6.046 | Lecture 3 | Feb. 12, 2015

TODAY: Fast Fourier Transform (FFT)

- polynomial operations vs. representations
- divide & conquer algorithm
- collapsing samples / roots of unity
- FFT, IFFT, & polynomial multiplication

degree(A)

Polynomial: $A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{\boxed{n-1}}$

$$= \sum_{k=0}^{n-1} a_k x^k$$

$$= \langle a_0, a_1, a_2, \ldots, a_{n-1} \rangle$$ (coefficient vector)

Operations on polynomials:

① evaluation: poly. $A(x)$ & number $x_0 \to A(x_0)$

- Horner's Rule $\Rightarrow O(n)$ time → #arithmetic ops.

$$A(x) = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1})\cdots))$$

② addition: polys. $A(x)$ & $B(x) \to C(x) = A(x) + B(x)\ \forall x$

- $O(n)$ time: i.e. $c_k = a_k + b_k$

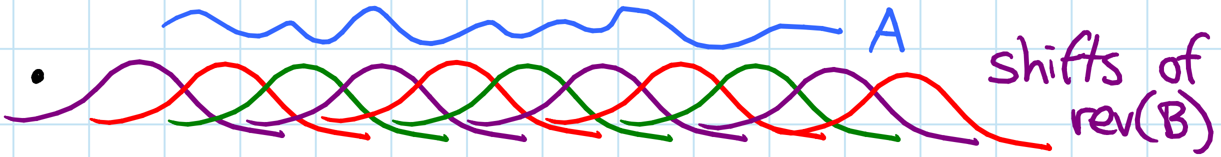
③ multiplication: polys. $A(x)$ & $B(x) \to C(x) = A(x) \cdot B(x)\ \forall x$

- i.e. $c_k = \sum_{j=0}^{k} a_j b_{k-j}$ for $0 \le k \le 2(n-1)$

(degree doubles)

= convolution of vectors A & reverse(B)

↳ inner product of all relative shifts



– useful for smoothing etc. in digital signal processing

– $O(n^2)$ time ☹

– $O(n^{\lg 3})$ or even $O(n^{1+\varepsilon})$ $\forall \varepsilon > 0$ via Strassen-like divide & conquer tricks

– TODAY: $O(n \lg n)$ time!

Representations of polynomials:

Ⓐ coefficient vector ("monomial basis")

Ⓑ roots + scale: (Fundamental Theorem of Algebra)

$$A(x) = (x - r_0) \cdot (x - r_1) \cdot \cdots \cdot (x - r_{n-1}) \cdot c$$

– but impossible to find exact roots with $+, -, *, /, \sqrt[k]{\ }$

⇒ addition hard/impossible

– multiplication: concatenate roots

– evaluate in $O(n)$

Ⓒ samples: $(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})$ with $A(x_i) = y_i \ \forall i$ & x_i's distinct uniquely determine degree-$(n-1)$ polynomial A [Lagrange & Fundamental Theorem of Algebra]

– add/multiply each y_i (assuming x_i's match)

– evaluate requires interpolation…

Algorithms $\boxed{\text{vs.}}$	Representations: Ⓐ coefficients	Ⓑ roots	Ⓒ samples
① evaluation	$O(n)$	$O(n)$	$\boxed{O(n^2)}$
② addition	$O(n)$	$\boxed{\infty}$	$O(n)$
③ multiplication	$\boxed{O(n^2)}$	$O(n)$	$O(n)$

TODAY: almost best of all worlds by converting coefficients ⟺ samples in $O(n \lg n)$ time

Matrix view:
[18.06]

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{pmatrix}}_{\text{Vandermonde matrix } V:\ v_{jk} = x_j^k} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

- coeff. → samples = matrix-vector product $V \cdot A$
  - $O(n^2)$

EVALUATION
INTERPOLATION

- samples → coeff. = matrix-vector solve $V \backslash Y$ Matlab
  - $O(n^3)$ via Gaussian elimination
  - $O(n^2)$ via matrix-vector product $V^{-1} \cdot Y = A$ precompute
- to do better than $\Theta(n^2)$, we will choose special values for $x_0, x_1, \ldots, x_{n-1}$ (so far we've only assumed they're distinct)

Divide & conquer algorithm: $A(x)$ for $x \in X$

① divide into even & odd coefficients:

$$A_{even}(x) = \sum_{k=0}^{\lceil n/2-1 \rceil} a_{2k} x^k = \langle a_0, a_2, a_4, \dots \rangle$$

$$\& \quad A_{odd}(x) = \sum_{k=0}^{\lfloor n/2-1 \rfloor} a_{2k+1} x^k = \langle a_1, a_3, a_5, \dots \rangle$$

② recursively conquer $A_{even}(y)$ for $y \in X^2$ & $A_{odd}(y)$ for $y \in X^2$

$\{x^2 \mid x \in X\}$

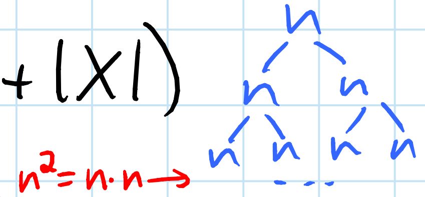
③ combine:

$$A(x) = A_{even}(x^2) + x \cdot A_{odd}(x^2) \text{ for } x \in X$$

$$T(n, |X|) = 2 \cdot T(n/2, |X|) + O(n + |X|)$$

↳ $|A|$

$$= O(n^2)$$ ☹



(courtesy of Jeff Erickson's lecture notes)

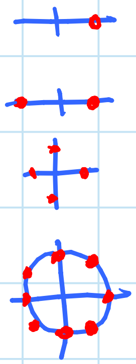
Collapsing set X if $|X^2| = |X|/2$ & X^2 is collapsing (recursively)

or $|X| = 1$ (base case)

($\Rightarrow |X| = 2^\ell$)

$$\Rightarrow T(n) = 2 \cdot T(n/2) + O(n) = O(n \lg n)$$ ☺

Constructing collapsing sets via √'s:



⓪ $\{1\}$ (any nonzero starting number)

① $\{1, -1\}$

② $\{1, -1, i, -i\}$ complex numbers!

③ $\{1, -1, \pm\frac{\sqrt{2}}{2}(1+i), \pm\frac{\sqrt{2}}{2}(-1+i)\}$ ← solve $(p+qi)^2 = i$

⋮

↳ on a circle!

nth roots of unity: n x's such that $x^n = 1$

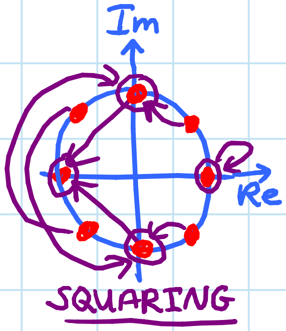
- uniformly spaced around unit circle in complex plane (& including 1)
- $(\cos\theta, \sin\theta) = \cos\theta + i\sin\theta = e^{i\theta}$ ↖ Euler's Formula

  for $\theta = 0, \frac{1}{n}\tau, \frac{2}{n}\tau, \dots, \frac{n-1}{n}\tau$

  ↳ 2π (full circle)
- $n = 2^\ell \Rightarrow$ collapsing:
  - $(e^{i\theta})^2 = e^{i(2\theta)} = e^{i(2\theta \bmod \tau)}$

    ↖ $(e^{i\tau} = 1)$

  $\Rightarrow$ even nth roots of unity = $(n/2)$nd roots of unity } repeated twice



Discrete Fourier Transform (DFT)

$= \underbrace{A \to A^* = V \cdot A}_{\text{coeffs.} \to \text{samples}}$ for $x_k = e^{i\tau k/n}$ & $n = 2^\ell$

i.e. $a_k^* = \sum_{j=0}^{n-1} \underbrace{e^{i\tau jk/n}}_{V_{kj} = V_{jk}} \cdot a_j$ [Clairaut 1754]

Fast Fourier Transform (FFT)

= $O(n \lg n)$ divide & conquer alg. for DFT

[used by Gauss circa 1805, (periodic asteroid orbits) popularized by Cooley & Tukey in 1965] (detecting Soviet nuclear tests from offshore readings)

- practical implementation: FFTW [Frigo & Johnson @ MIT]
- also often implemented directly in hardware (for fixed n)

Inverse (Discrete) Fourier Transform $= A^* \to V^{-1}\cdot A^*$

- in fact $V^{-1} = \overline{V}/n$ $\quad(\overline{p+qi} = p-qi)$

  i.e. $P = V\cdot\overline{V} = n\cdot I$
- proof: $p_{jk} = (\text{row } j \text{ of } V)\cdot(\text{col. } k \text{ of } \overline{V})$

$$= \sum_{m=0}^{n-1} e^{i\tau jm/n}\cdot\overline{e^{i\tau mk/n}} \quad \text{cw} \to \text{ccw}$$

$$= \sum_{m=0}^{n-1} e^{i\tau jm/n}\cdot e^{-i\tau mk/n}$$

$$= \sum_{m=0}^{n-1} e^{i\tau m(j-k)/n}$$

  - if $j=k$: $p_{jk} = \sum_{m=0}^{n-1} 1 = n$
  - else: geometric series:

$$p_{jk} = \sum_{m=0}^{n-1}\left(e^{i\tau(j-k)/n}\right)^m = \frac{\left(\overset{=1}{e^{i\tau}}{}^{(j-k)\cancel{/n}}\right)^{\cancel{n}}-1}{e^{i\tau(j-k)/n}-1} = 0$$

- so IDFT $= A \to V\cdot A$ for $x_k = e^{-i\tau k/n}$
- IFFT algorithm analogous

Fast polynomial multiplication: $C(x) = A(x)\cdot B(x)$

$O(n \lg n)$

- $A^* = \text{FFT}(A)$
- $B^* = \text{FFT}(B)$
- $c_k^* = a_k^*\cdot b_k^*$ for $k = 0, 1, \ldots, n-1$
- $C = \text{IFFT}(C^*)$

__Application__: Fourier (frequency) space

- A^* is complex
- $|a_k^*|$ = amplitude of frequency-k signal
- $\arg(a_k^*) = \text{angle}(a_k^*)$ = phase shift

__Example__: sound [Adobe Audition, Audacity, etc]

- high-pass filter = zero out high frequencies
- low – – · ——— · low — —
- pitch shift = shift frequency vector
- used in MP3 compression etc.