Dynamic Programming ← “program” as in “exercise program”, not “computer program”

Recall Fibonacci

**recursive**

\[
f(n) = \begin{cases} 
  0 & \text{if } n = 0 \\
  1 & \text{if } n = 1 \\
  f(n-1) + f(n-2) & \text{else}
\end{cases}
\]

Let \( T(n) \) be the number of integer additions (not bit complexity).

\[ T(n) = T(n-1) + T(n-2) + 1 \]

\( T(n) \) grows like the Fibonacci numbers. BAD!

**iterative**

\[
f(0) := 0 \\
f(1) := 1 \\
\text{for } i \text{ from } 2 \text{ to } n \text{ do} \\
  f(i) := f(i-1) + f(i-2)
\]

\( n - 1 \) integer additions. GOOD!

An example of dynamic programming.

Main idea of dynamic programming:

solve “subproblems” from smaller to larger (bottom up) storing solutions.

Run-time:

\[
(\# \text{ subproblems}) \times (\text{time to solve one subproblem})
\]
Text Segmentation

Given a string of letters $A[1 \ldots n]$, $A[i] \in \{A, B, \ldots, Z\}$, can you split into words?

Assume you have a test

$$\text{Word}[i, j] = \begin{cases} 
\text{True} & \text{if } A[i \ldots j] \text{ is a word} \\
\text{False} & \text{otherwise} 
\end{cases}$$

where each call takes $O(1)$

Example: THEMEMPTY splits into THEM EMPTY

Note: a greedy solution might try to find

- the shortest word $A[1 \ldots i]$ (prefix) : THEMEMPTY wrong
- or the longest word $A[1 \ldots i]$ : THEMEMPTY MPTY wrong

Can we do something like Fibonacci? Suppose we knew

$$\text{Split}[k] = \begin{cases} 
\text{True} & \text{if } A[1 \ldots k] \text{ is splittable} \\
\text{False} & \text{otherwise} 
\end{cases} \quad \text{for } k = 0 \ldots n - 1$$

Can we then find $\text{Split}[n]$? Try $\text{Split}[j]$ and $\text{Word}[j + 1, n]$ for all $j = 0 \ldots n - 1$.

Claim: $\text{Split}[n] \iff \text{at least one } j \text{ gives True. Why?}$

$\iff$ we have a way to split $A[1 \ldots n]$

$\Rightarrow$ if $A[1 \ldots n]$ is splittable, take $A[j + 1 \ldots n]$ as last word
Resulting algorithm:

\[
\text{Resulting algorithm:} \\
\begin{align*}
\text{Split}[0] & := \text{True} \\
\text{for } k \text{ from } 1 \text{ to } n \text{ do} \\
& \quad \text{Split}[k] := \text{False} \\
& \quad \text{for } j \text{ from } 0 \text{ to } k - 1 \text{ do} \\
& \quad \quad \text{if Split}[j] \text{ and Word}[j + 1, k] \text{ then} \\
& \quad \quad \quad \text{Split}[k] := \text{True}
\end{align*}
\]

Run-time: \( O(n^2) \)

Exercise: Show how to compute the actual split.
Longest Increasing Subsequence

Given a sequence of numbers, \( A[1 \ldots n] \), \( A[i] \in \mathbb{N} \), find the longest increasing subsequence.

Example: \( 5 \ 2 \ 1 \ 4 \ 3 \ 1 \ 6 \ 9 \ 2 \) increasing subsequence of length 4

Following previous approach, what if we set

\[ \text{LIS}[k] = \text{length of longest increasing subsequence of } A[1 \ldots k]? \]

This does not seem to give enough info to get \( \text{LIS}[n] \) from previous \( \text{LIS}[k]'s \).

\( \rightarrow \) need to see if \( A[n] \) is large enough to add to a previous sequence

Better Idea: Let \( \text{LISE}[k] = \text{length of longest increasing subsequence of } A[1..k] \)

that ends with \( A[k] \).

Algorithm

\[
\text{LISE}[1] := 1 \\
\text{for } k \text{ from } 2 \text{ to } n \text{ do} \\
\quad \text{LISE}[k] := 1 \\
\quad \text{for } j \text{ from } 1 \text{ to } k - 1 \text{ do} \\
\quad \quad \text{if } A[k] > A[j] \text{ then} \\
\quad \quad \quad \text{LISE}[k] := \max \{ \text{LISE}[k], \text{LISE}[j] + 1 \}
\]

Exercise: Argue correctness

Run-time: \( O(n^2) \)
Example:

\[
\begin{array}{cccccccc}
A &=& 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
LIS_e &=& 1 & 1 & 1 & 2 & 2 & 2 & 1 & \\
\text{coming from} &=& 0 & 0 & 0 & 0 & 2 & 2 & 0 & \\
\end{array}
\]

Run-time: \( O(n^2) \)

How do we get the final answer? Two possibilities:

- maximum entry in \( LIS_e \), or
- add dummy entry \( A[n + 1] = +\infty \) and return \( LIS_e[n + 1] - 1 \).

Note: there is an \( O(n \log n) \) time algorithm
Longest Common Subsequence

Recall pattern matching from CS 240:
Given a long string $T$ and short pattern $P$ find occurrences of $P$ in $T$.
Useful in grep, find, etc.

Also useful: given two long strings find longest common subsequence

$x = \text{T A R M A C}$
$y = \text{C A T A M A R A N}$

Note that we can skip letters in both strings, but must preserve ordering.

Given $x_1 \ldots x_n$ and $y_1 \ldots y_m$, let $M(i, j) =$ length of longest common subsequence of

$x_1 \ldots x_{i-1}x_i$ and $y_1 \ldots y_{j-1}y_j$.

How can we solve this subproblem based on solutions to “smaller” subproblems?

Choices: (1) match $x_i = y_j$; (2) skip $x_i$; (3) skip $y_j$

$M(i, 0) = 0$
$M(0, j) = 0$
$M(i, j) = \max \begin{cases} 1 + M(i - 1, j - 1) & \text{if } x_i = y_j \\ M(i - 1, j) \\ M(i, j - 1) \end{cases}$

Solve subproblems in any order with $M(i-1, j-1), M(i-1, j), M(i, j-1)$ before $M(i, j)$
<table>
<thead>
<tr>
<th></th>
<th>y</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>∅</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>A</td>
<td>M</td>
<td>A</td>
<td>R</td>
<td>A</td>
<td>R</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>∅</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>M</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>C</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

for $i = 0 \ldots n$: $M(i, 0) := 0$

for $j = 0 \ldots m$: $M(0, j) := 0$

for $i = 1 \ldots n$

for $j = 1 \ldots m$

$$M(i, j) := \max \left\{ \begin{array}{ll} 1 + M(i - 1, j - 1) & \text{if } x_i = y_j \\ M(i - 1, j) & \\ M(i, j - 1) & \end{array} \right.$$ 

Note that this is a correct ordering of $i$ and $j$.

In fact, if $x_i = y_j$ we can use the first choice (no need to check max of other two choices).
Run-time: $O(n \cdot m \cdot c)$

# of subproblems $\Rightarrow$ time to solve one subproblem (compare three possibilities)

To find the actual max. common subsequence: work backwards from $M(n, m)$.

→ Call OPT($n, m$).

\[
\text{OPT}(i, j) \quad \# \text{recursive routine}
\]

\[
\begin{align*}
\text{if } i = 0 \text{ or } j = 0 \text{ then done} \\
\text{if } M(i, j) = M(i - 1, j) \text{ then} \\
\quad \text{OPT}(i - 1, j) \\
\text{elif } M(i, j) = M(i, j - 1) \text{ then} \\
\quad \text{OPT}(i, j - 1) \\
\text{else} \quad \# \text{we must have matched } i \text{ and } j \\
\quad \text{output } i, j \\
\quad \text{OPT}(i - 1, j - 1)
\end{align*}
\]

Or we can record, when we fill $M(i, j)$, where the max comes from.

Next day: more sophisticated “edit” distance between strings.
Maximum common subsequence solves

Longest increasing subsequence

\[ L = 5 \ 2 \ 9 \ 6 \ 3 \ 7 \ 4 \]
\[ S = 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 9 \]

increasing subsequence of length 3

Note: \( S = \text{sorted} \ L \)

Claim: Longest increasing subsequence of \( L = \text{max common subsequence of} \ L \ \text{and} \ S. \)