Dynamic Programming III

Lecture Overview

- Text Justification
- Parenthesization
- Knapsack
- Pseudopolynomial Time
- Tetris Training

Readings

CLRS 15

Review:

* DP is all about subproblems & guessing

* 5 easy steps:

  (a) define subproblems: count $\sharp$ subprobs.
  (b) guess (part of solution): count $\sharp$ choices
  (c) relate subprob. solutions: compute time/subprob.
  (d) recurse + memoize <u>OR</u> build DP table bottom up:
      time = time/subprob. $\times\sharp$ subprobs
      (check subproblems related acyclically)
  (e) check original problem = a subproblem or solvable from DP table ($\Longrightarrow$ 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/Open Office) algorithm: put as many words fit on first line, repeat
- but this can make *very bad* lines

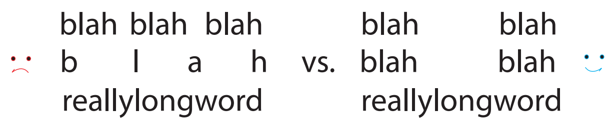

Figure 1: Good vs. Bad Justification

- define <u>badness</u>(i, j) for line of words $[i : j]$ e.g.,

$$\begin{cases} \infty & \text{if total length} > \text{page width} \\ (\text{page width - total length})^3 & \text{else} \end{cases}$$

- <u>goal:</u> split words into lines to min $\sum$ badness

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

Parenthesization:

Optimal evaluation of associative expression - e.g., multiplying rectangular matrices

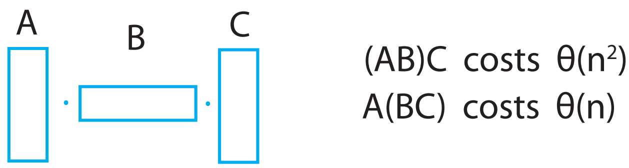

Figure 2: Evaluation of an Expression

2. guessing = outermost multiplication $(\underbrace{\cdots}_{\uparrow k-1})(\underbrace{\cdots}_{\uparrow k})$

$\implies \sharp$ choices $= O(n)$

1. subproblems = ~~prefixes & suffixes?~~ NO
   = cost of substring $A[i:j]$

$\implies \sharp$ subproblems $= \Theta(n^2)$

3. Relation:

   - DP$[i,j]$ = min(DP $[i,k]$ + DP$[k,j]$+ cost of multiplying $(A[i]\cdots A[k-1])$ by $(A[k]\cdots A[j-1])$ for k in range$(i+1,j)$)
   - DP$[i,i+1] = \phi$
     $\implies$ cost per subproblem $= O(n)$

4. total time $= O(n^3)$

5. solution $= DP[0,n]$
   (& use parent pointers to recover parens.)

Knapsack:

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 maximum total value subject to total size $\leq S$

First Attempt:

1. ~~subproblem = value for suffix i:~~ WRONG
2. guessing = whether to include item i $\implies \sharp$ choices $= 2$
3. relation:

- $DP[i] = \max(DP[i+1], v_i + DP[i+1] \text{ if } \cancel{s_i \leq S}?!)$
- 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

$\implies \sharp$ subproblems $= O(nS)$!

3. relation:

   - $DP[i, X] = \max(DP[i+1, X], v_i + DP[i+1, X - s_i] \text{ if } s_i \leq X)$
   - $DP[n, X] = \phi$
     $\implies$ time per subproblem $= O(1)$

4. total time $= O(nS)$

5. solution $= DP[\phi, S]$
   (& use parent pointers to recover subset)
   AMAZING: effectively trying all possible subsets!

Knapsack is in fact NP-complete! $\implies$ suspect no polynomial-time algorithm (polynomial in length of input).

What gives?

- here input $=< S, s_0, \cdots, s_{n-1}, v_0, \cdots, v_{n-1} >$
- length in binary: $O(\lg S + \lg s_0 + \cdots) \approx O(n \lg \ldots)$
- 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

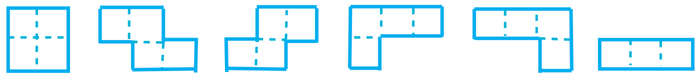

Figure 3: Tetris

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 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 $\implies$ $\sharp$ 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 $h_0, h_1, \cdots, h_{w-1}$
$\implies$ $\sharp$ subproblems $= O(n \cdot h^w)$

3. relation: $DP[i, \boldsymbol{h}] = \max(DP[i, \boldsymbol{m}]$ for valid moves $\boldsymbol{m}$ of piece i in $\boldsymbol{h})$
$\implies$ time per subproblem $= O(w)$

4. total time $= O(nwh^w)$

5. solution $= DP[\phi, \overline{\phi}]$
(& use parent pointers to recover moves)

Source: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-spring-2008/lecture-notes/lec21.pdf