CSE 417: Algorithms and Computational Complexity

Winter 2001

Lecture 14

Instructor: Paul Beame

1

Multiplying Faster

- On the first HW you analyzed our usual algorithm for multiplying numbers
  - $\Theta(n^2)$ time
- We can do better!
  - We'll describe the basic ideas by multiplying polynomials rather than integers
  - Advantage is we don't get confused by worrying about carries at first

2

Note on Polynomials

- These are just formal sequences of coefficients so when we show something multiplied by x^k it just means shifted k places to the left

3

Polynomial Multiplication

- Given:
  - Degree $m-1$ polynomials P and Q
    - $P = a_0 + a_1 x + a_2 x^2 + \ldots + a_{m-2}x^{m-2} + a_{m-1}x^{m-1}$
    - $Q = b_0 + b_1 x + b_2 x^2 + \ldots + b_{m-2}x^{m-2} + b_{m-1}x^{m-1}$
- Compute:
  - Degree $2m-2$ Polynomial $P\,Q$
  - $P\,Q = a_0b_0 + (a_0b_1+a_1b_0)\,x + (a_0b_2+a_1b_1+a_2b_0)\,x^2 + \ldots + (a_{m-2}b_{m-1}+a_{m-1}b_{m-2})\,x^{2m-3} + a_{m-1}b_{m-1}\,x^{2m-2}$
- Obvious Algorithm:
  - Compute all a_ib_j and collect terms
  - $\Theta(n^2)$ time

4

Naive Divide and Conquer

- Assume $m=2k$
  - $P = (a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}) + (a_k + a_{k+1} x + \ldots + a_{m-2}x^{k-2} + a_{m-1}x^{k-1})\,x^k$
    $= P_0 + P_1 x^k$
  - $Q = Q_0 + Q_1 x^k$
- $P\,Q = (P_0+P_1x^k)(Q_0+Q_1x^k)$
  $= P_0Q_0 + (P_1Q_0+P_0Q_1)x^k + P_1Q_1x^{2k}$
- 4 sub-problems of size $k=m/2$ plus linear combining
  - $T(m)=4T(m/2)+cm$
  - Solution $T(m) = O(m^2)$

5

Karatsuba's Algorithm

- A better way to compute the terms
  - Compute
    - P_0Q_0
    - P_1Q_1
    - $(P_0+P_1)(Q_0+Q_1)$ which is $P_0Q_0+P_1Q_0+P_0Q_1+P_1Q_1$
  - Then
    - $P_0Q_1+P_1Q_0 = (P_0+P_1)(Q_0+Q_1) - P_0Q_0 - P_1Q_1$
  - 3 sub-problems of size $m/2$ plus $O(m)$ work
    - $T(m) = 3\,T(m/2) + cm$
    - $T(m) = O(m^\alpha)$ where $\alpha = \log_2 3 = 1.59...$

6

Karatsuba's Algorithm Alternative

- Compute
  - $A_0=P_0Q_0$
  - $A_1=(P_0+P_1)(Q_0+Q_1)$, i.e. $P_0Q_0+P_1Q_0+P_0Q_1+P_1Q_1$
  - $A_{-1}=(P_0-P_1)(Q_0-Q_1)$, i.e. $P_0Q_0-P_1Q_0-P_0Q_1+P_1Q_1$
- Then
  - $P_1Q_1=(A_1+A_{-1})/2-A_0$
  - $P_0Q_1+P_1Q_0=(A_1-A_{-1})/2$
- 3 sub-problems of size $m/2$ plus $O(m)$ work
  - $T(m)=3\,T(m/2)+cm$
  - $T(m)=O(m^\alpha)$ where $\alpha=\log_2 3=1.59...$

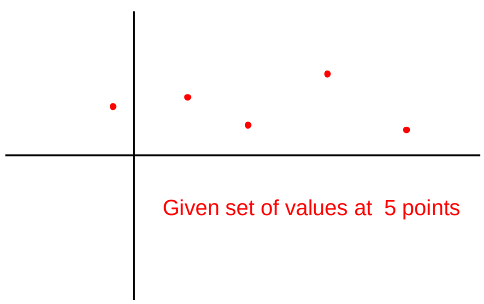
7

What Karatsuba's Algorithm did

- For $y=x^k$ we wanted to compute $P(y)Q(y)=(P_0+P_1y)(Q_0+Q_1y)$
- We evaluated
  - $P(0)=P_0$ $Q(0)=Q_0$
  - $P(1)=P_0+P_1$ and $Q(1)=Q_0+Q_1$
  - $P(-1)=P_0-P_1$ and $Q(-1)=Q_0-Q_1$
- We multiplied $P(0)Q(0)$, $P(1)Q(1)$, $P(-1)Q(-1)$
- We then used these 3 values to figure out what the degree 2 polynomial $P(y)Q(y)$ was
  - Interpolation

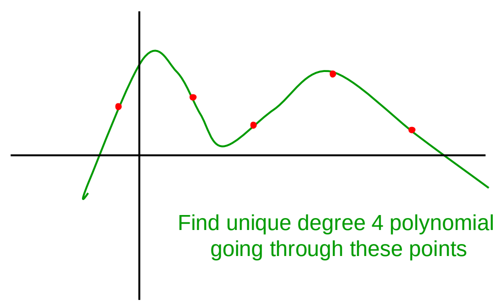
8

Interpolation



9

Interpolation



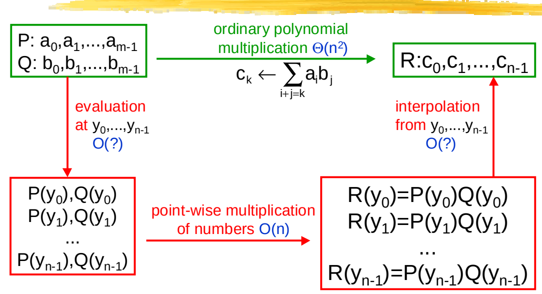
10

Multiplying Polynomials by Evaluation & Interpolation

- Any degree $n-1$ polynomial $R(y)$ is determined by $R(y_0)$, ... $R(y_{n-1})$ for any n distinct $y_0,\ldots,y_{n-1}$
- To compute PQ (assume degree at most $n-1$)
  - Evaluate $P(y_0),\ldots,P(y_{n-1})$
  - Evaluate $Q(y_0),\ldots,Q(y_{n-1})$
  - Multiply values $P(y_i)Q(y_i)$ for $i=0,\ldots,n-1$
  - Interpolate to recover PQ

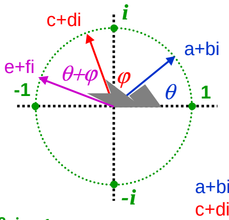
11

Multiplying Polynomials by Evaluation & Interpolation



12

Complex Numbers $i^2 = -1$



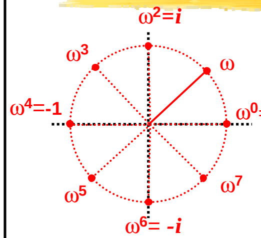
To multiply complex numbers:
1. add angles
2. multiply lengths

e+fi = (a+bi)(c+di)

$a+bi = \cos\theta + i\sin\theta = e^{i\theta}$
$c+di = \cos\varphi + i\sin\varphi = e^{i\varphi}$
$e+fi = \cos(\theta+\varphi) + i\sin(\theta+\varphi) = e^{i(\theta+\varphi)}$

$e^{2\pi i} = 1$
$e^{\pi i} = -1$

Primitive n-th root of 1 $\omega=\omega_n$



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

Properties:
- $\omega^n = 1$.
- Any other **z** s.t. $z^n = 1$ has $z = \omega^k$ for some k<n.
- If n is even $\omega^2 = \omega_{n/2}$ is a primitive n/2-th root of 1.
- $\omega^{k+n/2} = -\omega^k$

$i^2 = -1$
$e^{2\pi i} = 1$

Multiplying Polynomials by Fast Fourier Transform

P: $a_0, a_1, \ldots, a_{m-1}$
Q: $b_0, b_1, \ldots, b_{m-1}$

→ ordinary polynomial multiplication $\Theta(n^2)$: $c_k \leftarrow \sum_{i+j=k} a_i b_j$ → R: $c_0, c_1, \ldots, c_{n-1}$

↓ evaluation at $1, \omega, \ldots, \omega^{n-1}$ O(n log n)

P(1),Q(1)
P(ω),Q(ω)
$P(\omega^2), Q(\omega^2)$
...
$P(\omega^{n-1}), Q(\omega^{n-1})$

→ point-wise multiplication of numbers O(n) →

R(1)=P(1)Q(1)
R(ω)=P(ω)Q(ω)
$R(\omega^2) = P(\omega^2)Q(\omega^2)$
...
$R(\omega^{n-1}) = P(\omega^{n-1})Q(\omega^{n-1})$

↑ interpolation from $1, \omega, \ldots, \omega^{n-1}$ O(n log n)

The key idea (since n is even)

- $P(\omega) = a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 + a_4\omega^4 + \ldots + a_{n-1}\omega^{n-1}$
  $= a_0 + a_2\omega^2 + a_4\omega^4 + \ldots + a_{n-2}\omega^{n-2}$
  $+ a_1\omega + a_3\omega^3 + a_5\omega^5 + \ldots + a_{n-1}\omega^{n-1}$
  $= P_{even}(\omega^2) + \omega\, P_{odd}(\omega^2)$
- $P(-\omega) = a_0 - a_1\omega + a_2\omega^2 - a_3\omega^3 + a_4\omega^4 - \ldots - a_{n-1}\omega^{n-1}$
  $= a_0 + a_2\omega^2 + a_4\omega^4 + \ldots + a_{n-2}\omega^{n-2}$
  $- (a_1\omega + a_3\omega^3 + a_5\omega^5 + \ldots + a_{n-1}\omega^{n-1})$
  $= P_{even}(\omega^2) - \omega\, P_{odd}(\omega^2)$

The recursive idea for n a power of 2

- Also
  - P_{even} and P_{odd} have degree n/2
  - $P(\omega^k) = P_{even}(\omega^{2k}) + \omega^k P_{odd}(\omega^{2k})$
  - $P(-\omega^k) = P_{even}(\omega^{2k}) - \omega^k P_{odd}(\omega^{2k})$
- Recursive Algorithm
  - Evaluate P_{even} at $1, \omega^2, \omega^4, \ldots, \omega^{n-2}$
  - Evaluate P_{odd} at $1, \omega^2, \omega^4, \ldots, \omega^{n-2}$
  - Combine to compute P at $1, \omega, \omega^2, \ldots, \omega^{n/2-1}$
  - Combine to compute P at $-1, -\omega, -\omega^2, \ldots, -\omega^{n/2-1}$ (i.e. at $\omega^{n/2}, \omega^{n/2+1}, \omega^{n/2+2}, \ldots, \omega^{n-1}$)

ω^2 is an n/2-th root of 1 so problem is of same type but smaller size

Analysis and more

- Run-time
  - $T(n) = 2T(n/2) + cn$ so $T(n) = O(n \log n)$
- So much for evaluation ... what about interpolation?
  - Given
    - $r_0 = R(1), r_1 = R(\omega), r_2 = R(\omega^2), \ldots, r_{n-1} = R(\omega^{n-1})$
  - Compute
    - $c_0, c_1, \ldots, c_{n-1}$ s.t. $R(x) = c_0 + c_1 x + \ldots c_{n-1} x^{n-1}$

Interpolation ≈ Evaluation: strange but true

- Weird fact:
  - If we define a new polynomial $T(x) = r_0 + r_1x + r_2x^2 + ... + r_{n-1}x^{n-1}$ where $r_0, r_1, \ldots, r_{n-1}$ are the evaluations of R at $1, \omega, \ldots, \omega^{n-1}$
  - Then $c_k = T(\omega^{-k})/n$ for $k=0,\ldots,n-1$
- So...
  - evaluate T at $1, \omega^{-1}, \omega^{-2}, \ldots, \omega^{-(n-1)}$ then divide each by n to get the $c_0, \ldots, c_{n-1}$
  - ω^{-1} behaves just like ω did so the same $O(n \log n)$ evaluation algorithm applies !

19

Why this is called the discrete Fourier transform

- Real Fourier series
  - Given a real valued function f defined on $[0,2\pi]$ the Fourier series for f is given by $f(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x) + ... + a_m \cos(mx) + ...$ where

$$a_m = \frac{1}{2\pi}\int_0^{2\pi} f(x)\cos(mx)\,dx$$

    is the component of f of frequency m
  - In signal processing and data compression one ignores all but the components with large a_m and there aren't many since $\sum_{m=0}^{\infty} a_m^2 = 1$

20

Why this is called the discrete Fourier transform

- Complex Fourier series
  - Given a function f defined on $[0,2\pi]$ the complex Fourier series for f is given by $f(z) = b_0 + b_1 e^{iz} + b_2 e^{2iz} + ... + b_m e^{miz} + ...$ where

$$b_m = \frac{1}{2\pi}\int_0^{2\pi} f(z)\,e^{-miz}\,dz$$

    is the component of f of frequency m
  - If we **discretize** this integral using values at n **$2\pi/n$ apart** equally spaced points between 0 and 2π we get

$$\bar{b}_m = \frac{1}{n}\sum_{k=0}^{n-1} f_k e^{-2km\pi i/n} = \frac{1}{n}\sum_{k=0}^{n-1} f_k\ \omega^{-km} \quad \text{where } f_k = f(2k\pi/n)$$

just like interpolation!

21