6.006 | Lecture 21 | Apr. 29. 2008

**Outline:** Dynamic Programming III (of 4)

- text justification
- parenthesization
- knapsack
- pseudopolynomial time
- Tetris training

**Reading:** CLRS 15

**Review:**

* DP is all about subproblems & guessing

* 5 easy steps:

① define subproblems; count # subprobs.
② guess (part of solution); count # choices
③ relate subprob. solutions; compute time/subprob.
④ recurse + memoize time = time/subprob.
<u>OR</u> build DP table bottom-up; · # subprobs.
(check subproblems related acyclically
[⑤ check original problem = a subproblem
or solvable from DP table (⇒ extra time)]

* for sequences, good subproblems are often prefixes <u>OR</u> suffixes <u>OR</u> substrings

## Text justification: split text into "good" lines

- obvious (MS Word/OpenOffice) algorithm: put as many words fit on first line, repeat
- but this can make very bad lines:

|  |  |  |  |  |
|---|---|---|---|---|
|  | blah blah blah |  | blah    blah |  |
| ☹ | b    l    a    h | vs. | blah    blah | ☺ |
|  | reallylongword |  | reallylongword |  |

- define badness(i,j) for line of words[i:j]
  e.g. $\begin{cases} \infty & \text{if total length} > \text{page width} \\ (\text{page width} - \text{total length})^3 & \text{else} \end{cases}$
- goal: split words into lines to min. $\sum_i$ badness

① subproblem = min. badness for suffix words[i:]
⇒ # subproblems = $\Theta(n)$ where $n$ = # words

② guessing = where to end first line, say i:j
⇒ # choices = $n - i = O(n)$

③ relation:
- DP[i] = min(badness(i,j) + DP[j]
  for j in range(i+1, n+1))
- DP[n] = Ø

⇒ time per subproblem = $O(n)$

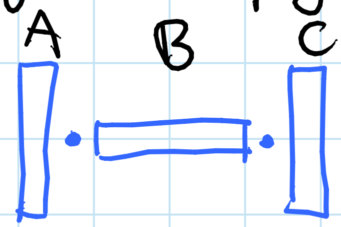
④ total time = $O(n^2)$

⑤ solution = DP[Ø]
(& use parent pointers to recover split)

# Parenthesization:

optimal evaluation of associative expression

- e.g. multiplying rectangular matrices



$(AB)C$ costs $\Theta(n^2)$
$A(BC)$ costs $\Theta(n)$

② guessing = outermost multiplication: $\underset{\uparrow\; k-1}{(\cdots)}\underset{\uparrow\; k}{(\cdots)}$

$\Rightarrow$ # choices = $O(n)$

① subproblems = ~~prefixes & suffixes~~? NO

= cost of substring $A[i:j]$

$\Rightarrow$ # subproblems = $\Theta(n^2)$

③ relation:

- $DP[i,j] = \min(DP[i,k] + DP[k,j] +$ cost of $(A[i]\cdots A[k-1])\cdot(A[k]\cdots A[j-1])$ for k in range(i+1, j))

- $DP[i, i+1] = Ø$

$\Rightarrow$ cost per subproblem = $O(n)$

④ total time = $O(n^3)$

⑤ solution = $DP[0,n]$

(& use parent pointers to recover parens.)

Knapsack of size $S$ you want to pack
- item $i$ has integer size $s_i$ & real value $v_i$
- goal: choose subset of items of max. total value
  subject to total size $\le S$

First attempt:

① ~~subproblem = value for suffix $i$:~~ WRONG

② guessing = whether to include item $i$
$\Rightarrow$ #choices = 2

③ relation:
- $DP[i] = \max(DP[i+1], v_i + DP[i+1]$ if $\cancel{s_i \le S}$ ?!)
- not enough information to know whether item $i$ fits — how much space is left?
  GUESS!

① subproblem = value for suffix $i$:
given knapsack of size $X$
$\Rightarrow$ # subproblems = $O(nS)$ (!)

③ relation:
- $DP[i, X] = \max(DP[i+1, X],$
  $v_i + DP[i+1, X-s_i]$ if $s_i \le X)$
- $DP[n, X] = \emptyset$

$\Rightarrow$ time per subproblem = $O(1)$

④ total time = $O(nS)$

⑤ solution = $DP[\emptyset, S]$
(& use parent pointers to recover subset)

AMAZING: effectively trying all possible subsets!

Knapsack is in fact NP-complete!

⇒ suspect no <u>polynomial-time</u> algorithm

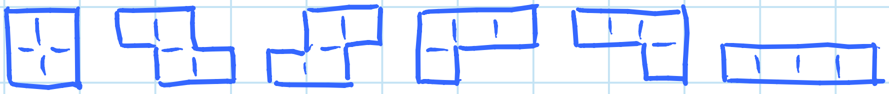
↳ polynomial in <u>length of input</u>

<u>What gives?</u>

- here input $= \langle S, s_0, \ldots, s_{n-1}, v_0, \ldots, v_{n-1} \rangle$
- length in binary: $O(\underbrace{\lg S}_{\text{not } S} + \lg s_0 + \cdots) \approx O(n \lg \cdot)$
- so $O(nS)$ is not "polynomial time"
- $O(nS)$ still pretty good if $S$ is small
- "<u>pseudopolynomial time</u>": polynomial in length of input & integers in the input

<u>Remember:</u> polynomial – GOOD<br>
exponential – BAD<br>
pseudopoly. – SO SO

# Tetris training:



- given sequence of n Tetris pieces & a board of small width w
- must choose orientation & x coordinate for each
- then must drop piece till it hits something
- full rows do not clear

  without these artificialities WE DON'T KNOW!
  (but: if w also large then NP-complete)
- goal: survive i.e. stay within height h

  [material below covered in recitation]

## First attempt:

① ~~subproblem = survive in suffix i: ?~~ WRONG

② guessing = how to drop piece i

⇒ # choices = $O(w)$

③ ~~relation: $DP[i] = DP[i+1]$~~ ?! not enough information!

→ What do we need to know about prefix :i?

① subproblem = survive? in suffix i:

given initial column occupancies $h_0, h_1, \ldots, h_{w-1}$

⇒ # subproblems = $O(n \cdot h^w)$

③ relation: $DP[i, \vec{h}] = \max(DP[i, \vec{m}]$ for valid moves $\vec{m}$ of piece i in $\vec{h})$

⇒ time per subproblem = $O(w)$

④ total time = $O(n\, w\, h^w)$

⑤ solution = $DP[\emptyset, \vec{\emptyset}]$

(& use parent pointers to recover moves)