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

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 and distance between closest pair on right.

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

- 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!

**Running time**:

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

---

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

- we have proved:
  - can be improved to $O(n \log n)$ by being more careful about maintaining sorted lists.
The DFT, the FFT, and polynomial multiplication

April 25, 2014

CS38 Lecture 8

13

Roots of unity

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

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

April 25, 2014

CS38 Lecture 8

14

Roots of unity

• An n-th root of unity is an element $\omega$ 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

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

April 25, 2014

CS38 Lecture 8

15

Discrete Fourier Transform (DFT)

• Given n-th root of unity $\omega$, $\text{DFT}_n$ is a linear map from $\mathbb{C}^n$ to $\mathbb{C}^n$:

$$\begin{pmatrix}
\omega^0 & \omega^1 & \omega^2 & \ldots & \omega^{n-1}
\end{pmatrix}$$

April 25, 2014

CS38 Lecture 8

16

Fast Fourier Transform (FFT)

• Given vector $x \in \mathbb{C}^n$, how many operations to compute $\text{DFT}_n x$?

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

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

April 25, 2014

CS38 Lecture 8

17

Fast Fourier Transform (FFT)

• try Divide and Conquer:

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

would lead to

– $T(n) = 4T(n/2) + \text{time to split/combine}$
which implies $T(n) = \Omega(n^2)$

April 25, 2014

CS38 Lecture 8

18
Fast Fourier Transform (FFT)

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

<table>
<thead>
<tr>
<th>multiples of:</th>
<th>same multiples plus:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2 (\omega) (\omega^2) (\omega^4) (\omega^8) (\omega^{16}) (\omega^{32}) (\omega^{64}) (\omega^{128})</td>
</tr>
<tr>
<td>4</td>
<td>4 (\omega) (\omega^2) (\omega^4) (\omega^8) (\omega^{16}) (\omega^{32}) (\omega^{64}) (\omega^{128})</td>
</tr>
<tr>
<td>6</td>
<td>6 (\omega^3) (\omega^6) (\omega^{12}) (\omega^{24}) (\omega^{48}) (\omega^{96}) (\omega^{192})</td>
</tr>
<tr>
<td>8</td>
<td>8 (\omega^4) (\omega^8) (\omega^{16}) (\omega^{32}) (\omega^{64}) (\omega^{128}) (\omega^{256})</td>
</tr>
<tr>
<td>rows repeat twice since (\omega^2 = 1)</td>
<td></td>
</tr>
</tbody>
</table>

- so we are actually computing:

\[
\text{DFT}_n(\text{even}) = \begin{pmatrix} D & \text{DFT}_n(\text{odd}) \end{pmatrix} (\omega^{n/2}\cdot \text{DFT}_n(\text{odd}))
\]

- so to compute DFT\(_n\) \(x\)

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

Discrete Fourier Transform (DFT)

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

\[
\begin{pmatrix} \omega^{00} & \omega^{01} & \cdots & \omega^{0n-1} \\ \omega^{10} & \omega^{11} & \cdots & \omega^{1n-1} \\ \cdots & \cdots & \cdots & \cdots \\ \omega^{n-10} & \omega^{n-11} & \cdots & \omega^{n-1n-1} \end{pmatrix} \begin{pmatrix} \omega^{-00} & \omega^{-01} & \cdots & \omega^{-0n-1} \\ \omega^{-10} & \omega^{-11} & \cdots & \omega^{-1n-1} \\ \cdots & \cdots & \cdots & \cdots \\ \omega^{-(n-1)0} & \omega^{-(n-1)1} & \cdots & \omega^{-(n-1)n-1} \end{pmatrix}
\]

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

Theorem: can compute DFT and inverse-DFT in \(O(n \log n)\) operations

- extremely efficient in practice
  - parallel implementation via “butterfly circuit”
the DFT and polynomials

- given a polynomial
  \[ a(x) = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \]
- observe that \( \text{DFT}_n a \) gives evaluations of \( a \) at \( \omega^i \) for \( i = 0, 1, \ldots, n-1 \)

- inverse-\( \text{DFT}_n \) (vector of these evaluations) must give back \( a \)

Polynomial multiplication

- given two polynomials
  \[ a(x) = a_0 x^0 + a_1 x^1 + a_2 x^2 + \ldots + a_{n-1} x^{n-1} \]
  \[ b(x) = b_0 x^0 + b_1 x^1 + b_2 x^2 + \ldots + b_{n-1} x^{n-1} \]
- 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 division

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
\[
g_0 = 1, \quad \text{and} \quad g_{i+1} \equiv 2g_i - (f)(g_i^2) \mod x^{2i+1}
\]
then \( fg_i \equiv 1 \mod x^{2i} \) for all \( i \).

Proof: induction on \( i \)
- **base case**: \( fg_0 \equiv f(0)g_0 = 1 \equiv 1 \mod x \)
- \( 1 - fg_{i+1} \equiv 1 - f(2g_i - (f)(g_i^2)) \equiv (1 - fg_i^2) \equiv 0 \mod x^{2i+1} \)

Polynomial division

- **Running time? (# operations)**
  - \( O(n \log n) \)
- **poly-division-with-rem**
  - **Output**: polynomials \( q, r \) satisfying \( a = bq + r \) and \( \deg(r) < \deg(b) \)
  1. \( r = \deg(a) - \deg(b) \)
  2. Compute inverse of \( \text{rev}_{\deg(b)}(b) \mod x^{r+1} \)
  3. \( q = (\text{rev}_{\deg(a)}(a))^{-1} \mod x^{r+1} \)
  4. \( \text{return}(q = \text{rev}_{\deg}(q)) \) and \( r = a - bq \)

**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 $O(n^{\log_2 3}) = O(n^{1.59})$

April 25, 2014
CS38 Lecture 8
37

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

April 25, 2014
CS38 Lecture 8
38

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. $a = \text{integer-mult}(x_1, y_1)$
3. $b = \text{integer-mult}(x_0, y_0)$
4. $c = \text{integer-mult}(x_0 + x_1, y_0 + y_1)$
5. return $(a \cdot 2^n + (c - a - b) \cdot 2^{n/2} + b)$

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

April 25, 2014
CS38 Lecture 8
39