Algorithms are not only for Sorting and Graphs!
Evaluating Polynomials

Given a polynomial
\[ p(x) = 5 + 2x + 8x^2 + 3x^3 + 4x^4 \]
and a value of \( x = 3.14159 \), how do we compute the value of \( p(x) \)?

In general, \( p(x) = \sum_{i=0}^{n} a_i x^i \)

Brute force computation: \( 2n-1 \) multiplications and \( n \) additions
Horner’s Rule

Represent the polynomial

\[ p(x) = 5 + 2x + 8x^2 + 3x^3 + 4x^4 \]

in the following form

\[ p(x) = 5 + x(2 + x(8 + x(3 + 4x))) \]

In general,

\[ p(x) = a_0 + x(a_1 + x(a_2 + x(... x(a_{n-1} + x a_n x)...) \]

Computing inside-out requires \( n \) multiplications and \( n \) additions
Simple Exponentiation

Given a value of $x$ and an exponent $n$, how do we compute $x^n$?

Brute force: $n-1$ multiplications

What if $n=2^k$?

How can we use the binary representation of $n$ to reduce the number of multiplications?
Graphs and Matrix Multiplication

If matrix $A$ represents a graph $G$, and $A \times A = C$, then what is the meaning of $C_{ij} = 0$? And of $C_{ij} \neq 0$?

Recall that $C_{ij} = \sum_{k=1}^{n} a_{ik}a_{kj}$

Let $A$ represent a graph $G$, $B = A + I$, and $C = B^m$. When is $C_{ij} = 0$?

Let $A$ represent $G$. How can we use matrix multiplication to decide if there is a path from $s$ to $t$ in $G$?
Basic Matrix Multiplication

Suppose that we want to multiply two matrices of size $2 \times 2$:

\[
\begin{pmatrix}
a_{11} & a_{12} \\
a_{21} & a_{22}
\end{pmatrix}
\times
\begin{pmatrix}
b_{11} & b_{12} \\
b_{21} & b_{22}
\end{pmatrix}
= 
\begin{pmatrix}
c_{11} & c_{12} \\
c_{21} & c_{22}
\end{pmatrix}
\]

\[
c_{11} = a_{11}b_{11} + a_{12}b_{21}
\]
\[
c_{12} = a_{11}b_{12} + a_{12}b_{22}
\]
\[
c_{21} = a_{21}b_{11} + a_{22}b_{21}
\]
\[
c_{22} = a_{21}b_{12} + a_{22}b_{22}
\]

$2 \times 2$ matrix multiplication can be computed using $8 (=2^3)$ multiplications.
Basic Matrix Multiplication

Matrix Multiplication algorithm

```plaintext
matrix_mult
for i ← 1 to N do
  for j ← 1 to N do
    compute C_{ij};
```

Time analysis

\[
C_{i,j} = \sum_{k=1}^{N} a_{i,k} b_{k,j}
\]

Thus \( T(N) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{k=1}^{N} c = cN^3 = O(N^3) \)
Divide and Conquer Matrix Multiply

\[ \begin{array}{cc}
A_0 & A_1 \\
A_2 & A_3 \\
\end{array} \times \begin{array}{cc}
B_0 & B_1 \\
B_2 & B_3 \\
\end{array} = \begin{array}{cc}
A_0 \times B_0 + A_1 \times B_2 & A_0 \times B_1 + A_1 \times B_3 \\
A_2 \times B_0 + A_3 \times B_2 & A_2 \times B_1 + A_3 \times B_3 \\
\end{array} \]

- Divide matrices into sub-matrices: \( A_0, A_1, A_2 \) etc.
- Use matrix multiply equations
- Recursively multiply sub-matrices
Divide and Conquer Matrix Multiply

\[
A \times B = C
\]

\[
\begin{bmatrix} a_0 \end{bmatrix} \times \begin{bmatrix} b_0 \end{bmatrix} = \begin{bmatrix} a_0b_0 \end{bmatrix}
\]

Terminate recursion with 1x1 matrices
( = scalar multiplication)
Time Analysis

\[ T(1) = 1 \]
\[ T(N) = 8 \, T(N/2) + cN^2 \]
\[
\vdots
\]
\[ T(N) = d8^{\log N} = dN^{\log 8} = \Theta(N^3) \]
Strassen’s Matrix Multiplication

- Strassen showed that $2 \times 2$ matrix multiplication can be accomplished using 7 multiplications and 18 additions or subtractions.

- By Divide and Conquer, this can be used to give an $\Theta(n^{2.81})$ matrix multiplication algorithm.
  
  (The exponent comes from $\log_2 7 = 2.81$)
Strassen’s Matrix Multiplication

\[
\begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\times
\begin{pmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{pmatrix}
=
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix}
\]

P_1 = (A_{11} + A_{22}) \times (B_{11} + B_{22})
P_2 = (A_{21} + A_{22}) \times B_{11}
P_3 = A_{11} \times (B_{12} - B_{22})
P_4 = A_{22} \times (B_{21} - B_{11})
P_5 = (A_{11} + A_{12}) \times B_{22}
P_6 = (A_{21} - A_{11}) \times (B_{11} + B_{12})
P_7 = (A_{12} - A_{22}) \times (B_{21} + B_{22})

C_{11} = P_1 + P_4 - P_5 + P_7
C_{12} = P_3 + P_5
C_{21} = P_2 + P_4
C_{22} = P_1 + P_3 - P_2 + P_6

Slides thanks to Sibel KIRMIZIGÜL
Example

\[ C_{11} = P_1 + P_4 - P_5 + P_7 \]

\[ = (A_{11} + A_{22}) \times (B_{11} + B_{22}) + A_{22} \times (B_{21} - B_{11}) - \]

\[ (A_{11} + A_{12}) \times B_{22} + (A_{12} - A_{22}) \times (B_{21} + B_{22}) \]

\[ = A_{11} B_{11} + A_{11} B_{22} + A_{22} B_{11} + A_{22} B_{22} + A_{22} B_{21} - A_{22} B_{11} - \]

\[ A_{11} B_{22} - A_{12} B_{22} + A_{12} B_{21} + A_{12} B_{22} - A_{22} B_{21} - A_{22} B_{22} \]

\[ = A_{11} B_{11} + A_{12} B_{21} \]
Strassen's Algorithm

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

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

**Strassen\_mult(A, B, N)**  
% A, B are N\times N matrices and N is a power of 2  
If N=1 then return \(a_{11} \times b_{11}\)  
else  
Compute \(P_1, ..., P_7\) by recursively calling **Strassen\_mult**  
Use above decomposition to compute \(C_{11}, ..., C_{22}\)  
Return C
Time Analysis

\[ T(1) = 1 \]

\[ T(N) = 7 \, T(N/2) + cN^2 \]

\[ \ldots \]

\[ T(N) = d7^{\log N} = dN^{\log 7} = \Theta(N^{2.81}) \]

The fastest known algorithm for Matrix multiplication runs in \( \Theta(N^{2.38}) \)
Harder problems

Recall: A simple path in a graph $G$ is a path with no cycles.

Consider the following problem (Hamiltonian Path):

Input: an undirected graph $G=(V,E)$
Desired output: **YES** if $G$ contains a simple path of length $n=|V|$, and **NO** if not.

Given a permutation of the nodes, we can easily check if it is a simple path in $G$.

Brute force solution: Check all $n!$ possible permutations of $V$.

**Theorem:** Hamiltonian Path is NP Complete.

No efficient algorithm is known for this problem, and for many other easy to state problems. A solution (or a proof that none exists) is worth $10^6$ in cash and much more... This is the $P=NP$? Problem.
Post’s Correspondence Problem

- **A dominoe**: \( \begin{array}{c}
a \\
ab 
\end{array} \) a string on top and one at bottom

- **A finite set of dominoes**: \{ \[ \begin{array}{c}
b \\
c 
\end{array} \], \[ \begin{array}{c}
a \\
ab 
\end{array} \], \[ \begin{array}{c}
ca \\
a 
\end{array} \], \[ \begin{array}{c}
abc \\
c 
\end{array} \] \}

- **A match**: a list of dominoes from the set (repetitions allowed) for which the top string equals bottom string:
PCP continued

Some sets of dominos cannot yield a match:

\[ \left\{ \left[ \frac{abc}{ab} \right], \left[ \frac{ca}{a} \right], \left[ \frac{acc}{ba} \right] \right\} \]

Post’s correspondence problem (PCP) is:
Input: a finite set of dominoes \( S \)
Desired output: YES if there is a match for \( S \), and NO if not.
PCP continued

**Theorem:** This problem is **undecidable**! No algorithm can answer correctly on all inputs.

Not every problem is computable. Some problems are computable but require exponential time. NP complete problems are easy to verify but are not known to be better than exponential.

Algorithms are central to web search, cryptography, optimization, physics, biology, etc. etc.

There is more to the story…

*More on this in "046002 תכן וניתוח אלגוריתמים" Spring 2010*
סוף

בהצלחה

ולהתראות!!