Dynamic Programming  

Recall Fibonacci

**recursive**

\[
\begin{align*}
    f(n) &= \text{if } n = 0 \text{ then return } 0 \\
            &\quad \text{elif } n = 1 \text{ then return } 1 \\
            &\quad \text{else return } f(n-1) + f(n-2) \\
\end{align*}
\]

\[T(n) = T(n-1) + T(n-2) + c\] so runtime grows like the Fibonacci numbers. **BAD!**

**iterative**

\[
\begin{align*}
    f(0) &:= 0 \\
    f(1) &:= 1 \\
    \text{for } i \text{ from } 2 \text{ to } n \text{ do} \\
        f(i) &:= f(i-1) + f(i-2) \\
    \text{od}
\end{align*}
\]

\[O(n)\] arithmetic operations. **GOOD!**

- an example of dynamic programming

Main idea of dynamic programming:

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

Runtime: \((\# \text{ subproblems}) \times (\text{time to solve one subproblem})\)
Text segmentation

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

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

where each call takes $O(1)$

e.g., THEMEMPTY splits into THEM EMPTY

Note: a greedy solution might try to find

- the shortest word $A[1..i]$ (prefix): THE MEMPTY wrong
- or the longest word $A[1..i]$: THEME MPTY wrong

Can we do something like Fibonacci? Suppose we knew

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

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

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

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

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

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

Runtime: $O(n^2)$

Ex. Show how to compute the actual split
Longest Increasing Subsequence

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

e.g., 5 2 1 4 3 1 6 9 2

Following previous approach, what if we set

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

This does not seem to give enough info to get $LIS[n]$ from previous $LIS[k]$’s.

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

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

that ends with $A[k]$.

Algorithm

$LISe[1] := 1$

for $k$ from 2 to $n$ do

$LISe[k] := 1$

for $j$ from 1 to $k - 1$ do

if $A[k] > A[j]$ then

$LISe[k] := \max\{LISe[k], LISe[j] + 1\}$

fi

od

od

Ex. Argue correctness

Runtime $O(n^2)$
### Example

<table>
<thead>
<tr>
<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>5</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>2</td>
</tr>
</tbody>
</table>

### LIS\(e\):

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Running time**: \(O(n^2)\)
- How do we get the final answer?
  - 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

$\text{T A R M A C}$

$\text{C A T A M A R A N}$

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

Given strings $x_1 \ldots x_n$ and $y_1 \ldots y_m$,

Let $M(i, j) = \text{length of longest common subsequence of } x_1 \cdots x_{i-1}x_i \text{ and } y_1 \cdots y_{j-1}y_j$.

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

Choices: match $x_i = y_j$, skip $x_i$, skip $y_j$

\begin{align*}
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}
\end{align*}

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>1</th>
<th>2</th>
<th>3</th>
<th>4</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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>∅</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>⬜</td>
<td>⬜</td>
<td>1</td>
<td>⬜</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>0</td>
<td>⬜</td>
<td>1</td>
<td>⬜</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>R</td>
<td>0</td>
<td>⬜</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

for $i = 0..n$: $M(i, 0) := 0$
for $j = 0..m$: $M(0, j) := 0$
for $i = 1..n$
for $j = 1..m$

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

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).
Runtime: \( O(n \cdot m \cdot c) \)

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

\( \rightarrow \) Call \( \text{OPT}(n, m) \).

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

\[
\text{if } i = 0 \text{ or } j = 0 \text{ then done fi}
\]

\[
\text{if } M(i, j) = M(i - 1, j) \text{ then}
\]

\[
\text{OPT}(i - 1, j)
\]

\[
\text{elif } M(i, j) = M(i, j - 1) \text{ then}
\]

\[
\text{OPT}(i, j - 1)
\]

\[
\text{else} \quad \text{— we must have matched } i \text{ and } j
\]

\[
\text{output } i, j
\]

\[
\text{OPT}(i - 1, j - 1)
\]

\[
\text{fi}
\]

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

$L = 5 \ 2 \ 9 \ 6 \ 3 \ 7 \ 4$

increasing subsequence of length 3

$S = 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 9$

$S = \text{sort } L$

Claim: Longest increasing subsequence of $L = \text{maximum common subsequence of } L \text{ and } S$. 