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.
  or build DP table bottom-up;
     * # subprobs.
  [check subproblems related acyclically]
  [check original problem = a subproblem or solvable from DP table ⇒ extra time]
* for sequences, good subproblems are often prefixes or suffixes or 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:
  ● a b l e h vs. b l a h b l a h
  really.long.word
  ● b l a h b l a h

- define \( \text{badness}(i,j) \) for line of words \([i:j]\)
  e.g. \( \{ \infty \text{ if total length} > \text{page width} \)
  \( \frac{1}{(\text{page width} - \text{total length})^3} \text{ else} \)
- goal: split words into lines to min. \( \Sigma \text{badness} \)

1. subproblem = min. badness for suffix words \([i:]\)
   \[ \Rightarrow \# \text{subproblems} = \Theta(n) \text{ where } n = \# \text{words} \]
2. guessing = where to end first line, say \(i:j\)
   \[ \Rightarrow \# \text{choices} = n - i = O(n) \]
3. relation:
   \[ \text{DP}[i] = \min \{ \text{badness}(i,j) + \text{DP}[j] \} \]
   \[ \text{for } j \text{ in range}(i+1, n+1) \]
   \[ \text{DP}[n] = \emptyset \]
   \[ \Rightarrow \text{time per subproblem} = O(n) \]
4. total time = \(O(n^2)\)
5. solution = \( \text{DP}[\emptyset] \)
   (& use parent pointers to recover split)
Parenthesization:

- Optimal evaluation of associative expression
  - e.g. multiplying rectangular matrices
    \[
    \begin{array}{ccc}
    A & B & C \\
    \end{array}
    \quad (AB)C \quad \text{costs } \Theta(n^3) \\
    A(BC) \quad \text{costs } \Theta(n)
    \end{array}
    \]

2. guessing = outermost multiplication: \((\ldots)(\ldots)\)  
   \(\Rightarrow \) # choices = \(O(n)\)

1. subproblems = prefixes & suffixes? \textcolor{red}{NO}
   \(= \) cost of substring \(A[i:j]\)
   \(\Rightarrow \) # subproblems = \(\Theta(n^2)\)

3. relation:
   \(-\) \(DP[i,j] = \min\{(DP[i,k] + DP[k,j]) + \text{cost of } (A[i] \ldots A[k-1]) \cdot (A[k] \ldots A[j-1])\}\)
   
   \(-\) \(DP[i,i+1] = \emptyset\)

\(\Rightarrow\) 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 $\leq S$

First attempt:
1. subproblem = value for suffix $i$: **WRONG**
2. guessing = whether to include item $i$
   $\Rightarrow$ #choices = 2
3. relation:
   \[-DP[i] = \max(DP[i+1], v_i + DP[i+1] \text{ if } s_i \leq S)?!\]
   - 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:
   \[-DP[i, X] = \max(DP[i+1, X],
     v_i + DP[i+1, X-s_i] \text{ if } s_i \leq X)\]
   \[-DP[n, X] = \emptyset\]
   $\Rightarrow$ 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 polynomial-time algorithm 

G polynomial in length of input

What gives?
- here input = \(<S, s_0, \ldots, s_{n-1}, v_0, \ldots, v_{n-1}\>
- length in binary: \(O(\lg S + \lg s_0 + \cdots) \approx O(n \lg n)\) 
- so \(O(nS)\) is not "polynomial time"
- \(O(nS)\) still pretty good if \(S\) is small
- "pseudopolynomial time": polynomial in length of input & integers in the input

Remember:  
polynomial - GOOD  
exponential - BAD  
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 $i$
   $\Rightarrow$ # choices $= O(w)$
   $\Rightarrow$ What do we need to know about prefix $i$?

1. subproblem = survive? in suffix $i$:
   given initial column occupancies $h_0, h_1, \ldots, h_{w-1}$
   $\Rightarrow$ # subproblems $= O(n \cdot h^w)$
3. relation: $DP[i, \hat{h}] = \max(DP[i, \hat{m}]$ for valid moves $\hat{m}$ of piece $i$ in $\hat{h}$)

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

4. total time $= O(n \cdot w \cdot h^w)$
5. solution $= DP[\emptyset, \emptyset]$
   (& use parent pointers to recover moves)