CS38
Introduction to Algorithms

Lecture 8

April 24, 2014

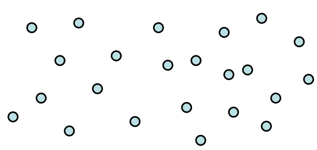
Outline

- Divide and Conquer design paradigm
  - closest pair (finishing up)
  - the DFT and the FFT
  - polynomial multiplication
  - polynomial division with remainder

  - integer multiplication
  - matrix multiplication

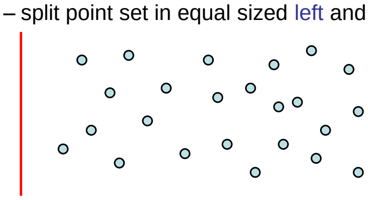
Closest pair in the plane

- Given n points in the plane, find the closest pair



Closest pair in the plane

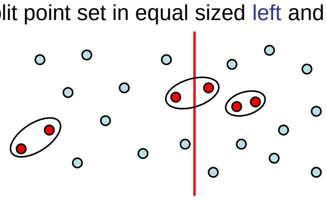
- Divide and conquer approach:
  - split point set in equal sized left and right sets



  - find closest pair in left, right, + across middle

Closest pair in the plane

- Divide and conquer approach:
  - split point set in equal sized left and right sets



  - find closest pair in left, right, + across middle

Closest pair in the plane

- Divide and conquer approach:
  - split point set in equal sized left and right sets
  - time to perform split?
  - sort by x coordinate: $O(n \log n)$
  - running time recurrence:

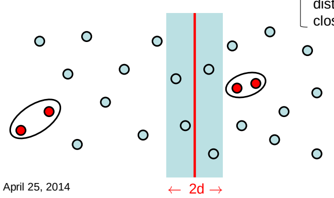
    $T(n) = 2T(n/2) + \text{time for middle} + O(n \log n)$

Is time for middle as bad as $O(n^2)$?

Closest pair in the plane

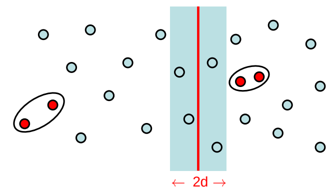
Claim: time for middle only $O(n \log n)$!!

- key: we know d = min of
  - distance between closest pair on left
  - distance between closest pair on right



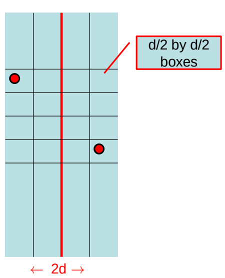
Observation: only need to consider points within distance d of the midline

Closest pair in the plane



- scan left to right to identify, then sort by y coord.
  - still $\Omega(n^2)$ comparisons?
  - Claim: only need do pairwise comparisons 15 ahead in this sorted list !

Closest pair in the plane



- no 2 points lie in same box (why?)
- if 2 points are within $\geq$ 16 positions of each other in list sorted by y coord…
- … then they must be separated by $\geq$ 3 rows
- implies dist. > (3/2)· d

Closest pair in the plane

Closest-Pair(P: set of n points in the plane)
1. sort by x coordinate and split equally into L and R subsets
2. (p,q) = **Closest-Pair(L)**
3. (r,s) = **Closest-Pair(R)**
4. d = min(distance(p,q), distance(r,s))
5. scan P by x coordinate to find M: points within d of midline
6. sort M by y coordinate
7. compute closest pair among all pairs within 15 of each other in M
8. return best among this pair, (p,q), (r,s)

- Running time:

  $T(2) = O(1)$; $T(n) = 2T(n/2) + O(n \log n)$

Closest pair in the plane

- Running time:

  $T(2) = a$; $T(n) = 2T(n/2) + bn \cdot \log n$

  set $c = \max(a/2, b)$

Claim: $T(n) \leq cn \cdot \log^2 n$

Proof: base case easy…

$T(n) \leq 2T(n/2) + bn \cdot \log n$

$\leq 2cn/2(\log n - 1)^2 + bn \cdot \log n$

$< cn(\log n)(\log n - 1) + bn \cdot \log n$

$\leq cn\log^2 n$

Closest pair in the plane

- we have proved:

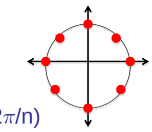
Theorem: There is an $O(n \log^2 n)$ time algorithm for finding the closest pair among n points in the plane.

- can be improved to $O(n \log n)$ by being more careful about maintaining sorted lists

The DFT, the FFT, and polynomial multiplication

Roots of unity

- An n-th root of unity is an element ω such that $\omega^n = 1$
  - primitive if $\omega^k \neq 1$ for $1 \leq k < n$



- examples:
  - in **C**: $e^{2\pi i/n} = \cos(2\pi/n) + i\cdot\sin(2\pi/n)$ is a primitive n-th root of unity
  - in integers mod 7: 2 is a primitive 3-th root of unity

Roots of unity

- An n-th root of unity is an element ω such that $\omega^n = 1$
  - primitive if $\omega^k \neq 1$ for $1 \leq k < n$
- key property:

$$\omega^{n-1} + \omega^{n-2} + \ldots + \omega^1 + \omega^0 = 0$$

why? $\omega \neq 1$ and

$$0 = \omega^n - 1 = (\omega - 1)(\omega^{n-1}+\omega^{n-2}+\ldots+\omega^1+\omega^0)$$

Discrete Fourier Transform (DFT)

- Given n-th root of unity ω, DFT_n is a linear map from C^n to C^n:

$$\begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ (\omega^2)^0 & (\omega^2)^1 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ \vdots & & & & \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix}$$

- (i,j) entry is ω^{ij}

Fast Fourier Transform (FFT)

- Given vector $x \in C^n$, how many operations to compute $DFT_n \cdot x$?

$$\begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ (\omega^2)^0 & (\omega^2)^1 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ \vdots & & & & \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

- standard matrix-vector multiplication: $O(n^2)$

Fast Fourier Transform (FFT)

- try Divide and Conquer:

$$\begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ (\omega^2)^0 & (\omega^2)^1 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ \vdots & & & & \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{N-1})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

- would lead to
  - $T(n) = 4T(n/2)$ + time to split/combine

  which implies $T(n) = \Omega(n^2)$

Fast Fourier Transform (FFT)

- DFT$_n$ has special structure (assume $n = 2^k$)
  - reorder columns: first even, then odd
  - consider exponents on ω along rows:

multiples of: | same multiples plus:

0 →	0	0	0	0	0	0	…	0	0	0	0	0	0	…	← 0
2 →	0	2	4	6	8	10	…	1	3	5	7	9	11	…	← 1
4 →	0	4	8	12	16	20	…	2	6	10	14	18	22	…	← 2
6 →	0	6	12	18	24	30	…	3	9	15	21	27	33	…	← 3
8 →	0	8	16	24	32	44	…	4	12	20	28	36	40	…	← 4

rows repeat twice since $\omega^n = 1$

Fast Fourier Transform (FFT)

- so we are actually computing:

ω^2 is (n/2)-th root of unity

$$\mathrm{DFT}_n \cdot \begin{pmatrix} x_{\text{even}} \\ x_{\text{odd}} \end{pmatrix} = \left(\begin{array}{c|c} \mathrm{DFT}_{n/2} & D \cdot \mathrm{DFT}_{n/2} \\ \mathrm{DFT}_{n/2} & \omega^{n/2} \cdot D \cdot \mathrm{DFT}_{n/2} \end{array}\right) \cdot \begin{pmatrix} x_{\text{even}} \\ x_{\text{odd}} \end{pmatrix}$$

- so to compute DFT$_n \cdot x$

D = diagonal matrix diag($\omega^0, \omega^1, \omega^2, \ldots, \omega^{n/2-1}$)

FFT(n:power of 2; x)
1. let ω be a n-th root of unity
2. compute a = FFT(n/2, x_{even})
3. compute b = FFT(n/2, x_{odd})
4. y_{even} = a + D·b and y_{odd} = a + $\omega^{n/2}$ ·D·b
5. return vector y

Fast Fourier Transform (FFT)

FFT(n:power of 2; x)
1. let ω be a n-th root of unity
2. compute a = FFT(n/2, x_{even})
3. compute b = FFT(n/2, x_{odd})
4. y_{even} = a + D·b and y_{odd} = a + $\omega^{n/2}$ ·D·b
5. return vector y

- Running time?
  - T(1) = 1
  - T(n) = 2T(n/2) + O(n)
  - solution: T(n) = O(n log n)

Discrete Fourier Transform (DFT)

- entry (i,j) of DFT$_n$ is ω^{ij} (n-th root of unity ω)
- claim: entry (i,j) of inverse-DFT$_n$ is ω^{-ij}/n

$$\begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & \cdots & (\omega^1)^{n-1} \\ \vdots & & & \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} (\omega^{-0})^0 & (\omega^{-0})^1 & \cdots & (\omega^{-0})^{n-1} \\ (\omega^{-1})^0 & (\omega^{-1})^1 & \cdots & (\omega^{-1})^{n-1} \\ \vdots & & & \\ (\omega^{-(n-1)})^0 & (\omega^{-(n-1)})^1 & \cdots & (\omega^{-(n-1)})^{n-1} \end{pmatrix}$$

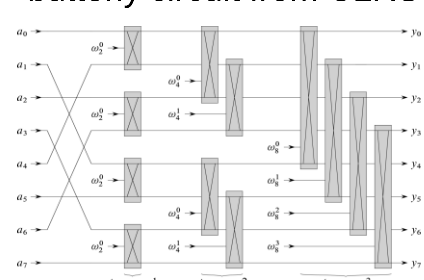
- entry (a,b) of this product is
  $\sum_k \omega^{ak}\omega^{-kb} = \sum_k \omega^{(a-b)k} = n$ if a=b; 0 otherwise

Discrete Fourier Transform (DFT)

<u>Theorem</u>: can compute DFT and inverse-DFT in O(n log n) operations

- extremely efficient in practice
  - parallel implementation via "butterfly circuit"

butterly circuit from CLRS



the DFT and polynomials

- given a polynomial
  $a(x) = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^{n-1}$
- observe that $DFT_n \cdot a$ gives evaluations of a at ω^i for $i=0,1,\ldots, n-1$

$$\begin{pmatrix} (\omega^0)^0 & (\omega^0)^1 & (\omega^0)^2 & \cdots & (\omega^0)^{n-1} \\ (\omega^1)^0 & (\omega^1)^1 & (\omega^1)^2 & \cdots & (\omega^1)^{n-1} \\ (\omega^2)^0 & (\omega^2)^1 & (\omega^2)^2 & \cdots & (\omega^2)^{n-1} \\ \vdots & & & & \\ (\omega^{n-1})^0 & (\omega^{n-1})^1 & (\omega^{n-1})^2 & \cdots & (\omega^{n-1})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix}$$

the DFT and polynomials

- since $DFT_n \cdot a$ gives evaluations of a at ω^i for $i=0,1,\ldots, n-1\ldots$
- inverse-$DFT_n \cdot$(vector of these evaluations) must give back a

$$\begin{pmatrix} (\omega^{-0})^0 & (\omega^{-0})^1 & \cdots & (\omega^{-0})^{n-1} \\ (\omega^{-1})^0 & (\omega^{-1})^1 & \cdots & (\omega^{-1})^{n-1} \\ \vdots & & & \\ (\omega^{-(n-1)})^0 & (\omega^{-(n-1)})^1 & \cdots & (\omega^{-(n-1)})^{n-1} \end{pmatrix} \cdot \begin{pmatrix} a(\omega^0) \\ a(\omega^1) \\ \vdots \\ a(\omega^{n-1}) \end{pmatrix}$$

Polynomial multiplication

- given two polynomials
  $a(x) = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^{n-1}$
  $b(x) = b_0x^0 + b_1x^1 + b_2x^2 + \ldots + b_{n-1}x^{n-1}$
- we want to compute the polynomial
  $a(x) \cdot b(x)$
  of degree at most 2n-2
- standard method takes $O(n^2)$ operations

Polynomial multiplication

- given two polynomials
  $a(x) = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^{n-1}$
  $b(x) = b_0x^0 + b_1x^1 + b_2x^2 + \ldots + b_{n-1}x^{n-1}$
  - $DFT_{2n} \cdot a$ and $DFT_{2n} \cdot b$ give evaluations of a, b at ω^i for $i = 0,1,\ldots, 2n-1$
  - can get evaluations of $a \cdot b$ at same points since $a(\omega^i)\cdot b(\omega^i) = (a\cdot b)(\omega^i)$
  - inverse-DFT_{2n} applied to (vector of these evaluations) gives back $a \cdot b$

Polynomial multiplication

polynomial-product(a, b: coeffs of degree n polynomials)
1. compute u = FFT(2n, a)
2. compute v = FFT(2n, b)
3. multiply vectors u,v pointwise to get vector w
4. return(inverse-FFT(2n,w))

- Running time?
  - O(n log n) for FFT and inverse-FFT
  - O(n) to multiply pointwise
- overall O(n log n)

Polynomial division

$$\begin{array}{r|l} & x^2 + 3x \qquad\qquad - 2 \\ \hline x^2 + 2 & x^4 + 3x^3 \qquad + 7x - 12 \\ & x^4 \qquad + 2x^2 \\ \hline & 3x^3 - 2x^2 + 7x - 12 \\ & 3x^3 \qquad + 6x \\ \hline & -2x^2 + x - 12 \\ & -2x^2 \qquad - 4 \\ \hline & x - 8 \end{array}$$

remainder : $x - 8$

check: $x^4 + 3x^3 + 7x – 12$ equals
$(x^2 + 2)(x^2 + 3x - 2) + (x – 8) = (x^4 + 3x^3 + 6x - 4) + (x – 8)$

Polynomial inversion

Theorem: given polynomial f with $f(0) = 1$, if

$g_0 = 1$, and

$g_{i+1} \equiv 2g_i - (f)(g_i)^2 \bmod x^{2^{i+1}}$

then $fg_i \equiv 1 \bmod x^{2^i}$ for all i.

Proof: induction on i

- base case: $fg_0 \equiv f(0)g_0 = 1 \cdot 1 \equiv 1 \pmod x$
- $1 - fg_{i+1} \equiv 1 - f(2g_i - f(g_i)^2) \equiv (1 - fg_i)^2 \equiv 0 \bmod x^{2^{i+1}}$

Polynomial division

polynomial-inversion (f: coeffs of deg. n poly; int. k)
output: polynomial g satisfying $fg = 1 \bmod x^k$
1. $g_0 = 1$; $r = \lceil \log k \rceil$
2. for i = 1 to r
3. $\quad g_i = 2g_i - (f)(g_{i-1})^2 \text{ rem } x^{2^i}$
4. return(g_r)

- Running time? (# operations)
  - $O(\log k) \cdot O(k \log k) = O(k \log^2 k)$
  - $O(k \log k)$ if careful about degrees in loop

Polynomial division

- (monic) polys a, b of deg. m, n ($m \le n$)
  we want polys q, r such that $a = qb + r$ and $\deg(r) < \deg(b)$
- key observation:
  $a(x) = a_0x^0 + a_1x^1 + a_2x^2 + \ldots + a_{n-1}x^n$
  $x^n\, a(1/x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x^0$
- denote by $\mathrm{rev}_n(a)$ this polynomial: $x^n a(1/x)$

Polynomial division

- (monic) polys a, b of deg. m, n ($m \le n$)
  we want polys q, r such that $a = qb + r$ and $\deg(r) < \deg(b)$

$\mathrm{rev}_{n-m}(b)$ is invertible mod x^{n-m+1} because constant coefficient is 1 (so $\mathrm{rev}_{n-m}(b)$ not divisible by x)

- algebra:
  $\mathrm{rev}_n(a) = \mathrm{rev}_{n-m}(q) \cdot \mathrm{rev}_m(b) + x^{n-m+1}\mathrm{rev}_{m-1}(r)$
  $\mathrm{rev}_n(a) \equiv \mathrm{rev}_{n-m}(q) \cdot \mathrm{rev}_m(b) \bmod x^{n-m+1}$
  $\mathrm{rev}_n(a) \cdot \mathrm{rev}_m(b)^{-1} \equiv \mathrm{rev}_{n-m}(q) \bmod x^{n-m+1}$

Polynomial division

poly-division-with-rem (a, b: coeffs of degr m, n polys)
output: polys q,r satisfying a = bq + r and deg(r) < deg(b)
1. r = deg(a) – deg(b)
2. compute inverse of $\mathrm{rev}_{\deg(b)}(b) \bmod x^{r+1}$
3. $q^* = (\mathrm{rev}_{\deg(a)}\, a) \cdot (\mathrm{rev}_{\deg(b)}\, b)^{-1} \text{ rem } x^{r+1}$
4. return($q = \mathrm{rev}_m(q^*)$ and $r = a - bq$)

- Running time? (# operations)
  - $O(n \log n)$

Polynomial multiplication and division

Theorem: can multiply and divide with remainder degree n polynomials in $O(n \log n)$ time

integer multiplication

- given 2 n-bit integers x, y
- compute their product xy
- standard multiplication $O(n^2)$
- simple divide and conquer improves to $O(n^{\log_2 3}) = O(n^{1.59})$

integer multiplication

- given 2 n-bit integers x, y
- write:
  - $x = x_1 \cdot 2^{n/2} + x_0$
  - $y = y_1 \cdot 2^{n/2} + y_0$
- note: $xy = x_1y_1 \cdot 2^n + (x_1y_0 + x_0y_1) \cdot 2^{n/2} + x_0y_0$
- clever idea:

  $(x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0$

integer multiplication

integer-mult(x, y: n-bit integers)
1. write $x = x_1 \cdot 2^{n/2} + x_0$ and $y = y_1 \cdot 2^{n/2} + y_0$
2. a = integer-mult(x_1, y_1)
3. b = integer-mult(x_0, y_0)
4. c = integer-mult($x_0 + x_1$, $y_0 + y_1$)
5. return($a \cdot 2^n + (c - a - b) \cdot 2^{n/2} + b$)

- Running time recurrence? (# operations)
  - $T(n) = 3T(n/2) + O(n)$
  - $T(n) = O(n^{\log_2 3}) = O(n^{1.59})$