Reading: 5.6.

Last time:

- Divide and Conquer
- Recurrences
- Mergesort, Integer Multiply

Today:

- Polynomial Multiplication
- Fast Fourier Transform

Convolution and Polynomial Multiplication

Example:

$$A(x) = -2 + 2x$$
$$B(x) = 3/2 - x/2$$
$$C(c) = A(x) \cdot B(x) = -3 + 4x - x^2$$

Fact: let
$$A(x) = a_0 + a_1 x + \ldots a_{n-1} x^{n-1}$$
$$B(x) = b_0 + b_1 x + \ldots b_{n-1} x^{n-1}$$
be degree $n-1$ polynomials. Then,
$$\begin{aligned} C(x) &= A(x) \cdot B(x) \\ &= c_0 + c_1 x + \ldots c_{2n-2} x^{n-2} \end{aligned}$$
with
$$c_k = \sum_{i,j \,:\, i+j=k} a_i b_j$$

Def: $\mathbf{a} = (a_0, \ldots, a_{n-1})$ is an n-vector.

Def: $\mathbf{c}$ (above) is the **convolution** of $\mathbf{a}$ with $\mathbf{b}$, denoted $\mathbf{c} = \mathbf{a} * \mathbf{b}$.

Def: for $\mathbf{a}$ and $\mathbf{b}$, the **pointwise vector product** is $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ with $c_k = a_j \cdot b_j$.

[[*what are runtimes of trivial algorithms for convolution and vector product?*]]

Runtimes:

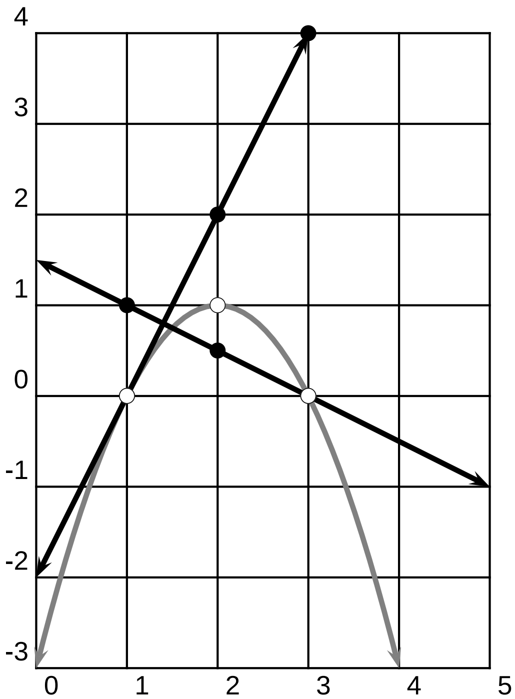
- pointwise product: $O(n)$.
- convolution: $O(n^2)$. [[*optimal?*]]

[[*can we do convolution faster?*]]

Polynomial multiplication via evaluation

Fact: a degree $n-1$ polynomial is uniquely determined by n points

Example: $A(x) = -2 + 2x$ determined by $(1, 0)$, $(2, 2)$.



[[*use fact to do poly mult another way*]]

Example:

$$A(x) = -2 + 2x$$

$$B(x) = 3/2 - x/2$$

Evaluate:

x	1	2	3
$A(x)$	0	2	4
$B(x)$	3/2	1/2	0

Multiply:

$C(x) = A(x)B(x)$	0	1	0

$\left[\left[\right.\right.$*What degree 2 poly goes through these points?*$\left.\left.\right]\right]$

Interpolate: $C(x) = -3 + 4x - x^2$.

[[*last step comes from "Algebra 2"*]]

Conclusion: Given

- x-coordinates $x_0, \ldots, x_{n-1}$
- function values $A_0, \ldots, A_{n-1}$

  (with $A_i = A(x_i)$)

there is correspondence:

$$\underset{\text{coefficients}}{\mathbf{a} = (a_0, \ldots, a_{n-1})} \underset{\text{interpolate}}{\overset{\text{evaluate}}{\rightleftarrows}} \underset{\text{values}}{\mathbf{A} = (A_0, \ldots, A_{n-1})}$$

Algorithm: Polynomial Mult (degree $n-1$)

1. choose $\mathbf{x}$ as $2n-1$ points $x_0, \ldots, x_{2n-2}$
2. evaluate on $\mathbf{x}$: $\mathbf{a}, \mathbf{b} \Rightarrow \mathbf{A}, \mathbf{B}$.
3. pointwise multiply: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$.
4. interpolate: $\mathbf{C} \Rightarrow \mathbf{c}$.

Runtime: $T(n) = O(n^2)$

$\left[\left[\right.\right.$*e.g., evaluate degree n poly on $2n$ points is $O(n^2)$. need a better idea*$\left.\left.\right]\right]$

Idea: Choose **x** to make evaluation/interpolation faster.

Fast Fourier Transform

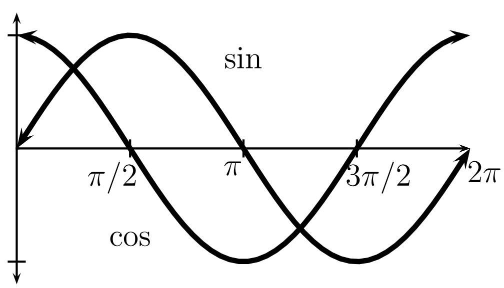
[[*crazy awesome math!!*]]
[[*take EECS 222 Fundamentals of Signals and Systems*]]

Fact (Euler's Formula):

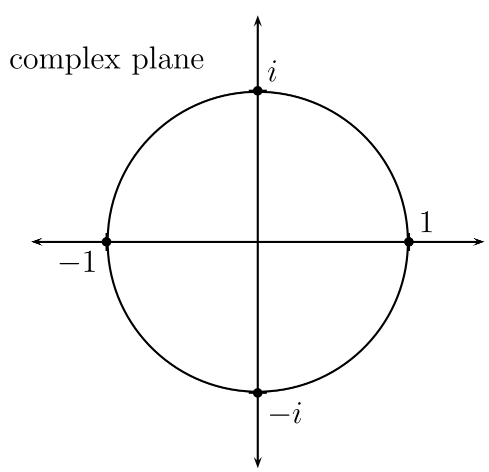
$$e^{i\theta} = \cos\theta + i\sin\theta$$

Proof: E.g., via Taylor series, see Wikipedia.

Recall: trigonometry



Example: Evaluate $e^{i\theta}$ at $\theta = \{0, \pi/2, \pi, 3\pi/2, 2\pi\}$



Fact: multiplying $\equiv$ adding angles

$$e^{i\theta_1}e^{i\theta_2} = e^{i(\theta_1+\theta_2)}$$

Fact (Euler's Identity): $e^{i2\pi} = 1$

[[*from Euler's formula*]]

Def: n**th roots of unity** are $e^{ij2\pi/n}$ for $j = 0, \ldots, n-1$.

Fact: nth roots of unity are solutions to $x^n = 1$.

[[*intuition: multiplying = adding angles*]]

Proof: $(e^{ij2\pi/n})^n = e^{ij2\pi} = (e^{i2\pi})^j = 1^j = 1$.

Idea: use $2n$th roots of unity as $x_0, \ldots, x_{2n-1}$.

Problem: Fourier Transform

Input: coefficients of degree $n-1$ poly.

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

Output: $A_0, \ldots, A_{n-1}$

with $A_j = A(e^{ij2\pi/n})$.

Divide and Conquer FFT

[[subproblem?]]

[[break $A(\cdot)$ into even and odd terms]]

Idea: write $A(x) = \underbrace{A''(x^2)}_{\text{evens}} + \underbrace{xA'(x^2)}_{\text{odds}}$

- $A'(\cdot)$, $A''(\cdot)$, degree $n/2 - 1$ polys on x^2.
- x^2 on nth roots of unity

  $\equiv x$ on $n/2$th roots of unity

Formally:

- $A''(x) = a_0 + a_2 x + \cdots a_{n-2} x^{n/2-1}$
- $A'(x) = a_1 + a_3 x + \cdots a_{n-1} x^{n/2-1}$

and

?

$$\begin{aligned} A(e^{ij2\pi/n}) &= A''(e^{ij2\pi/n^2}) + e^{ij2\pi/n} A'(e^{ij2\pi/n^2}) \\ &= A''(\underbrace{e^{ij\pi/n}}) + e^{ij2\pi/n} A'(\underbrace{e^{ij\pi/n}}) \end{aligned}$$

$n/2$th root of unity

Subproblems: evaluate $n/2 - 1$ degree polys $A'(\cdot)$, $A''(\cdot)$ on $n/2$th roots of unity.

Algorithm: FFT (evaluates $n-1$ degree poly on nth roots of unity)

0. if $n = 1$, return $A_0 = a_o$.
1. divide $\mathbf{a}$ into even & odd coefs, $\mathbf{a}'$ and $\mathbf{a}''$
2. $\mathbf{A}' = FFT(\mathbf{a}')$; $\mathbf{A}'' = FFT(\mathbf{a}'')$.
3. for each nth root of unity $e^{ij2\pi/n}$:

?

$$A_j = A''_{(j \mod 2)} + e^{ij2\pi/n} A'_{(j \mod 2)}$$

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

$\Rightarrow T(n) = O(n \log n)$.

Poly Mult w. FFT

Claim: can de-FFT ($\mathbf{A} \Rightarrow \mathbf{a}$) with similar divide and conquer alg.

Proof: See text.

Algorithm: Poly Mult w. FFT

[[use FFT/de-FFT for evaluate/interpolate in poly mult algorithm]]

1. take $2n$ bit FFTs: $\mathbf{a}, \mathbf{b} \Rightarrow \mathbf{A}, \mathbf{B}$
2. pointwise multiply: $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$
3. take $2n$ bit de-FFT: $\mathbf{C} \Rightarrow \mathbf{c}$.

Runtime: $T(n) = O(n \log n)$

Note: FFT with complex roots of unity can have numerical errors, with integer coefs, round solution to be integers.

[[Also, can get roots of units via number theory, e.g., integers modulo a prime.]]

Note: Can use FFT to integer multiply in $O(n \log^2 n)$