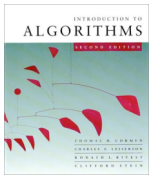
Introduction to Algorithms

6.046J/18.401J



Lecture 20

Prof. Piotr Indyk

Fast Fourier Transform

- Discrete Fourier Transform (DFT):
  - Given: coefficients of a polynomial
    $a(x)=a_0+a_1x+a_2x^2+\ldots+a_{n-1}x^{n-1}$
  - Goal: compute
    $a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1}),$
    where $\omega_n = e^{2\pi i/n} = \cos(2\pi/n)+i\sin(2\pi/n)$
- Challenge: perform DFT in $O(n \log n)$ time

Motivation I: 6.003

- FFT is essential for digital signal processing
  - $a_0, a_1, \ldots a_{n-1}$: discretized signal in the time domain
  - $a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1})$: signal in the frequency domain
  - FFT enables quick conversion from one domain to the other
- Used in compact disks, digital cameras, synthesizers, etc, etc.

Example application: SETI

- Searching For Extraterrestrial Intelligence:

- SETI@home: "At each drift rate, the client searches for signals at one or more bandwidths between 0.075 and 1,221 Hz. This is accomplished by using FFTs of length 2^n (n=3,…,17) to transform the data into a number of time-order spectra"

Motivation II: Computer Science

- We will see how to multiply two n-degree polynomials in $O(n \log n)$ time using FFT
- Multiplication of polynomials ⇒ multiplication of large integers - crypto
- Other surprising applications:
  - Pattern matching with wildcards (recitations)

FFT

- Very elaborate implementations
  (e.g., FFTW, "the Fastest Fourier Transform in the West", done at MIT)
- Hardware implementations

FFT

DFT: Preliminaries

- Goal: we want $a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1})$ where ω_n is the "principal n^{th} root of unity". I.e., $(\omega_n^j)^n=1$ for all $j=0,\ldots,n-1$
- We will work in the field of complex numbers where

$$\omega_n = e^{2\pi i/n} = \cos(2\pi/n)+i\sin(2\pi/n)$$

- Then

$$(\omega_n^j)^n = e^{2\pi ij} = \cos(2\pi j)+i\sin(2\pi j) = 1$$

Main Lemma

- If $n>0$ is even, then the squares of the n-th roots of unity are the $n/2$-th roots of unity, i.e.,

$$(\omega_n^j)^2 = \omega_{n/2}^j \quad \text{for } j=0\ldots n-1$$

- Note that $\omega_{n/2}^j = \omega_{n/2}^{j+n/2}$ for $j=0\ldots n/2-1$
- Therefore:

$$\{(\omega_n^0)^2, \ldots, (\omega_n^{n/2-1})^2, (\omega_n^{n/2})^2 \ldots, (\omega_n^{n-1})^2\} = \{\omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1}\}$$

- Proof: $(\omega_n^j)^2 = e^{2*2\pi ij/n} = e^{2\pi ij/(n/2)} = \omega_{n/2}^j$

FFT

- Divide and conquer algorithm
- Idea: divide $a(x)$ into $a^{[0]}(x)$ and $a^{[1]}(x)$:
  - $a^{[0]}(x) = a_0+a_2x+a_4x^2+\ldots+a_{n-2}x^{n/2-1}$
  - $a^{[1]}(x) = a_1+a_3x+a_5x^2+\ldots+a_{n-1}x^{n/2-1}$
- Note that

$$a(x) = a^{[0]}(x^2)+x\,a^{[1]}(x^2)$$

$$a(\omega_n^j) = a^{[0]}((\omega_n^j)^2)+\omega_n^j\,a^{[1]}((\omega_n^j)^2)$$

FFT: the algorithm

- Want: evaluate the polynomial $a(x)$ at points

$$\{\omega_n^0, \ldots, \omega_n^{n-1}\}$$

- FFT algorithm:
  - Recursively evaluate $a^{[0]}(x)$ and $a^{[1]}(x)$ at $P=\{(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2\} = \{\omega_{n/2}^0, \ldots, \omega_{n/2}^{n/2-1}\}$ — 2T(n/2)
  - Compute $a(\omega_n^j) = a^{[0]}((\omega_n^j)^2)+\omega_n^j\,a^{[1]}((\omega_n^j)^2)$ for $j=0\ldots n-1$ — O(n)
- Time: $T(n)=2T(n/2)+O(n) \Rightarrow T(n)=O(n \log n)$
- This is the whole FFT!

Comments

- We needed to assume that n is a power of 2
- This assumption cannot be easily removed
  - We need algebraic properties of ω_n

Implications

Inverse DFT

- We showed how to transform quickly
  $a_0, a_1, \ldots a_{n-1} \rightarrow a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1})$
- The opposite transformation, i.e.,
  $a(\omega_n^0), a(\omega_n^1), \ldots, a(\omega_n^{n-1}) \rightarrow a_0, a_1, \ldots a_{n-1}$
  is called "the Inverse DFT"
- Fact: Slightly modified FFT solves the inverse DFT in $O(n \log n)$ time [see CLRS, p. 836 for an optional proof]

Polynomial multiplication

- Input: $n-1$-degree polynomials $a(x)$, $b(x)$
- Output: a polynomial $c(x)=a(x)*b(x)$
  - $c_i=a_0*b_i + \ldots + a_i*b_0$
- Have seen sub-quadratic time algoritms
- Claim: there is $O(n \log n)$ time algorithm

FFT-based poly multiplication

- Extend a,b to degree $2n-1$ (by adding 0's)
- Using FFT, compute:
  - $a(\omega_{2n}^0), a(\omega_{2n}^1), \ldots, a(\omega_{2n}^{2n-1})$
  - $b(\omega_{2n}^0), b(\omega_{2n}^1), \ldots, b(\omega_{2n}^{2n-1})$
- Compute $c(\omega_{2n}^j)= a(\omega_{2n}^j)*b(\omega_{2n}^j)$, $j=0..2n-1$
- Compute $c_0, c_1, \ldots c_{2n-1}$ using inverse FFT
- Correctness: if we fix the values of a d-degree polynomial at $d+1$ points, then the polynomial is unique
  - E.g., there is only one line passing through two distinct points
  - See CLRS, p. 825 for an optional proof

Conclusions

- DFT + inverse DFT in $O(n \log n)$ time
- LOTS of applications!