CSE 102 2-11-20

Continues .... Strassen

Exercise

- Prove this is equivalent to matrix multiplication
- Write pseudo-code (P. 77 CLRS)

real no. mult.

Let T(n) = # of basic operations by this procedure on nxn matrices (where n is exact power of 2.)

Define, basic OP = floating Point (real num.) multiplication.

# of matrix multiplications = 8
each on size $\frac{n}{2} \times \frac{n}{2}$

# of matrix additions = 4
(# of real numb. additions $= n^2$)

$\Rightarrow$

$$T(n) = \begin{cases} 1 & n = 1 \\ 8T\left(\frac{n}{2}\right) + 1 & n \geq 2 \text{ (exact Pow. of 2)} \end{cases}$$

↑ const. cost of dividing & combining. Book has $\Theta(n^2)$ here, since they count additions

By case 1 of M.T.

$$T(n) = \Theta(n^3)$$

no better than ordinary algorithm!

1969: Volker Strassen

found a way to do only 7 sub-matrix multiplications

so now

$$T(n) = \begin{cases} 1 & n = 1 \\ 7T\left(\frac{n}{2}\right) + 1 & n \geq 2 \\ & (n \text{ pow. of } 2) \end{cases}$$

Key case 1:

comp. $n^{\log_2 7}$ to $n^0 = 1$ (or n^2)

$$T(n) = \Theta(n^{\log_2(7)})$$

note: $\log_2 7 = 2.8073\ldots$

so

$$T(n) = \Theta(n^{2.8073\ldots})$$

Exercise

- figure out how to extend to n not an exact pow. of 2.
- figure out how to extend to non-square matrices.
- show we lose nothing in asymptotic runtime.

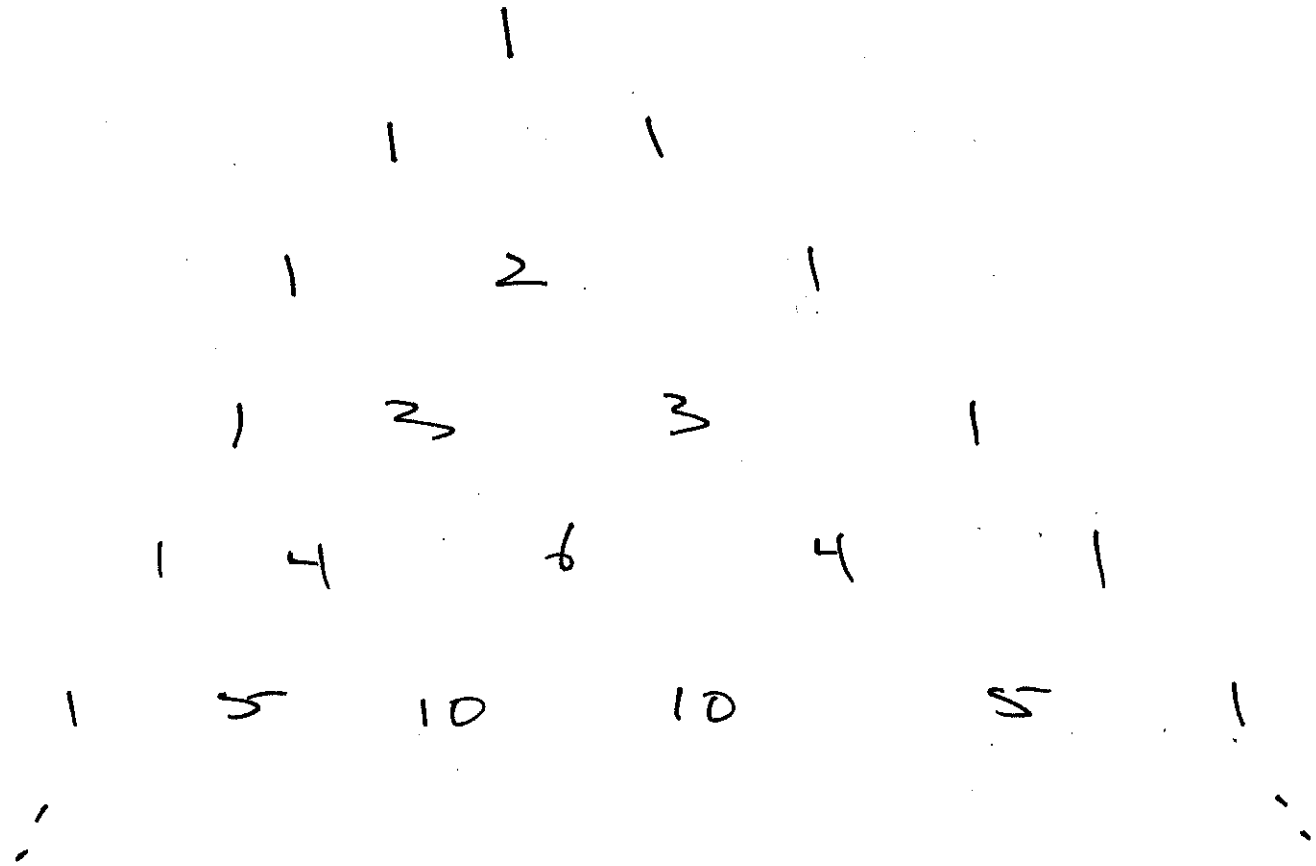
Dynamic Programming

Problem: compute binomial coefficient $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

Pascal's identity: (Pre: $0 \le k \le n$)

$$\binom{n}{k} = \begin{cases} 1 & \text{if } k=0 \text{ or } k=n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{if } 0<k<n \end{cases}$$

Pascal's Δ:



Runtime basic operation: add 1

Runtime $= \binom{n}{k}$

Recall: if $n = 2k$

$\binom{2k}{k} = \Theta\left(\frac{4^k}{\sqrt{k}}\right)$ bad!!

Why?

Why: over-lapping sub-problems

Answer to this Problem:

fill in table, consult table to see if already have value.

- Recursively: Memoization
- Iteratively: Dynamic Programming

Exercise

write memoization version of BinCoef(n, k, ...)

We do Dynamic Prog. (iterative) exclusively.

See DynBinCoef(n,k) on p. 2 of handout.

Dynamic Progamming is often used to solve optimization problems.

Problem Coin Changing

Given n coin denominations

$\{d_1, d_2, \ldots, d_n\}$

where $d_i \geq 1$

Given infinite supply of coins in each denomination

Given an amount N to pay with these coins.

Goal: Pay N with fewest # of coins possible

Actually 2 Problems:

Value of an opt solution

(1) what is least # of coins necessary to pay N units.

(2) Exactly which coins in each denom. must be disbursed to achieve # of coins in (1).

an optimal solution

for question (1), define

$$C[1 \ldots n; 0 \ldots N]$$

where

$C[i, j]$ = min. # of coins necessary to pay j units using only coins from $\{c_1, c_2, \ldots, c_i\}$

with $1 \leq i \leq n$, $0 \leq j \leq N$

we seek $C[n, N]$, which solves (1).

notice that for $C[i, j]$ we have 2 choices

- use no coins of value d_i (even though we can). The least # of coins would be

$$C[i-1, j]$$

- use at least one coin of value d_i. in this case least # of coins is

$$1 + C[i, j-d_i]$$

$C[i,j]$ must be whichever option produces smallest # of coins

$$C[i,j] = \min\left(C[i-1,j],\ 1 + C[i, j-d_i]\right)$$

Note: if either $i=1$ or $j<d_i$ one or both subinstances involve out-of-bounds values.

define out of bounds values to be

$$C[i,j] = +\infty \begin{cases} \text{if } i<1 \\ \text{or} \\ \text{if } j<0 \end{cases}$$