CSE 421: Introduction to Algorithms

Divide and Conquer

Winter 2003
Paul Beame

Algorithm Design Techniques

- Divide & Conquer
  - Reduce problem to one or more sub-problems of the same type
  - Typically, each sub-problem is at most a constant fraction of the size of the original problem
    - e.g. Mergesort, Binary Search, Strassen’s Algorithm, Quicksort (kind of)

Fast exponentiation

- Power(a,n)
  - Input: integer n and number a
  - Output: a^n
  - Obvious algorithm
    - n-1 multiplications
  - Observation:
    - if n is even, n=2m, then a^n=a^m*a^m

Divide & Conquer Algorithm

- Power(a,n)
  - if n=0 then return(1)
  - else if n=1 then return(a)
  - else
    - x ← Power(a,\lfloor n/2 \rfloor)
    - if n is even then
      - return(x*x)
    - else
      - return(a*x*x)

Analysis

- Worst-case recurrence
  - T(n)=T(\lfloor n/2 \rfloor)+2 for n=1
  - T(1)=0
  - Time
    - T(n)=T(\lfloor n/2 \rfloor)+2 ≤ T(\lfloor n/4 \rfloor)+2+2 ≤ \ldots ≤ T(1)+2^k+\ldots+2 = 2 \log_2 n \text{ copies}
  - More precise analysis:
    - T(n) = \lceil \log_2 n \rceil + \# of 1's in n’s binary representation

A Practical Application- RSA

- Instead of a^n want a^n mod N
  - a^n mod N = ((a^i mod N)+(a^j mod N)) mod N
  - same algorithm applies with each x*y replaced by
    - ((x mod N)+(y mod N)) mod N
  - In RSA cryptosystem (widely used for security)
    - need a^n mod N where a, n, N each typically have 1024 bits
    - Power: at most 2048 multiplies of 1024 bit numbers
      - relatively easy for modern machines
    - Naive algorithm: 2^{1024} multiplies
Binary search for roots (bisection method)

Given:
- continuous function \( f \) and two points \( a-b \) with \( f(a) \leq 0 \) and \( f(b) > 0 \)

Find:
- approximation to \( c \) s.t. \( f(c)=0 \) and \( a-c\leq b \)

Bisection method

\[
\text{Bisection}(a, b, \varepsilon) \quad \begin{cases} 
\text{if } (a-b) < \varepsilon & \text{then return}(a) \\
\text{else } c \leftarrow (a+b)/2 \\
\text{if } f(c) \leq 0 & \text{then return}(\text{Bisection}(c, b, \varepsilon)) \\
\text{else } \text{return}(\text{Bisection}(a, c, \varepsilon))
\end{cases}
\]

Time Analysis

- At each step we halved the size of the interval
- It started at size \( b-a \)
- It ended at size \( \varepsilon \)
- # of calls to \( f \) is \( \log_2(\frac{b-a}{\varepsilon}) \)

Euclidean Closest Pair

- Given a set \( P \) of \( n \) points \( p_1, \ldots, p_n \) with real-valued coordinates
- Find the pair of points \( p_i, p_j \in P \) such that the Euclidean distance \( d(p_i, p_j) \) is minimized
- \( \Theta(n^2) \) possible pairs
- In one dimension there is an easy \( O(n \log n) \) algorithm
  - Sort the points
  - Compare consecutive elements in the sorted list
- What about points in the plane?

Closest Pair in the Plane

- Sort the points by their \( x \) coordinates
- Split the points into two sets of \( n/2 \) points \( L \) and \( R \) by \( x \) coordinate
- Recursively compute
  - closest pair of points in \( L, (p_L, q_L) \)
  - closest pair of points in \( R, (p_R, q_R) \)
- Let \( \delta = \min(d(p_L, q_L), d(p_R, q_R)) \) and let \( (p, q) \) be the pair of points that has distance \( \delta \)
- This may not be enough!
  - Closest pair of points may involve one point from \( L \) and the other from \( R \)!
A clever geometric idea

Any pair of points $p \in L$ and $q \in R$ with $d(p,q) < \hat{d}$ must lie in band.

No two points can be in the same green box.

Only need to check pairs of points up to 2 rows above and below.

At most 15 other points!

Closest Pair Recombining

- Sort points by $y$ coordinate ahead of time.
- On recombination only compare each point in $L \cup R$ to the 12 points above it in the $y$ sorted order.
- If any of those distances is better than $\hat{d}$ replace $(p,q)$ by the best of those pairs.
- $O(n \log n)$ for $x$ and $y$ sorting at start.
- Two recursive calls on problems on half size.
- $O(n)$ recombination.
- Total $O(n \log n)$.

Sometimes two sub-problems aren’t enough

- More general divide and conquer.
  - You’ve broken the problem into a different sub-problems.
  - Each has size at most $n/b$.
  - The cost of the break-up and recombining the sub-problem solutions is $O(n^k)$.

  - Recurrence
    - $T(n) \leq a \cdot T(n/b) + c \cdot n^k$

Master Divide and Conquer Recurrence

- If $T(n) \leq a \cdot T(n/b) + c \cdot n^k$ for $n > b$ then
  - if $a > b^k$ then $T(n)$ is $O(n^{\log_b a})$.
  - if $a = b^k$ then $T(n)$ is $\Theta(n^k)$.
  - if $a < b^k$ then $T(n)$ is $O(n^k \log n)$.

- Works even if it is $\lceil n/b \rceil$ instead of $n/b$.

Proving Master Recurrence

Problem size

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n/b$</th>
<th>$n/b^2$</th>
<th>$b$</th>
<th>$1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(1) = c$</td>
<td>$a$</td>
<td>$a^2$</td>
<td>$a^d$</td>
<td>$a^d$</td>
</tr>
</tbody>
</table>

Problem size

<table>
<thead>
<tr>
<th>$n$</th>
<th>$n/b$</th>
<th>$n/b^2$</th>
<th>$b$</th>
<th>$1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d = \log n$</td>
<td>$a$</td>
<td>$a^2$</td>
<td>$a^d$</td>
<td>$a^d$</td>
</tr>
</tbody>
</table>

Proving Master Recurrence
### Geometric Series
- \( S = t + tr + tr^2 + \ldots + tr^{n-1} \)
- \( rS = tr + tr^2 + \ldots + tr^{n-1} + tr^n \)
- \((r-1)S = tr^n - t\)
- so \( S = t \frac{r^n - 1}{r-1} \) if \( r \neq 1 \).

- Simple rule
  - If \( r = 1 \) then \( S \) is a constant times largest term in series

### Multiplying Matrices
- \[ \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{bmatrix} \]
- \[ \begin{bmatrix} e_0 & e_1 & e_2 & e_3 \\ f_0 & f_1 & f_2 & f_3 \\ g_0 & g_1 & g_2 & g_3 \\ h_0 & h_1 & h_2 & h_3 \end{bmatrix} \]
- \[ \begin{bmatrix} a_0 e_0 + a_1 e_1 + a_2 e_2 + a_3 e_3 \\ b_0 f_0 + b_1 f_1 + b_2 f_2 + b_3 f_3 \\ c_0 g_0 + c_1 g_1 + c_2 g_2 + c_3 g_3 \\ d_0 h_0 + d_1 h_1 + d_2 h_2 + d_3 h_3 \end{bmatrix} \]
- \[ a_0 f_0 + a_1 f_1 + a_2 f_2 + a_3 f_3 \]
- \[ b_0 g_0 + b_1 g_1 + b_2 g_2 + b_3 g_3 \]
- \[ c_0 h_0 + c_1 h_1 + c_2 h_2 + c_3 h_3 \]
- \[ d_0 e_0 + d_1 e_1 + d_2 e_2 + d_3 e_3 \]

- \( n^3 \) multiplications, \( n^3 - n^2 \) additions

### Proving Master Recurrence
- Problem size
  - \( T(n) = aT(n/b) + cn^d \)
  - \( a \) number of problems
  - \( c \) cost per problem
  - \( n^d \) work per problem

- \( \log_b n \) iterations

### Total Cost
- Geometric series
  - ratio \( a/b^k \)
  - \( d+1 = \log_b n + 1 \) terms
  - first term \( cn^d \), last term \( ca^d \)
  - If \( a/b^k = 1 \)
    - all terms are equal \( T(n) = \Omega(n \log n) \)
  - If \( a/b^k < 1 \)
    - first term is largest \( T(n) = \Theta(n^d) \)
  - If \( a/b^k > 1 \)
    - last term is largest \( T(n) = \Theta(a^d) = \Theta((a \log_b n) = \Theta(n \log_b a) \)
      (To see this take \( \log_b \) of both sides)
The algorithm

\[ \begin{align*}
P_1 &= A_{12}(B_{11} + B_{22}) ; \\
P_2 &= A_{21}(B_{12} + B_{22}) \\
P_3 &= (A_{11} - A_{12})B_{11} ; \\
P_4 &= (A_{21} - A_{22})(B_{12} - B_{22}) \\
P_5 &= (A_{22} - A_{12})(B_{21} - B_{22}) \\
P_6 &= (A_{11} - A_{21})(B_{12} - B_{11}) \\
P_7 &= (A_{21} - A_{12})(B_{11} + B_{22}) \\
\end{align*} \]

\[ \begin{align*}
C_{11} &= P_1 + P_2 ; \\
C_{12} &= P_2 + P_3 + P_6 + P_7 \\
C_{21} &= P_1 + P_4 + P_5 + P_7 ; \\
C_{22} &= P_2 + P_4 \\
\end{align*} \]

**Strassen's Divide and Conquer Algorithm**

- Strassen's algorithm
  - Multiply 2x2 matrices using 7 instead of 8 multiplications (and lots more than 4 additions)
  - \( T(n) \leq 7T(n/2) + cn^2 \)
  - \( 7 > 2^2 \) so \( T(n) = \Theta(n\log_2 7) \) which is \( \Theta(n^2\log n) \)
- Fastest algorithms theoretically use \( \Theta(n^{2.376}) \) time
  - Not practical but Strassen's is practical provided calculations are exact and we stop recursion when matrix has size about 100 (maybe 10)

**Another Divide & Conquer Example: Multiplying Faster**

- If you analyze our usual grade school algorithm for multiplying numbers
  - \( \Theta(n^2) \) time
- On real machines each "digit" is, e.g., 32 bits long but still get \( \Theta(n^2) \) running time with this algorithm when run on \( n \)-bit multiplication
- 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
Notes on Polynomials

- These are just formal sequences of coefficients
  - when we show something multiplied by $x^k$ it just means shifted $k$ places to the left – basically no work

Usual polynomial multiplication

\[
\begin{array}{c}
4x^2 + 2x + 2 \\
\hline
x^2 - 3x + 1 \\
4x^2 + 2x + 2 \\
\end{array}
\]

- $-12x^2 + 6x - 6x$ 
- $4x^4 + 2x^3 + 2x^2$ 
- $4x^4 - 10x^3 + 0x^2 - 4x + 2$

Polynomial Multiplication

- Given:
  - Degree $n-1$ polynomials $P$ and $Q$
    \[
P = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 \quad \text{and} \quad Q = b_m x^m + b_{m-1} x^{m-1} + \ldots + b_0
    \]
  - $4n^2 - 3n + 1$ arithmetic operations
- Obvious Algorithm:
  - Compute all $a_i b_j$ and collect terms
  - $\Theta(n^2)$ time

Naive Divide and Conquer

- Assume $n=2k$ 
  - $P = (a_{2k} + a_{2k-1} x + \ldots + a_0) x^k + (a_{2k-1} + a_{2k-2} x + \ldots + a_0) x^k$
  - $Q = (b_{2k} + b_{2k-1} x + \ldots + b_0) x^k$
  - $PQ = (a_{2k} + a_{2k-1} x + \ldots + a_0) (b_{2k} + b_{2k-1} x + \ldots + b_0) x^k$
  - $4$ sub-problems of size $k=n/2$ plus linear combining
  - $T(n)=4T(n/2)+cn$ Solution $T(n) = \Theta(n^2)$

Karatsuba’s Algorithm

- A better way to compute the terms
  - Compute
    \[
    A = P \times Q \\
    B = P \times \bar{Q} \\
    C = (P + \bar{P}) \times (Q + \bar{Q})
    \]
  - Then
    \[
    PQ = A - C + B
    \]
  - $3$ sub-problems of size $n/2$ plus $O(n)$ work
  - $T(n) = 3T(n/2) + O(n)$ work
  - $T(n) = \Theta(n \log n)$ when $n = 2^k$

Karatsuba: Details

PolyMul($P, Q$):

- $P, Q$ are length $n=2k$ vectors, with $P[i], Q[i]$ being the coefficient of $x^i$ in polynomials $P, Q$ respectively.
- Let Pzero be elements 0..k-1 of P; Pone be elements k..n-1
- Let Qzero, Qone: similar
  - If $n=1$ then Return(P[0]Q[0])
  - Else
    - $A = \text{PolyMul}(Pzero, Qzero)$; // result is a $(2k-1)$-vector
    - $B = \text{PolyMul}(Pone, Qone)$; // ditto
    - $C = A + B$; // add corresponding elements
    - $Psum = \text{PolyMul}(Pzero, Qzero)$; // another $(2k-1)$-vector
    - $\text{Mid} = C - A - B$; // subtract correspond elements
    - $R = A + \text{Shift}(\text{Mid}, n/2) + \text{Shift}(B, n)$ // a $(2n-1)$-vector
    - Return( $R$ );

Multiplication

- Polynomials
  - Naïve: $\Theta(n^2)$
  - Karatsuba: $\Theta(n^{1.58})$
  - Best known: $\Theta(n \log n)$
  - “Fast Fourier Transform”
  - FFT widely used for signal processing
- Integers
  - Similar, but some ugly details re: carries, etc. gives $\Theta(n \log n \log \log n)$
  - Mostly unused in practice except for symbolic manipulation systems like Maple
Hints towards FFT: Interpolation

Given set of values at 5 points

Interpolation

- Given values of degree n-1 polynomial R at n distinct points y_1, ..., y_n
- Compute coefficients c_0, ..., c_{n-1} such that
  \[ R(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \]
- System of linear equations in c_0, ..., c_{n-1}
  \[ \begin{align*}
  c_0 + c_1 y_1 + c_2 y_1^2 + \cdots + c_{n-1} y_1^{n-1} &= R(y_1) \\
  c_0 + c_1 y_2 + c_2 y_2^2 + \cdots + c_{n-1} y_2^{n-1} &= R(y_2) \\
  &\vdots \\
  c_0 + c_1 y_n + c_2 y_n^2 + \cdots + c_{n-1} y_n^{n-1} &= R(y_n)
  \end{align*} \]

Matrix form of the linear system
\[ \begin{pmatrix}
  1 & y_1^2 & y_1^{n-1} \\
  1 & y_2^2 & y_2^{n-1} \\
  \vdots & \vdots & \vdots \\
  1 & y_n^2 & y_n^{n-1}
\end{pmatrix} \begin{pmatrix}
  c_0 \\
  c_1 \\
  c_2 \\
  \vdots \\
  c_{n-1}
\end{pmatrix} = \begin{pmatrix}
  R(y_1) \\
  R(y_2) \\
  \vdots \\
  R(y_n)
\end{pmatrix} \]

Fact: Determinant of the matrix is \( \prod_{i<j} (y_i - y_j) \)
which is not 0 since points are distinct
- System has a unique solution \( c_0, ..., c_{n-1} \)

Hints towards FFT: Evaluation & Interpolation

Karatsuba’s algorithm and evaluation and interpolation

- Strassen gave a way of doing 2x2 matrix multiplies with fewer multiplications
- Karatsuba’s algorithm can be thought of as a way of multiplying degree 1 polynomials (which have 2 coefficients) using fewer multiplications
  \[ P_0 + P_1 x + P_2 x^2 = (P_0 + P_2 x)(Q_0 + Q_2 x) + P_1 Q_1 x \]
- Evaluate at 0, 1, -1 (Could also use other points)
  \[ A = P_0 \cdot Q_0 = P_0 Q_0 + P_0 Q_2 x + P_2 Q_0 x^2 \]
- Interpolating, Karatsuba’s Mid=(C-D)/2 and \( B=(C+D)/2-A \)
Hints towards FFT: Evaluation at Special Points
- Evaluation of polynomial at 1 point takes $O(n)$
  - So $2n$ points (naively) takes $O(n^2)$—no savings
- Key trick:
  - use carefully chosen points where there's some sharing of work for several points, namely various powers of $\omega = e^{2\pi i/n}$, $i = \sqrt{-1}$
- Plus more Divide & Conquer.

Result:
- both evaluation and interpolation in $O(n \log n)$ time

The key idea for $n$ even
- $P(\omega) = a_0 + a_1\omega + a_2\omega^2 + a_3\omega^3 + \ldots + a_n\omega^n$
  - $P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2)$
  - $P(\omega^2) = a_0 + a_2\omega^2 + a_4\omega^4 + \ldots + a_n\omega^{2n-1}$
    - $P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2)$

where $P_{\text{even}}(x) = a_0 + a_2x + a_4 x^2 + \ldots + a_n x^{n/2-1}$ and

The recursive idea for $n$ a power of 2
- Also $P_{\text{even}}$ and $P_{\text{odd}}$ have degree $n/2$ where
  - $P(\omega^2) = P_{\text{even}}(\omega^2) + \omega P_{\text{odd}}(\omega^2)$
  - $P(\omega) = P_{\text{even}}(\omega n/2) + \omega P_{\text{odd}}(\omega n/2)$

Recursive Algorithm
- Evaluate $P_{\text{even}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
- Evaluate $P_{\text{odd}}$ at $1, \omega^2, \omega^4, \ldots, \omega^{n/2-1}$
- Combine to compute $P$ at $1, \omega, \omega^2, \ldots, \omega^{n/2-1}$
  - i.e. at $\omega^0, \omega^1, \omega^2, \ldots, \omega^{n/2-1}$

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 $S(x) = r_0 + r_1 x + r_2 x^2 + \ldots + r_{n-1} x^{n-1}$ where $r_0, r_1, \ldots, r_{n-1}$ are the evaluations of $R$ at $1, \omega, \omega^2, \ldots, \omega^{n-1}$
  - Then $c_k = S(\omega^k)/n$ for $k = 0, \ldots, n-1$

  So...
  - evaluate $S$ at $1, \omega^2, \omega^4, \ldots, \omega^{2(n-1)}$ then divide each answer by $n$ to get the $c_0, c_1, \ldots, c_{n-1}$
  - $\omega^k$ behaves just like $\omega$ did so the same $O(n \log n)$ evaluation algorithm applies!
Divide and Conquer Summary

- Powerful technique, when applicable
- Divide large problem into a few smaller problems of the same type
- Choosing sub-problems of roughly equal size is usually critical
- Examples:
  - Merge sort, quicksort (sort of), polynomial multiplication, FFT, Strassen's matrix multiplication algorithm, powering, binary search, root finding by bisection, …

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
    \[
    f(x) = a_0 + a_1 \cos(x) + a_2 \cos(2x) + \ldots + a_m \cos(mx) + \ldots
    \]
    where
    
    \[
    a_n = \frac{1}{2\pi} \int_{0}^{2\pi} f(x) \cos(nx) \, dx
    \]
  - is the component of \( f \) of frequency \( m \)
  - In signal processing and data compression one ignores all but the components with large \( a_n \) and there aren't many since

- Complex Fourier series
  - Given a function \( f \) defined on \([0,2\pi] \)
  - the complex Fourier series for \( f \) is given by
    \[
    f(z) = b_0 + b_1 e^{iz} + b_2 e^{2iz} + \ldots + b_m e^{miz} + \ldots
    \]
  - where
    
    \[
    b_n = \frac{1}{2\pi} \int_{0}^{2\pi} f(z) e^{-inz} \, dz
    \]
  - is the component of \( f \) of frequency \( m \)
  - If we discretize this integral using values at \( \frac{2\pi}{n} \) equally spaced points between 0 and \( 2\pi \) we get
    \[
    b_n = \frac{1}{n} \sum_{k=0}^{n-1} f(\frac{2\pi k}{n}) e^{-i2\pi km/n}
    \]
    where \( f(\frac{2\pi k}{n}) \) just like interpolation!