Today: Dynamic Programming III (of 4)
- subproblems for strings
- parenthesization
- edit distance (& longest common subseq.)
- knapsack
- pseudopolynomial time

* 5 easy steps to dynamic programming:
  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.
  5. build DP table bottom-up
* # subprobs.
  - check subprobs: acyclic/topological order
  - solve original problem: = a subproblem
  or by combining subprob. solutions (⇒ extra time)

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

* useful subproblems for strings/sequences x:
  - suffixes x[i::]
    \( \Theta(|x|) \) cheaper ⇒ use if possible
  - prefixes x[:i]
  - substrings x[i:j]
    \( \Theta(|x|^2) \)
- e.g. multiplying rectangular matrices

$$\begin{array}{ccc}
A & B & C \\
\cdot & \cdot & \cdot \\
\end{array} \quad \begin{array}{ccc}
(A \cdot B) \cdot C & \text{vs.} & A \cdot (B \cdot C) \\
\Theta(n^2) \text{ time} & & \Theta(n) \text{ time} \\
\end{array}$$

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

1. subproblems = prefixes & suffixes? NO
   $\Rightarrow$ # subproblems = $\Theta(n^2)$

3. recurrence:
   $- DP[i, j] = \min \left( DP[i, k] + DP[k, j] + \text{cost of} \right.
   \left. (A[i] \cdots A[k-1]) \cdot (A[k] \cdots A[j-1]) \right)$
   for $k$ in range($i+1, j$)
   $- DP[i, i+1] = \emptyset$
   $\Rightarrow$ cost per subproblem = $\Theta(j-i) = O(n)$

4. topological order: increasing substring size
   total time = $\Theta(n^3)$

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 ⇒ not path!

**Edit distance:** (used for DNA comparison, diff, CVS/SVN/..., spellchecking (typos), plagiarism detection, etc.)

given two strings $x$ & $y$, what's the cheapest possible sequence of character edits to transform $x$ into $y$?

- Cost of edit depends only on characters $c, c'$
  - e.g. in DNA, $C \rightarrow G$ common mutation ⇒ low cost
  - Cost of sequence = sum of costs of edits

- if insert & delete cost 1, replace costs $\phi$
- Min edit distance equivalent to finding longest common subsequence
  sequential but not necessarily contiguous

- e.g.: **HIEROGLYPH OLOGY**
  vs. **MICHAELANGELO**
  ⇒ **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 \( 0 \leq i < |x|, 0 \leq j < |y| \)
   \( \Rightarrow \Theta(|x| \cdot |y|) \) subproblems

2. guess whether to turn \( x \) into \( y \):
   - \( x[i] \) deleted
   - \( y[j] \) inserted
   - \( x[i] \) replaced by \( y[j] \)
   \( \Rightarrow 3 \) choices

3. recurrence: \( c(i, j) = \max \{ \)
   - \( \text{cost}(\text{delete } x[i]) + c(i+1, j) \) 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 \( i < |x| \) & \( j < |y| \)\}
   \( \Rightarrow \Theta(1) \) time per subproblem

4. topological order: DAG in 2D table:
   - bottom up or right to left
   - only need to keep last 2 rows/columns
   \( \Rightarrow \) linear space
   \( \Rightarrow \) total time = \( \Theta(|x| \cdot |y|) \)

5. original problem: \( c(\emptyset, \emptyset) \)
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 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?

Correct:
① subproblem = value for suffix $i$: **given knapsack of size $X$**
  $\Rightarrow$ #subproblems = $O(nS)$ (!)
③ recurrence:
  - $DP[i,X] = \max(DP[i+1,X], v_i + DP[i+1,X-s_i] \text{ if } s_i \leq X)$
  - $DP[n,X] = \emptyset$
  $\Rightarrow$ time per subproblem = $O(1)$
④ topological order: for $i$ in $n, \cdots, 0$: for $X$ in $0, \cdots, S$
  - total time = $O(nS)$
⑤ original problem = $DP[\emptyset, S]$
($&$ use parent pointers to recover subset)
Polynomial time = polynomial in input size
- here $O(n)$ if number $S$ fits in a word
- $O(n \log S)$ in general
- $S$ is exponential in $\log S$ (not polynomial)

Pseudopolynomial time = polynomial in the problem size AND the numbers in input
- $\Theta(nS)$ is pseudopolynomial

Remember: polynomial - GOOD
exponential - BAD
pseudopoly. - SO SO