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) choose (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 ⇒ extra time

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

* useful problems for strings/sequences $x$:
  suffixes $x[i:]$ \quad \{ \Theta(|x|) \} \quad \leftarrow \text{cheaper} \quad \Rightarrow \text{use if possible}
  prefixes $x[:i]$  
  substrings $x[i:j]$ \quad \{ \Theta(x^2) \}
Parenthesization:

Optimal evaluation of associative expression $A[0] \cdot A[1] \cdots A[n-1]$ — e.g., multiplying rectangular matrices. Multiplying a $p \times q$ matrix by a $q \times r$ matrix takes $\Theta(pqr)$ time (using the simple algorithm)

![Figure 1:](image)

For example, if $A$ is $100 \times 1$, $B$ is $1 \times 100$, and $C$ is $100 \times 1$, then computing $A(BC)$ takes time proportional to $10^2$ while $(AB)C$ takes time proportional to $10^4$.

2. guessing = outermost multiplication

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

1. subproblems = prefixes & suffixes? NO

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

3. recurrence:

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

- $DP[i, i+1] = 0$

$\implies$ cost per subproblem = $O(j - i) = O(n)$

We can also record, for each $(i, j)$, the optimal choice of $k$ that achieves the minimum above.

4. topological order: increasing substring size. Total time = $O(n^3)$
5. original problem = $DP[0, n]$
   (& use the recorded optimal $k$’s to recover parens.)

NOTE: Above DP is not shortest paths in the subproblem DAG! Two dependencies $\implies$ 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 $\infty$ (for $c \neq c'$), 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: $c(i, j) = \text{minimum of:}$

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

base case: \( c(|x|, |y|) = 0 \)

\[ \Rightarrow \Theta(1) \text{ time per subproblem} \]

can also record which of these three choices achieves the maximum

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

- bottom-up OR right to left
- only need to keep last 2 rows/columns
  \[ \Rightarrow \text{linear space} \]
- 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 \( i \) \[ \Rightarrow \# \text{ choices} = 2 \]
3. recurrence:
   - \( DP[i] = \max(DP[i + 1], v_i + DP[i + 1] \text{ if } s_i \leq S?!) \)
not enough information to know whether item $i$ fits — how much space is left? GUESS!

**Correct:**

( Assumes $S$ is an integer.)

1. subproblem = value for suffix $i$:
   
   $$
   \begin{align*}
   \text{given knapsack of size } X \\
   \implies \# \text{ subproblems } &= O(nS) \\
   \end{align*}
   $$

3. recurrence:
   
   $$
   \begin{align*}
   \text{DP}[i, X] &= \max(\text{DP}[i + 1, X], v_i + \text{DP}[i + 1, X - s_i] \text{ if } s_i \leq X) \\
   \text{DP}[n, X] &= 0 \\
   \implies \text{ time per subproblem } &= O(1)
   \end{align*}
   $$

4. topological order: for $i$ in $n, \ldots, 0$: for $X$ in $0, \ldots S$

   total time $= O(nS)$

5. original problem $= \text{DP}[0, S]$

   (& 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