$.
(F) **Observation**: $A_n = A$.

19.5.0.8 Example of shifting

Example 19.5.1. For $A = [3, 7, 9, 15]$, $n = 3$
$A_2 = [7, 9, 15, 0]$,
$A_5 = [0, 0, 3, 7]$.

19.5.0.9 Definition

Definition 19.5.2. Let $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$, for $i = 0, \ldots, 2n$. The vector $[c_0, \ldots, c_{2n}]$ is the ***convolution*** of A and B.

question How to compute the convolution of two vectors of length n?

19.5.0.10 Convolution via multiplication polynomials

(A) $p(x) = \sum_{i=0}^{n} \alpha_i x^i$, and $q(x) = \sum_{i=0}^{n} \beta_i x^i$.
(B) Coefficient of x^i in $r(x) = p(x)q(x)$ is $d_i = \sum_{j=0}^{i} \alpha_j \beta_{i-j}$.
(C) Want to compute $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$.
(D) Set $\alpha_i = a_i$ and $\beta_l = b_{n-l-1}$.

19.5.0.11 Convolution by example

(A) Consider coefficient of x^2 in product of $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ and $q(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$.
(B) Sum of the entries on the anti diagonal:

	a_0+	$a_1 x$	$+a_2 x^2$	$+a_3 x^3$
b_0			$a_2 b_0 x^2$	
$+b_1 x$		$a_1 b_1 x^2$		
$+b_2 x^2$	$a_0 b_2 x^2$			
$+b_3 x^3$				

(C) entry in the ith row and jth column is $a_i b_j$.

19.5.0.12 Convolution

Theorem 19.5.3. *Given two vectors $A = [a_0, a_1, \ldots, a_n]$, $B = [b_0, \ldots, b_n]$ one can compute their convolution in $O(n \log n)$ time.*

Proof: Let $p(x) = \sum_{i=0}^{n} a_{n-i} x^i$ and let $q(x) = \sum_{i=0}^{n} b_i x^i$. Compute $r(x) = p(x)q(x)$ in $O(n \log n)$ time using the convolution theorem. Let $c_0, \ldots, c_{2n}$ be the coefficients of $r(x)$. It is easy to verify, as described above, that $[c_0, \ldots, c_{2n}]$ is the convolution of A and B. ■