Lecture 21: Dynamic Programming III

Lecture Overview

- Subproblems for strings
- Parenthesization
- Edit distance (& longest common subseq.)
- Knapsack
- Pseudopolynomial Time

Review:

* 5 easy steps to dynamic programming

(a) define subproblems — count # subproblems

(b) guess (part of solution) — count # choices

(c) relate subproblem solutions — compute time/subproblem

(d) recurse + memoize — time = time/subproblem · # subproblems
OR build DP table bottom-up
check subproblems acyclic/topological order

(e) solve original problem: = a subproblem
OR by combining subproblem solutions $\Longrightarrow$ extra time

* problems from L20 (text justification, Blackjack) are on sequences (words, cards)

* useful problems for strings/sequences x:

$$\left.\begin{array}{l}\text{suffixes } x[i:] \\ \text{prefixes } x[:i]\end{array}\right\} \Theta(|x|) \quad \leftarrow \text{cheaper} \Longrightarrow \text{use if possible}$$

$$\text{substrings } x[i:j] \;\Big\}\; \Theta(x^2)$$

Parenthesization:

Optimal evaluation of associative expression $A[0] \cdot A[1] \cdots A[n-1]$ — e.g., multiplying rectangular matrices

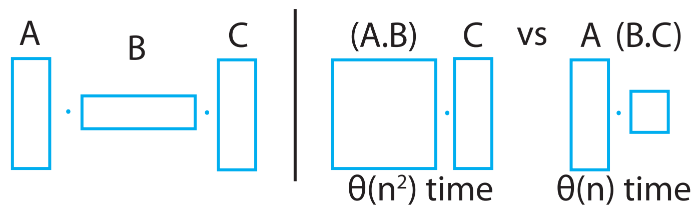


Figure 1:

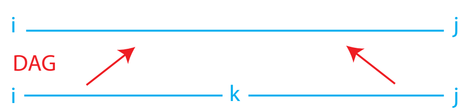
2. guessing = outermost multiplication $\underbrace{(\cdots)}_{\uparrow k-1}\underbrace{(\cdots)}_{\uparrow k}$

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

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

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

3. recurrence:
   - $\text{DP}[i,j] = \min(\text{DP}[i,k] + \text{DP}[k,j] +$ cost of multiplying $(A[i] \cdots A[k-1])$ by $(A[k] \cdots A[j-1])$ for k in range$(i+1,j))$



   - $\text{DP}[i,i+1] = 0$
     $\Longrightarrow$ cost per subproblem $= O(j-i) = O(n)$

4. topological order: increasing substring size. Total time $= O(n^3)$

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

NOTE: Above DP is not shortest paths in the subproblem DAG! Two dependencies $\Longrightarrow$ not path!

Edit Distance

Used for DNA comparison, diff, CVS/SVN/..., spellchecking (typos), plagiarism detection, etc.

Given two strings x & y, what is the cheapest possible sequence of character edits (insert c, delete c, replace c $\rightarrow$ c') to transform x into y?

- cost of edit depends only on characters c, c'
- for example in DNA, C $\rightarrow$ G common mutation $\implies$ low cost
- cost of sequence = sum of costs of edits
- If insert & delete cost 1, replace costs 0, minimum edit distance equivalent to finding longest common subsequence. Note that a subsequence is sequential but not necessarily contiguous.
- for example H I E R O G L Y P H O L O G Y vs. M I C H A E L A N G E L O $\implies$ HELLO

Subproblems for multiple strings/sequences

- combine suffix/prefix/substring subproblems
- multiply state spaces
- still polynomial for $O(1)$ strings

Edit Distance DP

(1) subproblems: $c(i, j) = \text{edit-distance}(x[i:], y[j:])$ for $0 \leq i < |x|, 0 \leq j < |y|$
$\implies \Theta(|x| \cdot |y|)$ subproblems

(2) guess whether, to turn x into y, (3 choices):

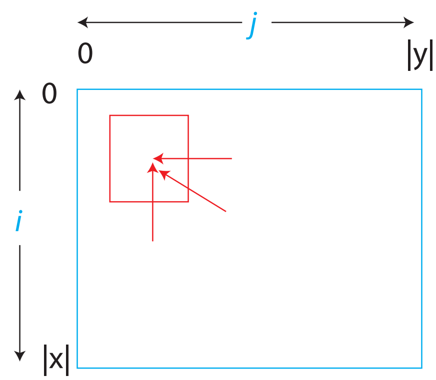
- $x[i]$ deleted
- $y[j]$ inserted
- $x[i]$ replaced by $y[j]$

(3) recurrence: $c(i, j)$ = maximum of:

- $\text{cost}(\text{delete } x[i]) + c(i+1, j)$ if $i < |x|$,
- $\text{cost}(\text{insert } y[j]) + c(i, j+1)$ if $j < |y|$,
- $\text{cost}(\text{replace } x[i] \rightarrow y[j]) + c(i+1, j+1)$ if $i < |x| \& j < |y|$

base case: $c(|x|, |y|) = 0$
$\implies \Theta(1)$ time per subproblem

(4) topological order: DAG in 2D table:



- bottom-up OR right to left
- only need to keep last 2 rows/columns
  $\implies$ linear space
- total time $= \Theta(|x| \cdot |y|)$

(5) original problem: $c(0, 0)$

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 $i \implies$ # choices = 2
3. recurrence:
   - $DP[i] = \max(DP[i+1], v_i + DP[i+1]$ if $\cancel{s_i \leq S}$?!)
   - not enough information to know whether item i fits — how much space is left? GUESS!

Correct:

1. subproblem = value for suffix i:
   given knapsack of size X

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

3. recurrence:
   - $DP[i, X] = \max(DP[i+1, X], v_i + DP[i+1, X - s_i] \text{ if } s_i \leq X)$
   - $DP[n, X] = 0$
     $\implies$ time per subproblem $= O(1)$
4. topological order: for i in $n, \ldots, 0$: for X in $0, \ldots S$

   total time $= O(nS)$
5. original problem $= DP[0, S]$
   (& use parent pointers to recover subset)

AMAZING: effectively trying all possible subsets! ... but is this actually fast?

Polynomial time

Polynomial time = polynomial in input size

- here $\Theta(n)$ if number S fits in a word
- $O(n \lg S)$ in general
- S is exponential in $\lg S$ (not polynomial)

Pseudopolynomial Time

Pseudopolynomial time = polynomial in the problem size AND the numbers (here: S, s_i's, v_i's) in input. $\Theta(nS)$ is pseudopolynomial.

Remember:
polynomial — GOOD
exponential — BAD
pseudopoly — SO SO